Statistics for Linguistics with R

Statistics for Linguistics with R

A Practical Introduction

2nd revised edition

by
Stefan Th. Gries

De Gruyter Mouton

ISBN 978-3-11-030728-3
e-ISBN 978-3-11-030747-4

Library of Congress Cataloging-in-Publication Data

A CIP catalog record for this book has been applied for at the Library of Congress.

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.dnb.de.

Cover image: section from visual tool © Stefan Th. Gries
Printing: Hubert & Co. GmbH & Co. KG, Göttingen
♾ Printed on acid-free paper

Printed in Germany

www.degruyter.com

To Pat, the most supportive
Department Chair I could have ever wished for.

Preface

This book is a revised and extended version of Gries (2009b). There are four main types of changes. The first and most important one is a complete overhaul of Chapter 5. After having taught dozens of workshops and bootcamps on statistics in linguistics with R, I realized that the most difficult aspects of regression modeling for beginners are (i) to understand the logic of the modeling process, (ii) how to interpret the numerical results (esp. with different contrasts), and (iii) how to visualize them revealingly. Thus, all sections on regression modeling have been rewritten from scratch. In addition, there is now an overview of more theoretical aspects of modeling that, hopefully, will make many things easier to understand.

The second set of changes is concerned with Chapter 1 and Chapter 4. In the former, I now discuss the notions of one- and two-tailed tests in a better way; in the latter, I discuss a set of questions and a visualization tool that should help choosing the right statistical tests for a particular study.

Third, I have added a small section on programming aspects and on how users can write their own functions and, now that that is explained, also make a few very small functions that I have written for myself available to the readers.

Then, this edition not only corrects errors that readers have reported to me (and I am very grateful for them to take the time to do so and hope I haven't added too many new ones …) but it also adds a multitude of small tweaks and changes that arose out of the statistics workshops and classes I have taught over the last few years. Some of these tweaks are in the book, but many are also 'hidden' in the code file so you will only see them if you – as you should – work your way through this book using the code all the time. Finally, all code from the book is now in one file, which will make handling the code and looking up functions much convenient and means that an index of function names is not useful anymore.

I hope you will enjoy, and benefit from, this book and the many changes that went into this revision. As usual, I would like to thank the team at De Gruyter Mouton who supported, in fact raised, the idea of a second edition very early on. Also, again thanks are due to the R Development Core Team and many contributors to bugfixes and packages for R and, also again, to R. Harald Baayen for exposing me to R the first time; I cannot imagine what my research would look like had he not done that …

Contents

Chapter 5

Chapter 1
Some fundamentals of empirical research

> When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind. It may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.
>
> William Thomson, Lord Kelvin.
> (<http://hum.uchicago.edu/~jagoldsm/Webpage/index.html>)

1. Introduction

This book is an introduction to statistics. However, there are already very many introductions to statistics – why do we need another one? Just like the first edition, this book is different from many other introductions to statistics in several ways:

- it has been written especially for linguists: there are many introductions to statistics for psychologists, economists, biologists etc., but only very few which, like this one, explain statistical concepts and methods on the basis of linguistic questions and for linguists;
- it explains how to do most of the statistical methods both 'by hand' as well as with statistical software, but it requires neither mathematical expertise nor hours of trying to understand complex equations – many introductions devote much time to mathematical foundations (and, thus, make everything more difficult for the novice), others do not explain any foundations and immediately dive into some nicely designed software, which often hides the logic of statistical tests behind a nice GUI;
- it not only explains statistical concepts, tests, and graphs, but also the design of tables to store and analyze data, summarize previous literature, and some very basic aspects of experimental design;
- it only uses open source software (mainly R): many introductions use SAS or in particular SPSS, which come with many disadvantages such that (i) users must buy expensive licenses that are restricted in how many functions they offer and how many data points they can handle)

and how long they can be used; (ii) students and professors may be able to use the software only on campus; (iii) they are at the mercy of the software company with regard to bugfixes and updates etc.;
- it does all this in an accessible and informal way: I try to avoid jargon wherever possible; the use of software will be illustrated in very much detail, and there are think breaks, warnings, exercises (with answer keys on the companion website), and recommendations for further reading etc. to make everything more accessible.

So, this book aims to help you do scientific quantitative research. It is structured as follows. Chapter 1 introduces the foundations of quantitative studies: what are variables and hypotheses, what is the structure of quantitative studies and what kind of reasoning underlies it, how do you obtain good experimental data, and in what kind of format should you store your data?

Chapter 2 provides an overview of the programming language and environment R, which will be used in all other chapters for statistical graphs and analyses: how do you create, load, and manipulate data to prepare for your analysis?

Chapter 3 explains fundamental methods of descriptive statistics: how do you describe your data, what patterns can be discerned in them, and how can you represent such findings graphically? Chapter 4 explains fundamental methods of analytical statistics: how do you test whether the obtained results actually mean something or have just arisen by chance? Chapter 5 introduces several multifactorial procedures, i.e. procedures, in which several potential cause-effect relations are investigated simultaneously. While this chapter will teach you a lot of things, Ican only deal with a few selected methods and will point you to additional references quite a few times.

Apart from the following chapters with their think breaks and exercises etc., the companion website for this book at <http://tinyurl.com/StatForLingWithR> is an important resource. You will have to go there anyway to download exercise files, data files, answer keys, errata etc., but at <http://groups.google.com/group/statforling-with-r> you will also find a newsgroup "StatForLing with R". I would like to encourage you to become a member of that newsgroup so that you can

- ask questions about statistics for linguists (and hopefully also get an answer from some kind soul);
- send suggestions for extensions and/or improvements or data for additional exercises;

– inform me and other readers of the book about bugs you find (and of course receive such information from other readers). This also means that if R commands, or *code*, provided in the book differs from that on the website, then the latter is most likely going to be correct.

Lastly, I have to mention one important truth right at the start: you cannot learn to do statistical analyses by reading a book about statistical analyses. You must *do* statistical analyses. There is no way that you read this book (or any other serious introduction to statistics) 15 minutes in bed before turning off the light and learn to do statistical analyses, and book covers or titles that tell you otherwise are, let's say, 'distorting' the truth for marketing reasons. I strongly recommend that, as of the beginning of Chapter 2, you work with this book directly at your computer with R running (ideally in RStudio) so that you can immediately enter the R code that you read and try out all relevant functions from the code files from the companion website; often (esp. in Chapter 5), the code files for this chapter will provide you with a lot of (!) important extra information, additional code snippets, further suggestions for explorations using graphs etc., and sometimes the exercise files will provide even more suggestions and graphs. Even if you do not understand every aspect of the code right away, this will still help you to learn all this book tries to offer.

2. On the relevance of quantitative methods in linguistics

Above I said this book introduces you to scientific quantitative research. But then, what are the goals of such research? Typically, one distinguishes three goals, which need to be described because (i) they are part of a body of knowledge that all researchers within an empirical discipline should be aware of and (ii) they are relevant for how this book is structured.

The first goal is the *description* of your data on some phenomenon and means that your data and results must be reported as accurately and revealingly as possible. All statistical methods described below will help you achieve this objective, but particularly those described in Chapter 3.

The second goal is the *explanation* of your data, usually on the basis of hypotheses about what kind(s) of relations you expected to find in the data. On many occasions, this will already be sufficient for your purposes. However, sometimes you may also be interested in a third goal, that of *prediction*: what is going to happen in the future or when you look at different

data. Chapters 4 and 5 will introduce you to methods to pursue these goals of explanation and prediction.

When you look at these goals, it may appear surprising that statistical methods were not in widespread use in linguistics for decades. This is all the more surprising because such methods are very widespread in disciplines with similarly complex topics such as psychology, sociology, economics. To some degree, this situation is probably due to how linguistics has evolved over the past decades, but fortunately this has changed remarkably in the recent decade. The number of studies utilizing quantitative methods has been increasing (in all linguistic sub-disciplines); the field is experiencing a paradigm shift towards more empirical methods. Still, even though such methods are commonplace in other disciplines, they still often meet some resistance in linguistic circles: statements such as "we've never needed something like that before" or "the really interesting things are qualitative in nature anyway and are not in need of any quantitative evaluation" or "I am a field linguist and don't need any of this" are far from infrequent.

Let me say this quite bluntly: such statements are not particularly reasonable. As for the first statement, it is not obvious that such quantitative methods were not needed so far – to prove that point, one would have to show that quantitative methods could impossibly have contributed something useful to previous research, a rather ridiculous point of view – and even then it would not necessarily be clear that the field of linguistics is not *now* at a point where such methods are useful. As for the second statement, in practice quantitative and qualitative methods go hand in hand: qualitative considerations precede and follow the results of quantitative methods anyway. To work quantitatively does not mean to just do, and report on, some number-crunching – of course, there must be a qualitative discussion of the implications – but as we will see below often a quantitative study allows to identify what merits a qualitative discussion in the first place. As for the last statement: even a descriptive (field) linguist who is working to document a near-extinct language can benefit from quantitative methods. If the chapter on tense discusses whether the choice of a tense is correlated with indirect speech or not, then quantitative methods can show whether there is such a correlation. If a study on middle voice in the Athabaskan language Dena'ina tries to identify how syntax and semantics are related to middle voice marking, quantitative methods can reveal interesting things (cf. Berez and Gries 2010).

The last two points lead up to a more general argument already alluded to above: often only quantitative methods can separate the wheat from the

chaff. Let's assume a linguist wanted to test the so-called aspect hypothesis according to which imperfective and perfective aspect are preferred in present and past tense respectively (cf. Shirai and Andersen 1995). Strictly speaking, the linguist would have to test all verbs in all languages, the so-called *population*. This is of course not possible so the linguist studies a *sample* of sentences to investigate their verbal morphology. Let's further assume the linguist took and investigated a small sample of 38 sentences in one language and got the results in Table 1.

Table 1. A fictitious distribution of tenses and aspects in a small corpus

	Imperfective	Perfective	Totals
Present tense	12	6	18
Past tense	7	13	20
Totals	19	19	38

These data look like a very obvious confirmation of the aspect hypothesis: there are more present tenses with imperfectives and more past tenses with perfectives. However, the so-called chi-squared test, which could perhaps be used for these data, shows that this tense-aspect distribution can arise by chance with a probability p that exceeds the usual threshold of 5% adopted in quantitative studies. Thus, the linguist would not be allowed to accept the aspect hypothesis for the population on the basis of this sample. The point is that an intuitive eye-balling of this table is insufficient – a statistical test is needed to protect the linguist against invalid generalizations.

A more eye-opening example is discussed by Crawley (2007: 314f.). Let's assume a study showed that two variables x and y are correlated such that the larger the value of x, the larger the value of y; cf. Figure 1.

Note, however, that the data actually also contain information about a third variable (with seven levels a to g) on which x and y depend. Interestingly, if you now inspect what the relation between x and y looks like for each of the seven levels of the third variable separately, you see that the relation suddenly becomes "the larger x, the *smaller* y"; cf. Figure 2, where the seven levels are indicated with letters. Such patterns in data are easy to overlook – they can only be identified through a careful quantitative study, which is why knowledge of statistical methods is indispensible.

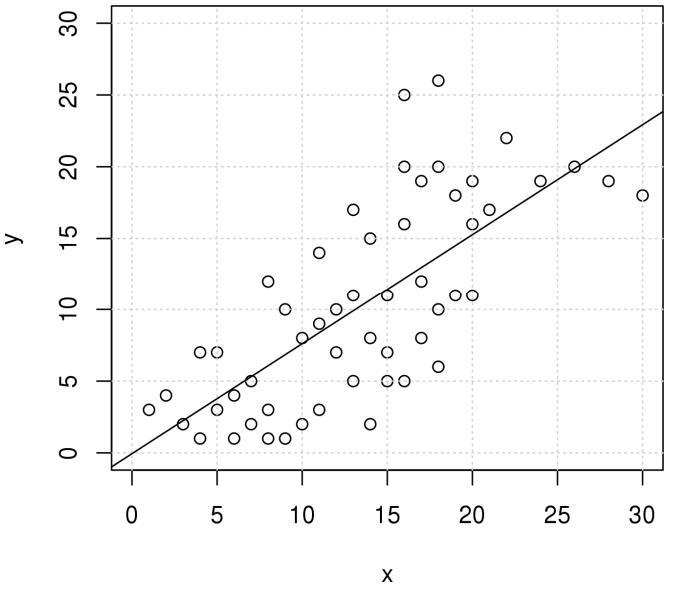

Figure 1. A correlation between two fictitious variables *x* and *y*

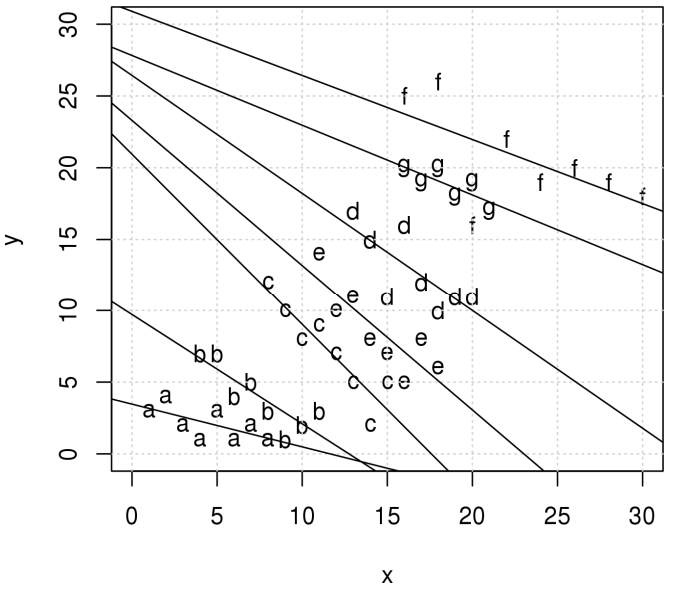

Figure 2. A correlation between two fictitious variables *x* and *y*, controlled for a fictitious third variable

For students of linguistics – as opposed to experienced practitioners – there is also a very practical issue to consider. Sometime soon you will want to write a thesis or dissertation. Quantitative methods can be extremely useful and powerful if only to help you avoid the pitfalls posed by the data in Table 1 and Figure 1 or data from published studies I regularly discuss in my classes and workshops. It is therefore hopefully obvious now that quantitative methods have a lot to offer, and I hope this book will provide you with some good and practical background knowledge.

This argument has an additional aspect to it. Contrary to, say, literary criticism, linguistics is an empirical science. Thus, it is necessary – in particular for students – to know about basic methods and assumptions of empirical research and statistics to be able to understand both scientific argumentation in general and linguistic argumentation in particular. This is especially relevant in the domains of, for example, contemporary quantitative corpus linguistics or psycholinguistics, where data are often evaluated with such a high degree of sophistication that a basic knowledge of the relevant terminology is required. Without training, what do you make of statements such as "The interaction between the size of the object and the size of the reference point does not reach standard levels of significance: $F_{1, 12} = 2.18$; $p = 0.166$; *partial eta*$^2 = 0.154$."? Who knows off the top of their head whether the fact that the average sentence length of ten female second language learners in an experiment was about two words larger than the average sentence length of ten male second language learners is more likely to mean something, or whether this is more likely a product of chance? Again, such data need serious statistical analysis.

3. The design and the logic of quantitative studies

In this section, we will have a very detailed look at the design of, and the logic underlying, quantitative studies. I will distinguish several phases of quantitative studies and consider their structure and discuss the reasoning employed in them. The piece of writing in which you then describe your quantitative research will often have four parts: *introduction*, *methods*, *results*, and *discussion*. If you discuss more than one case study in your writing, then typically each case study gets its own methods, results, and discussion sections, followed by a general discussion.

With few exceptions, the discussion in this section will be based on a linguistic example, particle placement in English, i.e. the constituent order alternation of transitive phrasal verbs exemplified in (1).

(1) a. He picked up [$_{NP}$ the book].
CONSTRUCTION: *VPO* (verb - particle - object)
b. He picked [$_{NP}$ the book] up.
CONSTRUCTION: *VOP* (verb - object - particle)

An interesting aspect of this alternation is that, most of the time, both constructions appear to be quite synonymous and native speakers of English usually cannot explain why they produce (1a) on one occasion and (1b) on some other occasion. In the past few decades, linguists have tried to describe, explain, and predict the alternation (cf. Gries 2003a for a recent overview), and in this section, we will use it to illustrate the structure of a quantitative study.

3.1. Scouting

At the beginning of your study, you want to get an overview of previous work on the phenomenon you are interested in, which also gives you a sense of what still can or needs to be done. In this phase, you try to learn of existing theories that can be empirically tested or, much more infrequently, you enter uncharted territory in which you are the first to develop a new theory. This is a list of the activities that is typically performed in this scouting phase:

- a first (maybe informal) characterization of the phenomenon;
- studying the relevant literature;
- observations of the phenomenon in natural settings to aid first inductive generalizations;
- collecting additional information (e.g., from colleagues, students, etc.);
- deductive reasoning on your part.

If you take just a cursory look at particle placement, you will quickly notice that there is a large number of variables that influence the constructional choice. A *variable* is defined as a symbol for a set of states, i.e., a characteristic that – contrary to a constant – can exhibit at least two different states or levels (cf. Bortz and Döring 1995: 6 or Bortz 2005: 6) or, more intuitively, as "descriptive properties" (Johnson 2008: 4) or as measurements of an item that can be either numeric or categorical (Evert, p.c.).

Variables that might influence particle placement include the following:[1]

- COMPLEXITY: is the direct object a *SIMPLE DIRECT OBJECT* (e.g., *the book*), a *PHRASALLY-MODIFIED DIRECT OBJECT* (e.g., *the brown book* or *the book on the table*) or a *CLAUSALLY-MODIFIED DIRECT OBJECT* (e.g., *the book I had bought in Europe*) (cf., e.g., Fraser 1966);
- LENGTH: the length of the direct object (cf., e.g., Chen 1986, Hawkins 1994), which could be measured in syllables, words, …;
- DIRECTIONAL OBJECT: the *PRESENCE* of a directional prepositional phrase (PP) after the transitive phrasal verb (e.g. in *He picked the book up from the table*) or its *ABSENCE* (cf. Chen 1986);
- ANIMACY: whether the referent of the direct object is *INANIMATE* as in *He picked up the book*, or *ANIMATE* as in *He picked his dad up* (cf. Gries 2003a: Ch. 2);
- CONCRETENESS: whether the referent of the direct object is *ABSTRACT* as in *He brought back peace to the region*, or *CONCRETE* as in *He brought his dad back to the station* (cf. Gries 2003a: Ch. 2);
- TYPE: is the part of speech of the head of the direct object a *PRONOUN* (e.g., *He picked him up this morning*), a *SEMIPRONOUN* (e.g., *He picked something up from the floor*), a *LEXICAL NOUN* (e.g., *He picked people up this morning*) or a *PROPER NAME* (e.g., *He picked Peter up this morning*) (cf. Van Dongen 1919).

During this early phase, it is often useful to summarize your findings in tabular format. One possible table summarizes which studies (in the columns) discussed which variable (in the rows). On the basis of the above list, this table could look like Table 2 and allows you to immediately recognize (i) which variables many studies have already looked at and (ii) the studies that looked at most variables. Another table summarizes the variable levels and their preferences for one of the two constructions. Again, on the basis of the above list, this table would look like Table 3, and you can immediately see that, for some variables, only one level has been associated with a particular constructional preference.

Table 3 already suggests that CONSTRUCTION: *VPO* is used with cognitively more complex direct objects: long complex NPs with lexical nouns referring to abstract things. CONSTRUCTION: *VOP* on the other hand is used with the opposite preferences. For an actual study, this first impression would of course have to be phrased more precisely. In addition, you should

1. I print variables in small caps and their levels in italicized small caps.

also compile a list of other factors that might either influence particle placement directly or that might influence your sampling of sentences or experimental subjects or … Much of this information would be explained and discussed in the first section of the empirical study, the introduction.

Table 2. Summary of the literature on particle placement I

	Fraser (1966)	Chen (1986)	Hawkins (1994)	Gries (2003a)	Van Dongen (1919)
COMPLEXITY	×				
LENGTH		×	×		
DIRECTIONALPP		×			
ANIMACY				×	
CONCRETENESS				×	
TYPE					×

Table 3. Summary of the literature on particle placement II

	Variable level for CONSTRUCTION: *VPO*	Variable level for CONSTRUCTION: *VOP*
COMPLEXITY	*PHRASALLY-MODIFIED* *CLAUSALLY MODIFIED*	
LENGTH	*LONG*	
DIRECTIONALPP	*ABSENCE*	*PRESENCE*
ANIMACY	*INANIMATE*	*ANIMATE*
CONCRETENESS	*ABSTRACT*	*CONCRETE*
TYPE		*PRONOMINAL*

3.2. Hypotheses and operationalization

Once you have an overview of the phenomenon you are interested in and have decided to pursue an empirical study, you usually formulate hypotheses. What does that mean and how do you proceed? To approach this issue, let us see what hypotheses are and what kinds of hypotheses there are.

3.2.1. Scientific hypotheses in text form

Following Bortz and Döring (1995: 7), I will consider a hypothesis to be a statement that meets the following three criteria:

- it is a general statement that is concerned with more than just a singular event;
- it is a statement that at least implicitly has the structure of a conditional sentence (*if …, then …* or *the …, the …*) or can be paraphrased as one;
- it is potentially falsifiable, which means it must be possible to think of events or situations that contradict the statement. Most of the time, this implies that the scenario described in the conditional sentence must also be testable. However, these two characteristics are not identical. There are statements that are falsifiable but not testable such as "If children grow up without any linguistic input, then they will grow up to speak Latin." This statement is falsifiable, but for obvious ethical reasons not testable (anymore; cf. Steinberg 1993: Section 3.1).

The following statement is a scientific hypothesis according to the above criteria: "Reducing the minimum age to obtain a driver's license from 18 years to 17 years in European countries will double the number of traffic accidents in these countries within two years." This statement is a general statement that is not restricted to just one event, just one country, etc. Also, this statement can be paraphrased as a conditional sentence: "If one reduces the minimum age …, then the number of traffic accidents will double …" Lastly, this statement is falsifiable because it is conceivable – actually, very likely – that if one reduced the minimum age, that the number of traffic accidents would not double. Accordingly, the following statement is not a scientific hypothesis: "Reducing the minimum age to obtain a driver's license from 18 years to 17 years in European countries may double the number of traffic accidents in these countries within two years." This statement is a general statement, it can be paraphrased into a conditional sentence, it is testable because the minimum age could be reduced, but it is not a hypothesis according to the above definition because the word *may* basically means 'may or may not': the statement is true if the number of traffic accidents doubles, but also if it does not. Put differently, whatever one observed after the reduction of the minimum age, it would be compatible with the statement.

With regard to particle placement, the following statements are examples of scientific hypotheses:

- if the direct object of a transitive phrasal verb is syntactically complex, then native speakers will produce the constituent order *VPO* more often than when the direct object is syntactically simple;
- if the direct object of a transitive phrasal verb is long, then native speak-

ers will produce the constituent order *VPO* more often than when the direct object is short;
- if a verb-particle construction is followed by a directional PP, then native speakers will produce the constituent order *VOP* more often than when no such directional PP follows (and analogously for all other variables mentioned in Table 3).

When you formulate a hypothesis, it is also important that the notions that you use in the hypothesis are formulated precisely. For example, if a linguistic theory uses notions such as *cognitive complexity* or *availability in discourse* or even something as seemingly straightforward as *constituent length*, then it will be necessary that the theory can define what exactly is meant by this; in Section 1.3.2.2 we will deal with this in much more detail.

We can distinguish two types of hypotheses. The first, the one we have been talking about so far, consists of two parts, an *if* part (*IV*) and a *then* part (*DV*). The *IV* stands for *independent variable*, the variable in the *if* part of the hypothesis that is often, but not necessarily, the cause of the changes/effects in the *then* part of the hypothesis. The *DV* on the other hand stands for *dependent variable*, the variable in the *then* part of the hypothesis and whose values, variation, or distribution is to be explained. In addition, it is useful for later to also mention *confounding variables* and *moderator variables*. The former can be defined as variables that are correlated with independent dependent variables; the latter can be defined as variables (often extraneous to the initial design of a study) that influence/moderate the relationship between the independent and the dependent variable(s).

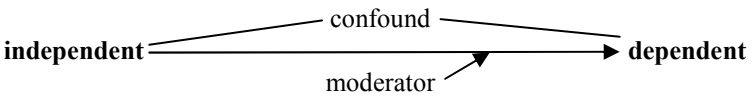

Figure 3. Different types of variables

With this terminology, we can now paraphrase the above hypotheses. In the first, *IV* is the syntactic complexity of the direct object (COMPLEXITY with the three levels *SIMPLE*, *PHRASALLY-MODIFIED*, and *CLAUSALLY-MODIFIED*), and *DV* is the choice of construction (CONSTRUCTION with the two levels *VPO* and *VOP*). In the second hypothesis, *IV* is the length of the direct object (LENGTH with values from 1 to *x*), and *DV* is again the choice of construction (CONSTRUCTION with the two levels *VPO* and *VOP*), etc.

The second type of hypothesis only contains one dependent variable, but no independent variable with which the dependent variable's behavior

is explained. In such cases, the hypothesis is 'only' a statement about what the values, variation, or distribution of the dependent variable looks like. Frequent examples postulate equal distributions (e.g., frequencies) or particular shapes of distributions (e.g., bell-shaped normal curves):

– The two constructions or, more technically, the two levels of CONSTRUCTION (*VPO* and *VOP*) are not equally frequent; note again how this does not mention an independent variable.
– The lengths of direct objects are not normally distributed.

In what follows, we will deal with both kinds of hypotheses (with a bias toward the former).

Thus, we can also define a scientific hypothesis as a statement about either the relation(s) between two or more variables or, for the second kind, as a statement about one variable in some sampling context, which is expected to also hold in similar contexts and/or for similar objects in the population. Thus, once potentially relevant variables to be investigated have been identified, you formulate a hypothesis by relating the relevant variables in the appropriate conditional sentence or some paraphrase thereof.

After your hypothesis has been formulated in the above text form, you also have to define – before you collect data! – which situations or states of affairs would falsify your hypothesis. Thus, in addition to your own hypothesis – the so-called *alternative hypothesis* H_1 – you now also formulate another hypothesis – the so-called *null hypothesis* H_0 – which is the logical opposite to your H_1. Often, that means that you get the H_0 by inserting the word *not* into the H_1. For the first of the above three hypotheses involving both a dependent and and independent variable, this is what the text version of H_0 would look like:

$H_{0\ type\ 1}$: If the direct object of a transitive phrasal verb is syntactically complex, then native speakers will *not* produce the constituent order *VPO* more often than when the direct object is syntactically simple.

For the first of the above two hypotheses involving only a dependent variable, H_0 would be this:

$H_{0\ type\ 2}$: The two constructions or, more technically, the two levels of CONSTRUCTION (*VPO* and *VOP*) are *not* not equally frequent, i.e. are equally frequent.

It is crucial to formulate H_0 as mentioned above, essentially by inserting *not*. The idea is that both hypotheses – H_1 and H_0 – cover the whole result space, i.e. every result theoretically possible. Thus, if your H_1 was "Complex objects lead to more CONSTRUCTION: *VPO* than CONSTRUCTION: *VOP*," then your H_0 should *not* be "Complex objects lead to *fewer* CONSTRUCTION: *VPO* than CONSTRUCTION: *VOP*" because these two hypotheses do not cover all results possible – they do not cover the case where the two constructions are equally frequent.

In the vast majority of cases, the first type of H_0 states that there is no difference between (two or more) groups or no relation between the independent variable(s) and the dependent variable(s) and that whatever difference or effect you get is only due to chance or random variation. The second type of H_0 typically states that the dependent variable is distributed randomly or in accordance with some well-known mathematically definable distribution such as the normal distribution. However, an additional complication is that you must distinguish two kinds of H_1s: *directional H_1s* not only predict that there is some kind of effect or difference or relation but also the direction of the effect – note the expression "more often" in the above type 1 H_1 relating CONSTRUCTION and COMPLEXITY. On the other hand, *non-directional H_1s* only predict that there is some kind of effect or difference or relation without specifying the direction of the effect. A non-directional H_1 for the above type 1 example would therefore be this:

$H_{1\ \text{type 1 non-dir.}}$: If the direct object of a transitive phrasal verb is syntactically complex, then native speakers will produce the constituent order *VPO differently often* than when the direct object is syntactically simple.

Thus, H_0 states that there is no correlation between the syntactic complexity of a direct object and the constructional choice in the population, and that if you nevertheless find one in the sample, then this is only a chance effect. Both H_1s state that there is a correlation – thus, you should also find one in your sample. Both of these hypotheses must be formulated *before* the data collection so that one cannot present whatever result one gets as the 'predicted' one. Of course, all of this has to be discussed in the introduction of the written version of your paper or, maybe, at the beginning of the methods section.

3.2.2. Operationalizing your variables

Formulating your hypotheses in the above text form is not the last step in this part of the study, because it is as yet unclear how the variables invoked in your hypotheses will be investigated. For example and as mentioned above, a notion such as cognitive complexity can be defined in many different and differently useful ways, and even something as straightforward as constituent length is not always as obvious as it may seem: do we mean the length of, say, a direct object in letters, phonemes, syllables, morphemes, words, syntactic nodes, etc.? Therefore, you must find a way to *operationalize* the variables in your hypothesis. This means that you decide what will be observed, counted, measured etc. when you investigate your variables.

For example, if you wanted to operationalize a person's KNOWLEDGE OF A FOREIGN LANGUAGE, you could do this as follows:

- COMPLEXITY OF THE SENTENCES that a person can form in the language in a test (only main clauses? also compound sentences? also complex sentences? how many of each?);
- AMOUNT OF TIME in seconds between two errors in conversation;
- NUMBER OF ERRORS PER 100 WORDS in a text that the person writes in 90 minutes.

What is wrong with the following two proposals for operationalization?

- AMOUNT OF ACTIVE VOCABULARY;
- AMOUNT OF PASSIVE VOCABULARY.

THINK
BREAK

These proposals are not particularly useful because, while knowing these amounts would certainly be very useful to assess somebody's knowledge of a foreign language, they are not directly observable: it is not clear what you would count or measure since it is not exactly practical to tell a learner to write down all the words he knows … If you in turn operationalize the amount of passive vocabulary on the basis of the number of words a person knows in a vocabulary test (involving, say, words from

different frequency bands) or in a synonym finding test, then you know what to count – but the above is too vague.

From the above it follows that operationalizing involves using levels of numbers to represent states of variables. A number may be a measurement (402 ms reaction time, 12 words in a synonym finding test, the direct object is four syllables long), but levels, i.e. discrete non-numerical states, can theoretically also be coded using numbers. Thus, variables are not only distinguished according to their role in the hypotheses – independent vs. dependent – but also according to their level of measurement:

- nominal or categorical variables are variables with the lowest information value. Different values of these variables only reveal that the objects with these different values exhibit different characteristics. Such variables are called *nominal variables* (or *binary variables*) when they can take on only two different levels; such variables are called *categorical variables* when they can take on three or more different levels. In our example of particle placement, the variable DIRECTIONALPP could be coded with 1 for the *ABSENCE* and 2 for *PRESENCE*, but note that the fact that the value for *PRESENCE* is twice as large as that for *ABSENCE* does not mean anything (other than that the values are different) – theoretically, you could code *ABSENCE* with 34.2 and *PRESENCE* with 7.[2] Other typical examples of nominal or categorical variables are ANIMACY (*ANIMATE* vs. *INANIMATE*), CONCRETENESS (*CONCRETE* vs. *ABSTRACT*), STRESS (*STRESSED* vs. *UNSTRESSED*), AKTIONSART (*ACTIVITY* vs. *ACCOMPLISHMENT* vs. *ACHIEVEMENT* vs. *STATE*) etc.
- *ordinal variables* not only distinguish objects as members of different categories the way that nominal/categorical variables do – they also allow to rank-order the objects in a meaningful way. However, differences between ranks cannot be meaningfully compared. Grades are a typical example: a student with an A (4 grade points) scored a better result than a student with a C (2 grade points), but just because 4 is two times 2, that does not necessarily mean that the A-student did exactly twice as well as the C-student – depending on the grading system, the

2. Often, nominal variables are coded using 0 and 1. There are two reasons for that: (i) a conceptual reason: often, such nominal variables can be understood as the presence (=1) or the absence (=0) of something or even as a ratio variable (cf. below); i.e., in the example of particle placement, the nominal variable CONCRETENESS could be understood as a ratio variable NUMBER OF CONCRETE REFERENTS; (ii) for reasons I will not discuss here, it is computationally useful to use 0 and 1 and, somewhat counterintuitively, some statistical software other than R even requires that kind of coding.

A-student may have given three times as many correct answers as the C-student. In the particle placement example, the variable COMPLEXITY is an ordinal variable if you operationalize it as above: *SIMPLE NP* (1) vs. *PHRASALLY-MODIFIED* (2) vs. *CLAUSALLY-MODIFIED* (3). It is useful to make the ranks compatible with the variable: if the variable is called SYNTACTIC COMPLEXITY, then large rank numbers should represent large degrees of complexity, i.e., complex direct objects. If, on the other hand, the variable is called SYNTACTIC SIMPLICITY, then large rank numbers should represent large degrees of simplicity, i.e. simple direct objects. Other typical examples are SOCIO-ECONOMIC STATUS or DEGREE OF IDIOMATICITY or PERCEIVED VOCABULARY DIFFICULTY (e.g., *LOW*/1 vs. *INTERMEDIATE*/2 vs. *HIGH*/3).

– *ratio variables* not only distinguish objects as members of different categories and with regard to some rank ordering – they also allow to meaningfully compare the differences and ratios between values. For example, LENGTH IN SYLLABLES is such a ratio variable: when one object is six syllables long and another is three syllables long, then the first is of a different length than the second (the categorical information), the first is longer than the second (the ordinal information), and it is exactly twice as long as the second. Other typical examples are annual salaries, or reaction times in milliseconds.[3]

These differences can be clearly illustrated in a table of a fictitious data set on lengths and degrees of complexity of subjects and objects – which column contains which kind of variable?

Table 4. A fictitious data set of subjects and objects

DATA POINT	COMPLEXITY	DATA SOURCE	SYLLLENGTH	GRMRELATION
1	*HIGH*	*D8Y*	6	*OBJECT*
2	*HIGH*	*HHV*	8	*SUBJECT*
3	*LOW*	*KB0*	3	*SUBJECT*
4	*INTERMEDIATE*	*KB2*	4	*OBJECT*

THINK
BREAK

3. Strictly speaking, there is also a class of so-called *interval variables*, which I am not going to discuss here separately from ratio variables.

DATA POINT is essentially a categorical variable: every data point gets its own number so that you can uniquely identify it, but the number as such may represent little more than the order in which the data points were entered. COMPLEXITY is an ordinal variable with three levels. DATA SOURCE is another categorical variable: the levels of this variable are file names from the British National Corpus. SYLLLENGTH is a ratio variable since the third object can correctly be described as half as long as the first. GRMRELATION is a nominal/categorical variable. These distinctions are very important since these levels of measurement determine which statistical tests can and cannot be applied to a particular question and data set, as we will see below. As a rule of thumb already, it is usually best to work with the highest level of measurement; I will come back to this shortly.

The issue of operationalization is one of the most important of all. If you do not operationalize your variables properly, then the whole study might be useless since you may actually end up not measuring what you want to measure. Without an appropriate operationalization, the *validity* of your study is at risk. If we investigated the question of whether subjects in English are longer than direct objects and looked through sentences in a corpus, we might come across the sentence in (2):

(2) [$_{\text{SUBJECT}}$ The younger bachelors] ate [$_{\text{OBJECT}}$ the nice little parrot].

The result for this sentence depends on how LENGTH is operationalized. If LENGTH is operationalized as *number of morphemes*, then the subject is longer than the direct object: 5 (*The*, *young*, comparative *-er*, *bachelor*, plural *s*) vs. 4 (*the*, *nice*, *little*, *parrot*). However, if LENGTH is operationalized as *number of words*, the subject (3 words) is shorter than the direct object (4 words). And, if LENGTH is operationalized as *number of characters without spaces*, the subject and the direct object are equally long (19 characters). In this contrived case, thus, the operationalization alone determines the result.

3.2.3. Scientific hypotheses in statistical/mathematical form

Once you have formulated both your own H_1 and the logically complementary H_0 in text form and have defined how the variables will be operationalized, you also formulate two statistical versions of these hypotheses. That is, you first formulate the two text hypotheses, and in the statistical hypotheses you then express the numerical results you expect on the basis of the

text hypotheses. Such numerical results usually involve one of five different mathematical forms:

- frequencies;
- means;
- dispersions;
- correlations;
- distributions.

We begin by looking at a simple example of an H_1 regarding particle placement: if a verb-particle construction is followed by a directional PP, then native speakers will produce the constituent order *VOP* more often than when no such directional PP follows. To formulate the statistical hypothesis counterpart to this text form, you have to answer the question, if I investigated, say, 200 sentences with verb-particle constructions in them, how would I know whether H_1 is (more likely) correct or not? (As a matter of fact, you actually have to proceed a little differently, but we will get to that later.) One possibility of course is to count how often CONSTRUCTION: *VPO* and CONSTRUCTION: *VOP* are followed by a directional PP, and if there are more directional PPs after CONSTRUCTION: *VOP* than after CONSTRUCTION: *VPO*, then this provides support for H_1. Thus, this possibility involves frequencies and the statistical hypotheses are:

$H_{1\ \text{directional}}$: $n_{\text{dir. PPs after CONSTRUCTION: } VPO} < n_{\text{dir. PPs after CONSTRUCTION: } VOP}$
$H_{1\ \text{non-directional}}$: $n_{\text{dir. PPs after CONSTRUCTION: } VPO} \neq n_{\text{dir. PPs after CONSTRUCTION: } VOP}$
H_0: $n_{\text{dir. PPs after CONSTRUCTION: } VPO} = n_{\text{dir. PPs after CONSTRUCTION: } VOP}$[4]

Just in passing: what do these statistical hypotheses presuppose?

THINK
BREAK

4. Note: I said above that you often obtain H_0 by inserting *not* into H_1. Thus, when the statistical version of H_1 involves a "<", then you might expect the statistical version of H_0 to contain a "≥". However, we will follow the usual convention also mentioned above that H_0 states the absence of a difference/effect/correlation etc., which is why we write "=". You will see below that the cases covered by "≥" will still be invoked in the computations that are based on these statistical hypotheses.

They presuppose that you investigate equally many instances of both constructions because otherwise a small observed frequency of directional PPs after CONSTRUCTION: *VOP* – the frequency we expect to be large – could simply be due to a small overall frequency of CONSTRUCTION: *VOP*. For the variable COMPLEXITY, you could formulate similar hypotheses based on frequencies, if COMPLEXITY is operationalized on the basis of, for example, the three levels mentioned above.

Let us now turn to an example involving statistical hypotheses based on means: if the direct object of a transitive phrasal verb is long, then native speakers will produce the constituent order *VPO* more often than when it is not. One way to proceed is to measure the average lengths of direct objects in CONSTRUCTION: *VPO* and CONSTRUCTION: *VOP* and then compare these average lengths to each other. You could therefore write:

$H_{1\ directional}$: *mean* Length of the direct object in CONSTRUCTION: *VPO* > *mean* Length of the direct object in CONSTRUCTION: *VOP*

$H_{1\ non\text{-}directional}$: *mean* Length of the direct object in CONSTRUCTION: *VPO* ≠ *mean* Length of the direct object in CONSTRUCTION: *VOP*

H_0: *mean* Length of the direct object in CONSTRUCTION: *VPO* = *mean* Length of the direct object in CONSTRUCTION: *VOP*

With similarly obvious operationalizations, the other text hypotheses from above can be transformed into analogous statistical hypotheses. Now, and only now, we finally know what needs to be observed in order for us to reject H_0. (We will look at hypotheses involving correlations, dispersion, and distributions later.)

All hypotheses discussed so far were concerned with the simple case where a sample of verb-particle constructions was investigated regarding whether the two constructions differ with regard to one independent variable (e.g., DIRECTIONALPP). The statistical methods to handle such cases are the subject of Chapter 4. However, things are often not that simple: most phenomena are multifactorial in nature, which means dependent variables are usually influenced by, or at least related to, more than one independent variable. While the overall logic is the same as above, some complications arise and we will postpone their discussion until Chapter 5.

3.3. Data collection and storage

Only after all variables have been operationalized and all hypotheses have

been formulated do you actually collect your data. For example, you run an experiment or do a corpus study or … However, you will hardly ever study the whole population of events but a sample so it is important that you choose your sample such that it is representative and balanced with respect to the population to which you wish to generalize. Here, I call a sample *representative* when the different parts of the population are reflected in the sample, and I call a sample *balanced* when the sizes of the parts in the population are reflected in the sample. Imagine, for example, you want to study the frequencies and the uses of the discourse marker *like* in the speech of Californian adolescents. To that end, you want to compile a corpus of Californian adolescents' speech by asking some Californian adolescents to record their conversations. In order to obtain a sample that is representative and balanced for the population of all the conversations of Californian adolescents, the proportions of the different kinds of conversations in which the subjects engage would ideally be approximately reflected in the sample. For example, a good sample would not just include the conversations of the subjects with members of their peer group(s), but also conversations with their parents, teachers, etc., and if possible, the proportions that all these different kinds of conversations make up in the sample would correspond to their proportions in real life, i.e. the population.

While it is important you try to stick to these rules as much as possible, why are they often more of a theoretical ideal?

THINK BREAK

This is often just a theoretical ideal because we don't know all parts and their proportions in the population. Who would dare say how much of an average Californian adolescent's discourse – and what is an average Californian adolescent anyway? – takes place within his peer group, with his parents, with his teachers etc.? And how would we measure the proportion – in words? sentences? minutes? Still, even though these considerations will often only result in estimates, you must think about the composition of your sample(s) just as much as you think about the exact operationalization of your variables. If you do not do that, then the whole study may well fail because you may be unable to generalize from whatever you find in your sample to the population. One important rule in this connection is to choose the elements that enter into your sample randomly, to randomize. For ex-

ample, if the adolescents who participate in your study receive a small recording device with a lamp and are instructed to always record their conversations when the lamp lights up, then you could perhaps send a signal to the device at random time intervals (as determined by a computer). This would make it more likely that you get a less biased sample of many different kinds of conversational interaction, which would then reflect the population better.

Let us briefly look at a similar example from the domain of first language acquisition. It was found that the number of questions in recordings of caretaker-child interactions was surprisingly high. Some researchers suspected that the reason for that was parents' (conscious or unconscious) desire to present their child as very intelligent so that they asked the child "And what is that?" questions all the time so that the child could show how many different words he knew. Some researchers then changed their sampling method such that the recording device was always in the room, but the parents did not know exactly when it would record caretaker-child interaction. The results showed that the proportion of questions decreased considerably …

In corpus-based studies, you will often find a different kind of randomization. For example, you will find that a researcher first retrieved all instances of the word he is interested in and then sorted all instances according to random numbers. When the researcher then investigates the first 20% of the list, he has a random sample. However you do it, randomization is one of the most important principles of data collection.

Once you have collected your data, you have to store them in a format that makes them easy to annotate, manipulate, and evaluate. I often see people – students as well as seasoned researchers – print out long lists of data points, which are then annotated by hand, or people annotate concordance lines from a corpus in a text processing software. This may seem reasonable for small data sets, but it doesn't work or is extremely inconvenient for larger ones, and the generally better way of handling the data is in a spreadsheet software (e.g., LibreOffice Calc) or a database, or in R. However, there is a set of ground rules that defines the desired so-called *case-by-variable format* and needs to be borne in mind.

i. the first row contains the names of all variables;
ii. each of the other rows represents one and only one data point, where I am using *data point* to refer to a single observation of the dependent variable;
iii. the first column just numbers all *n* cases from 1 to *n* so that every row

can be uniquely identified and so that you can always restore one particular ordering (e.g., the original one);

iv. each of the remaining columns represents one and only one variable or feature with respect to which every data point gets annotated. In a spreadsheet for a corpus study, for example, one additional column may contain the name of the corpus file in which the word in question is found; another column may provide the line of the file in which the word was found. In a spreadsheet for an experimental study, one column should contain some unique identifier of each subject; other columns may contain the age of the subject, the sex of the subject, the exact stimulus or some index representing the stimulus the subject was presented with, the order index of a stimulus presented to a subject (so that you can test whether a subject's performance changes systematically in the course of the experiment), …;

v. missing data are entered as NA and not just with empty cells (which also means no other variable level should be abbreviated as NA) in order to preserve the formal integrity of the data set (i.e., have all rows and columns contain the same number of elements) and to be able to do follow-up studies on the missing data to see whether, for example, there is a pattern in the missing data points which needs to be accounted for.

Some additional very helpful suggestions especially for working with R are to have the column names in the first row be in all caps, to never code the levels of categorical levels as numbers but as words/character strings in small letters, and to not use 'weird' characters such as spaces, periods, commas, tabs, #, single/double quotes or others in variable names or levels.

To make sure these points are perfectly clear, let us look at two examples. Let's assume for your study of particle placement you had looked at a few sentences and counted the number of syllables of the direct objects. First, a question: in this design, what is the dependent variable and what is the independent variable?

THINK
BREAK

The independent variable is the ratio variable LENGTH (in syllables), which can take on all sorts of positive integer values. The dependent variable is the nominal variable CONSTRUCTION, which can be either *VPO* or

VOP. When all hypotheses were formulated and, subsequently, data were collected and coded, then I sometimes see a format such as the one represented in Table 5.

Table 5. A not-so-good table 1

	LENGTH: *2*	LENGTH: *3*	LENGTH: *5*	LENGTH: *6*
CONSTRUCTION: *VPO*	||	||	|||	||
CONSTRUCTION: *VOP*	||||	|||	||	|

As a second example, let's look at the hypothesis that subjects and direct objects are differently long (in words). Again the question: what is the dependent variable and what is the independent variable?

THINK
BREAK

The independent variable is the nominal variable RELATION, which can be *SUBJECT* or *OBJECT*. The dependent variable is LENGTH, which can take on positive integer values. If you formulated all four hypotheses (H_1: text and statistical form; H_0: text and statistical form) and then looked at the small corpus in (3), then your spreadsheet should *not* look like Table 6.

(3) a. The younger bachelors ate the nice little cat.
b. He was locking the door.
c. The quick brown fox hit the lazy dog.

Table 6. A not-so-good table 2

SENTENCE	SUBJ	ONJ
The younger bachelors ate the nice little cat.	3	4
He was locking the door.	1	2
The quick brown fox hit the lazy dog.	4	3

Both Table 5 and Table 6 violate all of the above rules. In Table 6, for example, every row represents two data points, not just one, namely one data point representing some subject's length and one representing the length of the object from the same sentence. Also, not every variable is

represented by one and only column – rather, Table 6 has two columns with data points, each of which represents one level of an independent variable, not one variable. Before you read on, how would you have to reorganize Table 6 to make it compatible with the above rules?

THINK
BREAK

Table 7 is a much better way to store the data: every data point has its own row and is characterized according to the two variables in their respective columns. An even more comprehensive version may now even include one column containing just the subjects and objects so that particular cases can be found more easily. In the first row of such a column, you would find *The younger bachelor*, in the second row of the same column, you would find *the nice little cat* etc. The same logic applies to the improved version of Table 5, which should look like Table 8.

Table 7. A much better coding of the data in Table 6

CASE	SENT#	SENTENCE	RELATION	LENGTH
1	1	The younger bachelors ate the nice little cat.	subj	3
2	1	The younger bachelors ate the nice little cat.	obj	4
3	2	He was locking the door.	subj	1
4	2	He was locking the door.	obj	2
5	3	The quick brown fox hit the lazy dog.	subj	4
6	3	The quick brown fox hit the lazy dog.	obj	3

With very few exceptions, this is the format in which you should always save your data.[5] Ideally, you enter the data in this format into a spreadsheet software and save the data (i) in the native file format of that application (to preserve colors and other formattings you may have added) and (ii) into a tab-delimited text file, which is easier to import into R.

5. There are some more complex statistical techniques which can require different formats, but in the vast majority of cases, the standard format discussed above (also sometimes called *long format*) is the one that you will need and that will allow you to easily switch to another format.

Table 8. A much better coding of the data in Table 5

CASE	CONSTRUCTION	LENGTH
1	vpo	2
2	vpo	2
3	vop	2
4	vop	2
5	vop	2
6	vop	2
7	vpo	3
8	vpo	3
9	vop	3
10	vop	3
11	vop	3
…	…	…

All these steps having to do with the data collection must be described in the methods part of your written version: what is the population to which you wanted to generalize, how did you draw your (ideally) representative and balanced sample, which variables did you collect data for, etc.

3.4. The decision

When the data have been stored in a format that corresponds to that of Table 7/Table 8, you can finally do what you wanted to do all along: evaluate the data with some statistical test. (For now I will not address how you decide which statistical test to choose but I will return to this topic at the beginning of Chapter 4.) As a result of that evaluation you will obtain frequencies, means, dispersions, correlation coefficients, or distributions. However, one central aspect of this evaluation is that you actually do not simply try to show that your H_1 is correct – contrary to what you might expect you try to show that the statistical version of H_0 is wrong, and since H_0 is the logical counterpart to H_1, this supports your H_1. The obvious question now is, why this 'detour'? The answer to this question can be approached again with reference to the example of subjects and objects: let's assume you formulated these hypotheses:

H_1: The subjects and direct objects in transitive clauses are differently long.
H_0: The subjects and direct objects in transitive clauses are not differently long.

Now consider the following two questions:

- how many subjects and direct objects do you maximally have to study to show that the above H_1 is correct?
- how many subjects and direct objects do you minimally have to study to show that the above H_0 is incorrect?

THINK
BREAK

You probably figured out quickly that the answer to the first question is "infinitely many." Strictly speaking, you can only be sure that H_1 is correct if you have studied all subjects and direct objects and found not a single counterexample. The answer to the second question is "one each" because if the first subject is longer or shorter than the first object, we know that, strictly speaking, H_0 is not correct. However, especially in the humanities and social sciences you do not usually reject a hypothesis on the basis of just one counterexample. Rather, you use the following four-step procedure, which is sometimes referred to as the Null Hypothesis Significance Testing (NHST) paradigm:

i. you define a so-called *significance level* p_{critical}, which is usually set to 0.05 (i.e., 5%) and represents the threshold value for rejecting or sticking to H_0;
ii. you analyze your data by computing some effect *e* using the statistic in your statistical hypotheses;
iii. you compute the so-called *probability of error p* how likely it is to find *e* or something that deviates from H_0 even more in your sample when, in the population, H_0 is true;
iv. you compare p_{critical} and *p* and decide: if $p < p_{\text{critical}}$, then you can reject H_0 and accept H_1 – otherwise, you must stick to H_0.

For example, if in your sample the mean length difference between subjects and direct objects is 1.4 syllables, then you compute the probability of error *p* to find this difference of 1.4 syllables or an even larger difference when you in fact don't expect any such difference (because that is what H_0 predicts). Then, there are two possibilities:

- if this probability p of a 1.4-syllable difference is smaller than $p_{critical}$ of 0.05, then you can reject the H_0 that there is no difference between subjects and direct objects in the population. In the results section of your paper, you can then write that you found a significant difference between the means in your sample, and in the discussion section of your paper you would discuss what kinds of implications this has, etc.
- if this probability p is equal to or larger than $p_{critical}$ of 0.05, then you cannot reject the H_0 that there is no difference between subjects and direct objects in the population. In the results section of your paper, you would then state that you have not found a significant difference between the lengths in your sample. In the discussion part of your paper, you should then discuss the implications of this finding as well as speculate or reason about why there was no significant difference – there may have been outliers in the corpus data or in the experiment (because subjects reacted strangely to particular stimuli, coding errors, etc. (*Outliers* are values in the sample that are rather untypical given the rest of the sample.)

Two aspects of this logic are very important: First, the fact that an effect is significant does not necessarily mean that it is an important effect despite what the everyday meaning of *significant* might suggest. The word *significant* is used in a technical sense here, meaning the effect (here, the difference) is large enough for us to assume that, given the size of the sample(s), it is probably not a random difference. Second, just because you accept H_1 given a significant result, that does not mean that you have *proven* H_1. This is because there is still the probability of error p that the observed result *has* come about even though H_0 is correct – the probability of error p is just small enough to *accept* H_1, but not to *prove* it.

This line of reasoning may appear a bit confusing at first especially since we suddenly talk about two different probabilities. One is the probability of 5% (to which the other probability is compared), that other probability is the probability to obtain the observed result when H_0 is correct. The former, the significance level $p_{critical}$, is *defined before data are obtained* whereas the latter, the probability of error, is the so-called p-value and *computed on the basis of the data*. Why is this probability called probability of error? It is because – recall from above – it is the probability to err when you accept H_1 given the observed data. Sometimes, you will find that people use different wordings for different p-values:

- $p < 0.001$ is sometimes referred to as *highly significant* and indicated with ***;
- $0.001 \leq p < 0.01$ is sometimes referred to as *very significant* and indicated with **;
- $0.01 \leq p < 0.05$ is sometimes referred to as *significant* and indicated with *;
- $0.05 \leq p < 0.1$ is sometimes referred to as *marginally significant* and indicated with *ms* or a period but since such *p*-values are larger than the usual standard of 5%, calling such results marginally significant amounts, polemically speaking at least, to saying "Look, I didn't really get the significant results I was hoping for, but they are still pretty nice, don't you think?", which is why I typically discourage the use of this expression.

Warning/advice
You must never change your hypotheses *after* you have obtained your results and then sell your study as successful support of the 'new' H_1. Also, you must never explore a data set – the nicer way to say 'fish for something useable' – and, when you then find something significant, sell this result as a successful test of a 'previously formulated' H_1. You may of course explore a data set in search of patterns and hypotheses, but if a data set generates a hypothesis, you must test that hypothesis with different data.

But while we have seen above how this comparison of the two probabilities contributes to the decision in favor of or against H_1, it is still unclear how this *p*-value is computed.

3.4.1. One-tailed p-*values from discrete probability distributions*

Let's assume you and I decided to toss a coin 100 times. If we get heads, I get one dollar from you – if we get tails, you get one dollar from me. Before this game, you formulate the following hypotheses:

Text H_0: Stefan does not cheat: the probability for heads and tails is 50% vs. 50%.
Text H_1: Stefan cheats: the probability for heads is larger than 50%.

This scenario can be easily operationalized using frequencies:

Statistical H_0: Stefan will win just as often as I will, namely 50 times.
Statistical H_1: Stefan will win more often than I will, namely more than 50 times.

Now my question: when we play the game and toss the coin 100 times, after which result will you suspect that I cheated?

THINK
BREAK

- when you lost 51 times (probably not …)?
- when you lost 55 times? when you lost 60 times? (maybe …)?
- when you lost 80 times or even more often? (most likely …)?

Maybe without realizing it, you are currently thinking along the lines of significance tests. Let's make this more concrete (by assuming you lost 60 times) and also paraphrase it in terms of the above four steps of the null-hypothesis significance testing paradigm:

i. let's assume you set the significance level $p_{critical}$ to its usual value of 0.05;
ii. you observe the effect *e*, namely that you lose 60 times;
iii. you (try to) compute the so-called probability of error *p* how likely it is to lose 60 times or more often in the sample (our game of 100 tosses) when H_0 is true and you should have lost 50 times. Why "60 times or more often"? Well above we said

> you compute the so-called *probability of error p* how likely it is to find *e* or something that deviates from H_0 even more in your sample when, in the population, H_0 is true;

iv. if you can compute *p*, you compare $p_{critical}$ and *p* and decide what to believe: if $p < p_{critical}$, then you can reject H_0, accept your H_1, and accuse me of cheating – otherwise, you must stick to H_0 and accept your losses.

Thus, you must ask yourself how and how much does the observed result deviate from the result expected from H_0. Obviously, your number of losses is larger: $60 > 50$. Thus, the results that deviate from H_0 that much or even more in the predicted direction are those where you lose 60 times or more often: 60 times, 61 times, 62, times, …, 99 times, and 100 times. In a

more technical parlance, you set the significance level to 0.05 and ask yourself "how likely is it that Stefan did not cheat but still won 60 times although he should only have won 50 times?" This is exactly the logic of significance testing.

It is possible to show that the probability p to lose 60 times or more just by chance – i.e., without me cheating – is 0.02844397, i.e., 2.8%. Since this p-value is smaller than 0.05 (or 5%), you can now accuse me of cheating. If we had been good friends, however, so that you would not have wanted to risk our friendship by accusing me of cheating prematurely and had set the significance level to 1%, then you would *not* be able to accuse me of cheating, since 0.02844397 > 0.01.

This example has hopefully clarified the overall logic even further, but what is probably still unclear is how this p-value is computed. To illustrate that, let us reduce the example from 100 coin tosses to the more manageable amount of three coin tosses. In Table 9, you find all possible results of three coin tosses and their probabilities provided that H_0 is correct and the chance for heads/tails on every toss is 50%. More specifically, the three left columns represent all possible results, column 4 and column 5 show how many heads and tails are obtained in each of the eight possible results, and the rightmost column lists the probability of each possible result. (I will explain the four boxes in the right half shortly.) As you can see, these are all the same, 0.125. Why is that so?

Two easy ways to explain this are conceivable, and both of them require you to understand the crucial concept of *independence*.

Table 9. All possible results of three coin tosses and their probabilities (when H_0 is correct)

Toss 1	Toss 2	Toss 3	# heads	# tails	p_{result}
heads	heads	heads	3	0	0.125
heads	heads	tails	2	1	0.125
heads	tails	heads	2	1	0.125
heads	tails	tails	1	2	0.125
tails	heads	heads	2		0.125
tails	heads	tails	1	2	0.125
tails	tails	heads	1	2	0.125
tails	tails	tails	0	3	0.125

The first one involves understanding that, according to H_0, the probability of heads and tails is the same on every trial and that all trials are independent of each other. This notion of independence is important: trials are

independent of each other when the outcome of one trial (here, one toss) does not influence the outcome of any other trial (i.e., any other toss). Similarly, samples are independent of each other when there is no meaningful way in which you can match values from one sample onto values from another sample. For example, if you randomly sample 100 transitive clauses out of a corpus and count their subjects' lengths in syllables, and then you randomly sample 100 *different* transitive clauses from the same corpus and count their direct objects' lengths in syllables, then the two samples – the 100 subject lengths and the 100 object lengths – are independent. If, on the other hand, you randomly sample 100 transitive clauses out of a corpus and count the lengths of the subjects and the objects in syllables, then the two samples – the 100 subject lengths and the 100 object lengths – are dependent because you can match up the 100 subject lengths onto the 100 object lengths perfectly by aligning each subject with the object from the very same clause. Similarly, if you perform an experiment twice with the same subjects, then the two samples made up by the first and the second experimental results are dependent, because you can match up each subject's data point in the first experiment with the same subject's data point in the second. This notion will become very important later on.

Returning to the three coin tosses: since there are eight different outcomes of three tosses that are all independent of each other – i.e. equally probable – the probability of each of the eight outcomes is $^1/_8 = 0.125$.

The second way to understand the rightmost column of Table 9 involves computing the probability of each of the eight events separately. For the first row that means the following: the probability to get head in the first toss, in the second, in the third toss is always 0.5. Since the tosses are independent of each other, you obtain the probability to get heads three times in a row by multiplying the individual events' probabilities: $0.5 \cdot 0.5 \cdot 0.5 = 0.125$ (the multiplication rule in probability theory). Analogous computations for every row show that the probability of each result is 0.125. Thus, we can show that H_0 predicts that each of us should win 1.5 times on average (i.e., if we played the three-toss game 100 times).

Now imagine you lost two out of three times. If you had again set the level of significance to 5%, could you accuse me of cheating?

THINK
BREAK

Of course not. Let me first ask again which events need to be considered. The observed result – that you lost two times – and the result(s) that deviate(s) even more from H_0 in the predicted direction. This is easy here: the only such result is that you lose all three times. Let us compute the sum of the probabilities of these events.

As you can see in column 4, there are three results in which you lose two times in three tosses: H H T (row 2), H T H (row 3), and T H H (row 5). Thus, the probability to lose exactly two times is 0.125+0.125+0.125 = 0.375, and that is already much much more than your level of significance 0.05 allows. However, to that you still have to add the probability of the event that deviates even more from H_0, which is another 0.125 (row 1); all these events and their probabilities are highlighted with the four boxes. If you add this all up, the probability p to lose two or more times in three tosses when H_0 is true is 0.5. This is ten times as much as the level of significance so there is no way that you can accuse me of cheating. Note that even if you had lost all three tosses, you could still not accuse me of cheating, because the probability of that happening when H_0 is true is still 0.125

We can also represent this logic graphically and at the same time go back to larger numbers of tosses. Figure 4 has six panels, one for 3 tosses, one for 6, one for 12, and then 25, 50, and 100. In each, the summed probabilities for all possible numbers of heads given the number of tosses made are represented as bars, and the most extreme result (I always win) is represented with a grey bar and an arrow pointing to it. In the cases of 3 and 6 tosses, I also plotted the probabilities of these events on top of the bars.

Thus, if you lost more often than you should have according to H_0 and you want to determine the probability of losing as many times and even more often, you move from the expectation of H_0, which is in the middle (along the x-axis) of the graph, away to the observed result (say, at $x = 3$) and add the length of that bar to the lengths of all other bars you encounter if you continue to move in the same direction, where here there is only one bar at $x = 3$ so you're done immediately.

Figure 4 also illustrates another very important point. First, recall that the basic distribution underlying this data is a discrete and non-normal probability distribution, namely 0.5 (heads) vs. 0.5 (tails). Second, as the numbers of tosses in our games increase, the probabilities of the possible results look more and more like the bell-shaped curve we know from normal distributions. Thus, even though the underlying distribution is not normal, once the sample size becomes large enough, we still get a bell-shaped curve. This also means that, if the data under investigation are distributed in a way that is sufficiently similar to the normal distribution (or

another one of several widely used probability density functions, such as the *F*-, *t*-, or χ^2-distribution), then one does not have to compute, and sum over, exact probabilities as we did above, but one can approximate the *p*-value from parameters of equations underlying the above distributions; this is often called using *parametric tests*. Crucially, this approximation of a *p*-value on the basis of a function can be only as good as the data's distributional fit to the corresponding function. We will revisit this below.

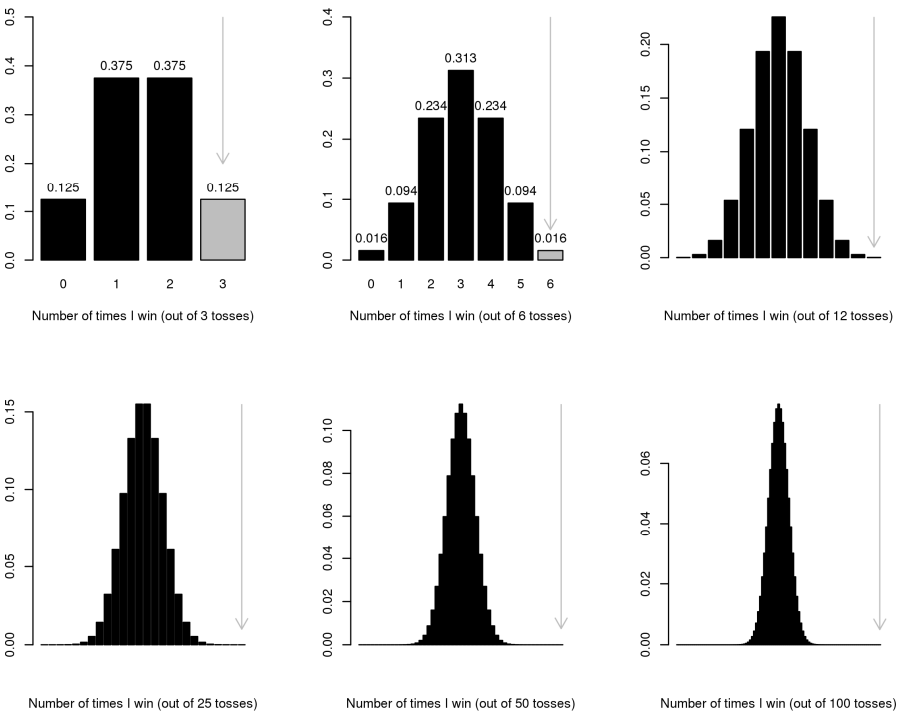

Figure 4. All probabilities of possible results of 3, 6, 12, 25, 50, 100 coin tosses and their probabilities (when H_0 is correct, one-tailed)

3.4.2. Two-tailed p-*values from discrete probability distributions*

Now, we have to add another perspective. In the last section, we were concerned with *directional H_1s*: your H_1 was "Stefan cheats: the probability for heads is larger than 50% [and not just different from 50%]." The kind of significance test we discussed is correspondingly called *one-tailed tests* because you were only interested in one direction in which the observed

result deviates from the expected result (say because you knew for sure you didn't cheat). Thus, when you summed up the bar lengths in Figure 4 you only moved away from H_0's expectation in one direction.

However, often you only have a *non-directional H_1*. In such cases, you have to look at both ways in which results may deviate from the expected result. Let us return to the scenario where you and I toss a coin three times, but this time we also have an impartial observer who has no reason to suspect that only I would be cheating. He therefore formulates the following hypotheses (with a significance level of 0.05):

Statistical H_0: Stefan will win just as often as the other player, namely 50 times (or "Both players will win equally often").
Statistical H_1: Stefan will win more or less often than the other player (or "The players will not win equally often").

Imagine now again you lost three times. The observer now asks himself whether one of us should be accused of cheating. As before, he needs to determine which events to consider and he also uses a table of all possible results to help him figure things out. Consider, therefore, Table 10.

Table 10. All possible results of three coin tosses and their probabilities (when H_0 is correct)

Toss 1	Toss 2	Toss 3	# heads	# tails	p_{result}
heads	heads	heads	3	0	0.125
heads	heads	tails	2	1	0.125
heads	tails	heads	2	1	0.125
heads	tails	tails	1	2	0.125
tails	heads	heads	2		0.125
tails	heads	tails	1	2	0.125
tails	tails	heads	1	2	0.125
tails	tails	tails	0	3	0.125

First, the observer considers the observed result that *you* lost three times, which is listed in row 1 and arises with a probability of 0.125. But then he also considers the probabilities of events deviating from H_0 just as much or even more. With a directional H_1, you moved from H_0 only in one direction – but this time there is no directional hypothesis so the observer also looks for deviations just as large or even larger in the other direction of H_0's expectation. As you can see in Table 10, there is another deviation from H_0 that is just as extreme, namely that *I* lose three times. Since the

observer only has a non-directional hypothesis, he includes the probability of that event, too, arriving at a cumulative probability of 0.25. This logic is graphically represented in Figure 5 in the same way as above.

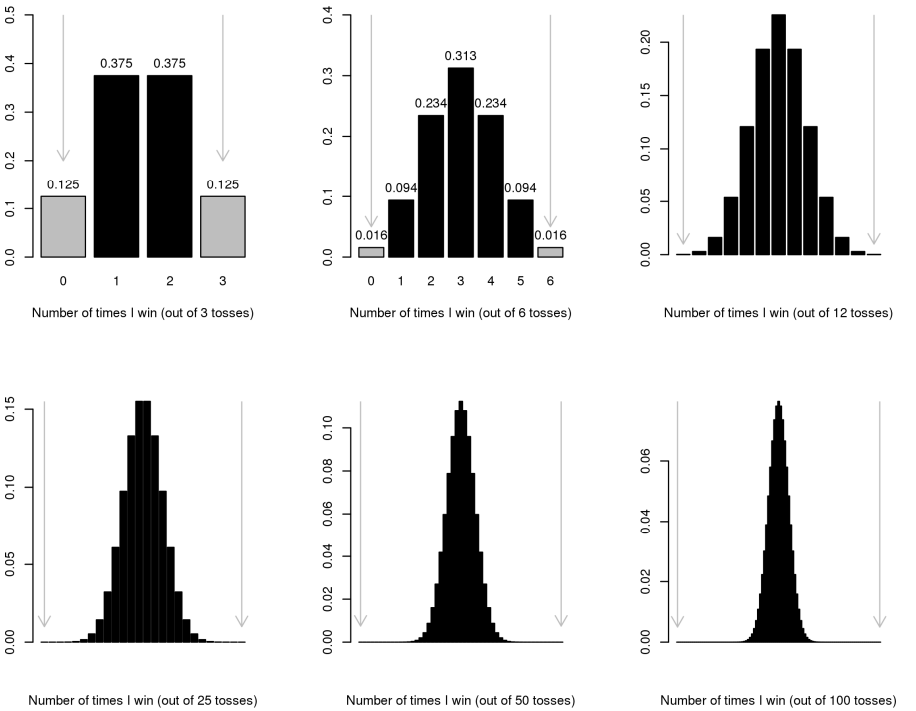

Figure 5. All probabilities of possible results of 3, 6, 12, 25, 50, 100 coin tosses and their probabilities (when H_0 is correct, two-tailed)

Note that when you tested your directional H_1, you looked at the result 'you lost three times', but when the impartial observer tested his non-directional H_1, he looked at the result 'somebody lost three times.' This has one very important consequence: when you have prior knowledge about a phenomenon that allows you to formulate a directional, and not just a non-directional, H_1, then the result you need for a significant finding can be less extreme than if you only have a non-directional H_1. In most cases, it will be like here: the *p*-value you get for a result with a directional H_1 is half of the *p*-value you get for a result with a non-directional H_1. Prior knowledge is rewarded, which will be illustrated once more now.

Let us now return to the example game involving 100 tosses. Again, we first look at the situation through your eyes (directional H_1), and then, sec-

ond, through those of an impartial observer (non-directional H_1), but this time you and the observer try to determine *before the game* which results are so extreme that one will be allowed to adopt the H_1. We begin with your perspective: In Figure 6, you find the by now familiar graph for 100 tosses with the expected frequency for heads of 50. (The meaning of the black lines will be explained presently.)

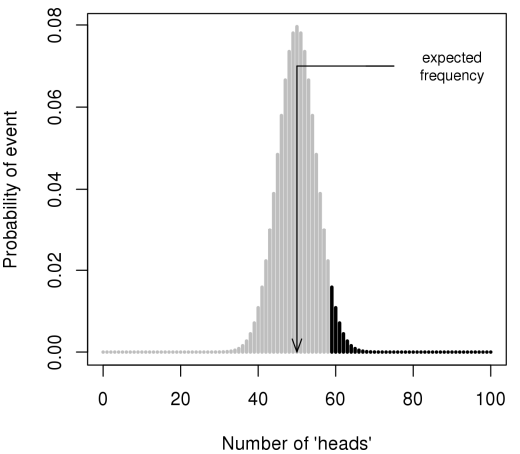

Figure 6. All possible results of 100 coin tosses and their probabilities (when H_0 is correct, one-tailed H_1)

Above, we had an empirical result whose *p*-value we were interested in, and in order to get that *p*-value, we moved from the expected H_0 results to the extreme values. Now we want to determine, but not exceed, a *p*-value before we have results and have to proceed the other way round: from an extreme point to the expectation of H_0. For example, to determine how many times you can lose without getting a cumulative probability exceeding 0.05, you begin at the most extreme result on the right – that you lose 100 times – and begin to add the lengths of the bars. (Of course, you would compute that and not literally measure lengths.) The probability that you lose all 100 tosses is $7.8886 \cdot 10^{-31}$. To that you add the probability that you lose 99 out of 100 times, the probability that you lose 98 out of 100 times, etc. When you have added all probabilities until 59 times heads, then the sum of all these probabilities reaches 0.0443; all these are represented in black in Figure 6. Since the probability to get 58 heads out of 100 tosses is 0.0223, you cannot add this event's probability to the others anymore without exceeding the level of significance value of 0.05. Put differently, if you don't want to cut off more than 5% of the summed bar lengths, then you

must stop adding probabilities at $x = 59$. You conclude: if Stefan wins 59 times or more often, then I will accuse him of cheating, because the probability of that happening is the largest one that is still smaller than 0.05.

Now consider the perspective of the observer shown in Figure 7, which is very similar, but not completely identical to Figure 6. The observer also begins with the most extreme result, that I get heads every time: $p_{100\ \text{heads}} \approx 7{,}8886 \cdot 10^{-31}$. But since the observer only has a non-directional H_1, he must also include the probability of the opposite, equally extreme result, that we get heads 0 times. For each additional number of heads – 99, 98, etc. – the observer must now also add the corresponding opposite results – 1, 2, etc. Once the observer has added the probabilities 61 times heads / 39 times tails and 39 times heads / 61 times tails, then the cumulative sum of the probabilities reaches 0.0352 (cf. the black bars in Figure 7).

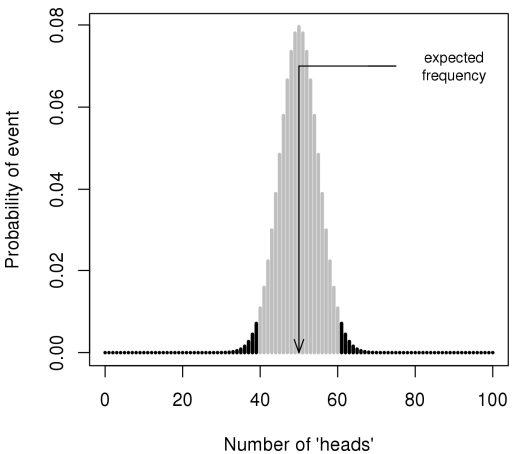

Figure 7. All possible results of 100 coin tosses and their probabilities (when H_0 is correct, two-tailed H_1)

Since the joint probability for the next two events – 60 heads / 40 tails and 40 heads / 60 tails – is 0.0217, the observer cannot add any further results without exceeding the level of significance of 0.05. Put differently, if the observer doesn't want to cut off more than 5% of the summed bar lengths on both sides, then he must stop adding probabilities by going from right to the left at $x = 61$ and stop going from the left to right at $x = 39$. He concludes: if Stefan or his opponent wins 61 times or more often, then someone is cheating (most likely the person who wins more often).

Again, observe that in the same situation the person with the directional H_1 needs a less extreme result to be able to accept it than the person with a

non-directional H_1: with the same level of significance, *you* can already accuse me of cheating when you lose 59 times (only 9 times more often than the expected result) – *the impartial observer* needs to see someone lose 61 times (11 times more often than the expected result) before he can start accusing someone. Put differently, if you lose 60 times, you can accuse me of cheating, but the observer cannot. This difference is very important and we will use it often.

While reading the last few pages, you probably sometimes wondered where the probabilities of events come from: How do we know that the probability to get heads 100 times in 100 tosses is $7.8886 \cdot 10^{-31}$? Essentially, those are computed in the same way as we handled Table 9 and Table 10, just that we do not write results up anymore because the sample space is too huge. These values were therefore computed with R on the basis of the so-called binomial distribution. You can easily compute the probability that one out of two possible events occurs *x* out of *s* times when the event's probability is *p* in R with the function `dbinom`.[6] The arguments of this function we deal with here are:

- *x*: the frequency of the event (e.g., three times heads);
- *s*: the number of trials the event could occur (e.g., three tosses);
- *p*: the probability of the event in each trial (e.g., 50%).

You know that the probability to get three heads in three tosses when the probability of head is 50% is 12.5%. In R:

```
> dbinom(3, 3, 0.5)¶
[1] 0.125
```

As a matter of fact, you can compute the probabilities of all four possible numbers of heads – 0, 1, 2, and 3 – in one line (because, as we will see below, sequences of integers can be defined with a colon):

```
> dbinom(0:3, 3, 0.5)¶
[1] 0.125 0.375 0.375 0.125
```

In a similar fashion, you can also compute the probability that heads will occur two or three times by summing up the relevant probabilities:

6. I will explain how to install R etc. in the next chapter. It doesn't really matter if you haven't installed R and/or can't enter or understand the above input yet. We'll come back to this …

```
> sum(dbinom(2:3, 3, 0.5))¶
[1] 0.5
```

Now you do the same for the probability to get 100 heads in 100 tosses,

```
> dbinom(100, 100, 0.5)¶
[1] 7.888609e-31
```

the probability to get heads 58 or more times in 100 tosses (which is larger than 5% and does not allow you to accept a one-tailed/directional H_1),

```
> sum(dbinom(58:100, 100, 0.5))¶
[1] 0.06660531
```

the probability to get heads 59 or more times in 100 tosses (which is smaller than 5% and does allow you to accept a one-tailed/directional H_1):

```
> sum(dbinom(59:100, 100, 0.5))¶
[1] 0.04431304
```

In fact, you would not have to do this by trial and error as the above may suggest. You can use the function `qbinom` to get the largest number of heads whose cumulative probability with every even more extreme result does not exceed 0.05, and you can see that this matches the above finding:

```
> qbinom(0.05, 100, 0.5, lower.tail=FALSE)¶
[1] 58
```

For two-tailed tests, you can do the same, e.g., compute the probability to get heads 40 times or less often, or 60 times and more often (which is larger than 0.05 and does not allow you to accept a two-tailed/non-directional H_1):

```
> sum(dbinom(c(0:40, 60:100), 100, 0.5))¶
[1]  0.05688793
```

Here's the probability to get heads 39 times or less often, or 61 times and more often (which is smaller than 0.05 and allows you to accept a two-tailed/non-directional H_1):

```
> sum(dbinom(c(0:39, 61:100), 100, 0.5))¶
[1] 0.0352002
```

Again, no need to do this by manual trial and error. You can again use `qbinom` to get the largest number of heads whose cumulative probability with every even more extreme result does not exceed 0.05 – the only complication is that since you want to 'add bar lengths' on two sides and the bar lengths are identical on both sides (because the curves in Figure 6 and Figure 7 are symmetric), you must get the result that does not exceed 0.05 when you add both sides, i.e. when one side does not exceed 0.025. Then, you again see that this matches our above manual finding:

```
> qbinom(0.05/2, 100, 0.5, lower.tail=FALSE)¶
[1] 60
```

3.4.3. Extension: continuous probability distributions

In the above examples, we always had only one variable with two levels: TOSS: *HEADS* vs. *TAILS*. Unfortunately, life is usually not that easy. On the one hand, we have seen above that our categorical variables will often involve more than two levels. On the other hand, if the variable in question is ratio-scaled, then the computation of the probabilities of all possible states or levels is not possible. For example, you cannot compute the probabilities of all possible reaction times to a stimulus. For this reason and as mentioned above, many statistical techniques do not compute an exact *p*-value as we did, but are based on the fact that, as the sample size increases, the probability distributions of events begin to approximate those of mathematical distributions whose functions/equations and properties are very well known. Four such distributions will be important for Chapters 4 and 5:

– the standard normal distribution with *z*-scores (`norm`);
– the *t*-distribution (`t`);
– the *F*-distribution (`f`);
– the chi-squared- / χ^2-distribution (`chisq`).

For each of these distributions, just like for `binom` from above, there is a function whose name begins with *q* and ends with the above function name (i.e. `qnorm`, `qt`, `qf`, `qchisq`) and a function whose name begins with *p* and ends with the above function name (i.e. `pnorm`, `pt`, `pf`, `pchisq`). The former compute the *quantile functions* of these (four and other) probability distributions whereas the latter compute the inverses of these, the so-called *cumulative distribution functions*. We can explain this relatively easily on the

basis of Figure 8, both panels of which plot the density function of the standard normal distribution.

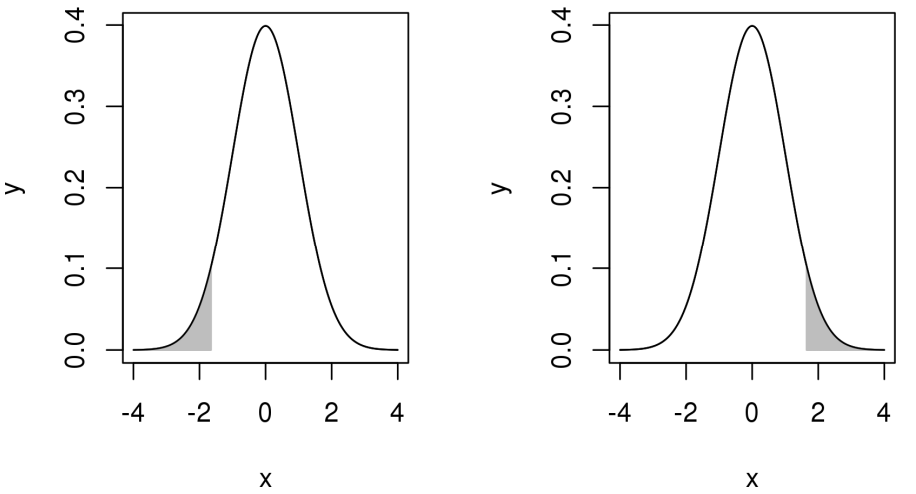

Figure 8. Density function of the standard normal distribution with $p_{\text{one-tailed}} = 0.05$

In Figure 6, we were interested in determining how much a result can deviate from the expected result of, there, 50 heads and 50 tails, without being significant, where 'being significant' meant arising with a cumulative probability of less than 0.05 of the whole result space. In that case, we added up lengths of the bars that make up the curve of the binomial distribution (using `dbinom`) or directly identified the largest number of heads whose cumulative probability with more extreme results did not exceed 0.05 (with `qbinom`).

```
> sum(dbinom(58:100, 100, 0.5))¶
[1] 0.06660531
> qbinom(0.05, 100, 0.5, lower.tail=FALSE)¶
[1] 58
```

For the continuous distributions of the kind illustrated in Figure 8, there are no bar lengths to add up, but the corresponding notion is the area under the curve, which is defined as 1 and of which any value on the *x*-axis can cut something off to the left or to the right. For such computations, we can again use functions with `q` and `p`. For example, if we want to know which *x*-value cuts of 5%, i.e. 0.05, of the left area under the curve, we can com-

pute it in the following ways with `qnorm`:

```
> qnorm(0.05, lower.tail=TRUE)¶
[1] -1.644854
> qnorm(1-0.95, lower.tail=TRUE)¶
[1] -1.644854
> qnorm(0.95, lower.tail=FALSE)¶
[1] -1.644854
> qnorm(1-0.05, lower.tail=FALSE)¶
[1] -1.644854
```

Thus, the grey area under the curve in the left panel of Figure 8 in the range $-\infty \leq x \leq -1.644854$ corresponds to 5% of the area under the curve. Since the standard normal distribution is symmetric, the same is true of the grey area under the curve in the right panel in the range $1.644854 \leq x \leq \infty$.

```
> qnorm(0.95, lower.tail=TRUE)¶
[1] 1.644854
> qnorm(1-0.05, lower.tail=TRUE)¶
[1] 1.644854
> qnorm(0.05, lower.tail=FALSE)¶
[1] 1.644854
> qnorm(1-0.95, lower.tail=FALSE)¶
[1] 1.644854
```

These are one-tailed tests because you only look at one side of the curve, either the left (when `lower.tail=TRUE` in the left panel) or the right (when `lower.tail=FALSE` in the right panel). For corresponding two-tailed tests at the same significance level of 0.05, you would have to proceed as with `binom` and consider both areas under the curve (as in Figure 9), namely 2.5% on each edge to arrive at 5% altogether. Thus, to get the *x*-axis values that *jointly* cut off 5% under the curve, this is what you could enter into R:

```
> qnorm(0.025, lower.tail=TRUE)¶
[1] -1.959964
> qnorm(1-0.975, lower.tail=TRUE)¶
[1] -1.959964
> qnorm(0.975, lower.tail=FALSE)¶
[1] -1.959964
> qnorm(1-0.025, lower.tail=FALSE)¶
[1] -1.959964
```

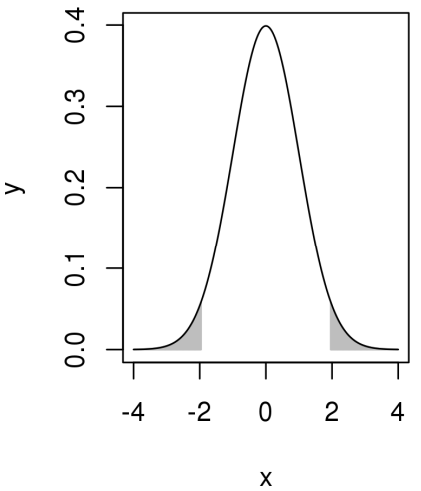

Figure 9. Density function of the standard normal distribution with $p_{\text{two-tailed}} = 0.05$

```
> qnorm(0.975, lower.tail=TRUE)¶
[1] 1.959964
> qnorm(1-0.025, lower.tail=TRUE)¶
[1] 1.959964
> qnorm(0.025, lower.tail=FALSE)¶
[1] 1.959964
> qnorm(1-0.975, lower.tail=FALSE)¶
[1] 1.959964
```

Again, you see that with non-directional two-tailed tests you need a more extreme result for a significant outcome: a value of -1.7 is less than -1.644854 and would be significant in a one-tailed test (if you had predicted the negative direction), but that same value is greater than -1.959964 and thus not small enough for a significant two-tailed test. In sum, with the q-functions we determine the minimum one- or two-tailed statistic we need to obtain a particular *p*-value. For one-tailed tests, you typically use $p = 0.05$; for two-tailed tests $p = {}^{0.05}/_{2} = 0.025$ on each side. The functions whose names start with *p* do the opposite of those beginning with *q*: with them, you determine which *p*-value our statistic corresponds to. The following two lines get you *p*-values for one-tailed tests (cf. Figure 8 again):

```
> pnorm(-1.644854, lower.tail=TRUE)¶
[1] 0.04999996
> pnorm(1.644854, lower.tail=FALSE)¶
[1] 0.04999996
```

For the two-tailed test, you of course must multiply the probability by two because whatever area under the curve you get, you must consider it on both sides of the curve. (cf. Figure 9 again):

```
> 2*pnorm(-1.959964, lower.tail=TRUE)¶
[1] 0.05
> 2*pnorm(1.959964, lower.tail=FALSE)¶
[1] 0.05
```

The other p/q-functions work in the same way, but will require some additional information, namely so-called degrees of freedom. I will not explain this notion here in any detail but instead cite Crawley's (2002: 94) rule of thumb: "[d]egrees of freedom [*df*] is the sample size, *n*, minus the number of parameters, *p* [not related to the other *p*s above, STG], estimated from the data." For example, if you compute the mean of four values, then $df = 3$ because when you want to make sure you get a particular mean out of four values, then you can choose three values freely, but the fourth one is then set. If you want to get a mean of 8, then the first three values can vary freely and be 1, 2, and 3, but then the last one must be 26. Degrees of freedom are the way in which sample sizes and the amount of information you squeeze out of a sample are integrated into the significance test.

The parametric tests that are based on the above distributions are usually a little easier to compute (although this is usually not an important point anymore, given the computing power of current desktop computers) and more powerful, but they have the potential problem alluded to above. Since they are only estimates of the real *p*-value based on the equations defining *z*-/*t*-/*F*-/χ^2-values, their accuracy is dependent on how well these equations reflect the distribution of the data. In the above example, the binomial distribution in Figure 4 and Figure 5 and the normal distribution in Figure 8 and Figure 9 are extremely similar, but this may be very different on other occasions. Thus, parametric tests make distributional assumptions – the most common one is in fact that of a normal distribution – so you can use such tests only if the data you have meet these assumptions. If they don't, then you must use a so-called *non-parametric test* or an exact test (as we have done for the coin tosses above) or a permutation test or other resampling methods. For nearly all tests introduced in Chapters 4 and 5 below, I will list the assumptions which you have to test before you can apply the test, explain the test itself with the computation of a *p*-value, and illustrate how you would summarize the result in the third (results) part of the written version of your study. I can already tell you that you should always provide the sample sizes, the obtained effect (such as the mean, the

percentage, the difference between means, etc.), the name of the test you used, its statistical parameters, the *p*-value, and your decision (in favor of or against H_1). The interpretation of these findings will then be discussed in the fourth and final section of your study.

Recommendation(s) for further study
Good and Hardin (2012: Ch. 1, 2, and 3) for many interesting and practically relevant tips as well as Good and Hardin (2012: Ch. 8) on information you should provide in your methods and results sections

Warning/advice
Do not give in to the temptation to use a parametric test when its assumptions are not met. What have you gained when you do wrong tests and either get slammed by reviewers or, worse even, get published with wrong results that are cited because of your methodological mistake(s)?

4. The design of a factorial experiment: introduction

In this section, we will deal with a few fundamental rules for the design of experiments.[7] The probably most central notion in this section is the token set (cf. Cowart 1997). I will distinguish two kinds of token sets, schematic token sets and concrete token sets. A *schematic token set* is typically a tabular representation of all experimental conditions. To explain this more clearly, let us return to the above example of particle placement.

Let us assume you want to investigate particle placement not only on the basis of corpus data, but also on the basis of experimental data. For instance, you might want to determine how native speakers of English rate the acceptability of sentences (the dependent variable ACCEPTABILITY) that differ with regard to the constructional choice (the first independent variable CONSTRUCTION: *VPO* vs. *VOP*) and the part of speech of the head of the direct object (the second independent variable OBJPOS: *PRONOMINAL* vs. *LEXICAL*).[8] Since there are two independent variables for each of the two levels, there are $2 \cdot 2 = 4$ experimental conditions. This set of experimental conditions is the schematic token set, which is represented in two different forms in Table 11 and Table 12. The participants/subjects of course never

7. I will only consider *factorial designs*, where every variable level is combined with every other variable level, but most of the rules discussed also apply to other designs.
8. For expository reasons, I only assume two levels of OBJPOS.

get to see the schematic token set. For the actual experiment, you must develop concrete stimuli – a *concrete token set* that realizes the variable level combinations of the schematic token set.

Table 11. Schematic *token set* for CONSTRUCTION × OBJPOS 1

	OBJPOS: *PRONOMINAL*	OBJPOS: *LEXICAL*
CONSTRUCTION: *VPO*	V Part pron. $NP_{dir.\ obj.}$	V Part lexical $NP_{dir.\ obj.}$
CONSTRUCTION: *VOP*	V pron. $NP_{dir.\ obj.}$ Part	V lexical $NP_{dir.\ obj.}$ Part

Table 12. Schematic *token set* for CONSTRUCTION × OBJPOS 2

Experimental condition	CONSTRUCTION	OBJPOS
1	*VPO*	*PRONOMINAL*
2	*VPO*	*LEXICAL*
3	*VOP*	*PRONOMINAL*
4	*VOP*	*LEXICAL*

However, both the construction of such concrete token sets and the actual presentations of the concrete stimuli are governed by a variety of rules that aim at minimizing undesired sources of noise in the data. Three such sources are particularly important:

- *knowledge of what the experiment is about*: you must make sure that the participants in the experiment do not know what is being investigated before or while they participate (after the experiment you can of course tell them). This is important because otherwise the participants might make their responses socially more desirable or change the responses to 'help' the experimenter.
- *undesirable experimental effects*: you must make sure that the responses of the subjects are not influenced by, say, habituation to particular variable level combinations. This is important because in the domain of, say, acceptability judgments, Nagata (1987, 1989) showed that such judgments can change because of repeated exposure to stimuli and this may not be what you're interested in.
- *evaluation of the results*: you must make sure that the responses of the subjects can be interpreted unambiguously. Even a large number of willing and competent subjects is useless if your design does not allow for an appropriate evaluation of the data.

In order to address all these issues, you have to take the rules in (4) to (12) under consideration. Here's the first one in (4):

(4) The stimuli of each individual concrete token set differ with regard to the variable level combinations under investigation (and ideally only with regard to these and nothing else).

Consider Table 13 for an example. In Table 13, the stimuli differ only with respect to the two independent variables. If this was not the case (for example, because the left column contained the stimuli *John picked up it* and *John brought it back*) and you found a difference of acceptability between them, then you would not know what to attribute this difference to – the different construction (which would be what this experiment is all about), the different phrasal verb (that might be interesting, but is not what is studied here), to an interaction of the two … (4) is therefore concerned with the factor 'evaluation of the results'.

Table 13. A concrete token set for CONSTRUCTION × OBJPOS 1

	OBJPOS: *PRONOMINAL*	OBJPOS: *LEXICAL*
CONSTRUCTION: *VPO*	John picked up it.	John picked up the keys.
CONSTRUCTION: *VOP*	John picked it up.	John picked the keys up.

When creating the concrete token sets, it is also important to consider variables which you are not interested in but which may make it difficult to interpret the results with regard to the variables that you are interested in. In the present case, for example, the choice of the verbs and the direct objects may be important. For instance, it is well known that particle placement is also correlated with the concreteness of the referent of the direct object. There are different ways to take such variables, or sources of variation, into account. One is to make sure that 50% of the objects are abstract and 50% are concrete for each experimental condition in the schematic token set (as if you introduced an additional independent variable). Another one is to use only abstract or only concrete objects, which would of course entail that whatever you find in your experiment, you could strictly speaking only generalize to that class of objects.

Recommendation(s) for further study
Good and Hardin (2012: 31ff.) and Good (2005: Ch. 5)

(5) You must use more than one concrete token set, ideally as many concrete token sets as there are variable level combinations (or a multiple thereof).

One reason for (5) is that, if you only used the concrete token set in Table 13, then a conservative point of view would be that you could only generalize to other sentences with the transitive phrasal verb *pick up* and the objects *it* and *the book*, which would probably not be the most interesting study ever. Thus, the first reason for (5) is again concerned with the factor 'evaluation of results', and the remedy is to create different concrete token sets with different verbs and different objects such as those shown in Table 14 and Table 15, which also must conform to (4).

Table 14. A concrete token set for CONSTRUCTION × OBJPOS 2

	OBJPOS: *PRONOMINAL*	OBJPOS: *LEXICAL*
CONSTRUCTION: *VPO*	Mary brought back him.	Mary brought back his dad.
CONSTRUCTION: *VOP*	Mary brought him back.	Mary brought his dad back.

Table 15. A concrete token set for CONSTRUCTION × OBJPOS 3

	OBJPOS: *PRONOMINAL*	OBJPOS: *LEXICAL*
CONSTRUCTION: *VPO*	I eked out it.	I eked out my living.
CONSTRUCTION: *VOP*	I eked it out.	I eked my living out.

A second reason for (5) is that if you only used the concrete token set in Table 13, then subjects would probably be able to guess the purpose of the experiment right away: since our token set had to conform to (4), the subject can identify the relevant variable level combinations quickly because those are the only things according to which the sentences differ. This immediately brings us to the next rule:

(6) Every subject sees maximally one item out of a concrete token set.

As I just mentioned, if you do not follow 0, the subjects might guess from the minimal variations within one concrete token set what the whole experiment is about: the only difference between *John picked up it* and *John picked it up* is the choice of construction. Thus, when subject X gets to see the variable level combination (CONSTRUCTION: *VPO* × OBJPOS: *PRONOMINAL*) in the form of *John picked up it*, then the other experimental

items of Table 13 must be given to other subjects. In that regard, both (5) and (6) are (also) concerned with the factor 'knowledge of what the experiment is about'.

(7) Every subject is presented every variable level combination.

The motivation for (7) are the factors 'undesirable experimental effects' and 'evaluation of the results'. First, if several experimental items you present to a subject only instantiate one variable level combination, then habituation effects may distort the results; this you could of course take into account by adding a variable to your analysis that mentions for each presentation of an experimental condition how often it has been presented already. Second, if you present one variable level combination to a subject very frequently and another one only rarely, then whatever difference you find between these variable level combinations may theoretically be due to the different frequencies of exposure and not due to the effects of the variable level combinations under investigation.

(8) Every subject gets to see every variable level combination more than once and equally frequently.
(9) Every experimental item is presented to more than one subject and to equally many subjects.

These rules are motivated by the factor 'evaluation of the results'. You can see what their purpose is if you think about what happens when you try to interpret a very unusual reaction by a subject to a stimulus. On the one hand, that reaction could mean that the item itself is unusual in some respect in the sense that every subject would react unusually to it – but you can't test that if that item is not also given to other subjects, and this is the reason for the rule in (9). On the other hand, the unusual reaction could mean that only this particular subject reacts unusually to that variable level combination in the sense that the same subject would react more 'normally' to other items instantiating the same variable level combination – but you can't test that if that subject does not see other items with the same variable level combination, and this is the reason for (8).

(10) The experimental items are interspersed with distractors / filler items; there are minimally as many filler items as real experimental items per subject, but ideally two or three times as many filler items as real experimental items per subject.

The reason for (10) is obviously 'knowledge of what the experiment is about': you do not want the subjects to be able to guess the purpose of the experiment (or have them *think* they know the purpose of the experiment) so that they cannot distort the results.[9]

An additional well-known factor that can distort results is the order in which items and distractors are presented. To minimize such effects, you must take into consideration the final two rules:

(11) The order of experimental and filler items is pseudorandomized.
(12) The order of experimental and filler items is pseudorandomized differently for every subject.

The rule in (11) requires that the order of experimental items and filler items is randomized using a random number generator, but it is not completely random – hence *pseudo*randomized – because the ordering resulting from the randomization must usually be 'corrected' such that

- the first stimulus (e.g., the first question on a questionnaire) is not an experimental item but a distractor;
- experimental items do not follow each other directly;
- ideally, experimental items exhibiting the same variable level combinations do not follow each other, which means that, after *John picked it up*, the next experimental item must not be *Mary brought him back* even if the two are interrupted by distractors.

The rule in (12) means that the order of stimuli must vary pseudorandomly across subjects so that whatever you find cannot be attributed to systematic order effects: every subject is exposed to a different order of experimental items and distractors. Hence, both (11) and (12) are concerned with 'undesirable experimental effects ' and 'evaluation of the results'. (This re-ordering of stimuli can be quite tedious, especially when your experiment involves many test items and subjects, which is why, once you are more proficient with R, it may be useful to write a function called, say, `stimulus.randomizer` to do this for you, which is how I do this.)

9. In many psychological studies, not even the person actually conducting the experiment (in the sense of administering the treatment, handing out the questionnaires, …) knows the purpose of the experiment. This is to make sure that the experimenter cannot provide unconscious clues to desired or undesired responses. An alternative way to conduct such so-called double-blind experiments is to use standardized instructions in the forms of videotapes or have a computer program provide the instructions.

Only after all these steps have been completed properly can you begin to print out the questionnaires and have subjects participate in an experiment. It probably goes without saying that you must carefully describe how you set up your experimental design in the methods section of your study. Since this is a rather complex procedure, we will go over it again in the following section.

One final remark about this before we look at another example. I know from experience that the previous section can have a somewhat discouraging effect. Especially beginners read this and think "how am I ever going to be able to set up an experiment for my project if I have to do all this? (I don't even know my spreadsheet software well enough yet …)" And it is true: I myself still need a long time before a spreadsheet for an experiment of mine looks the way it is supposed to. But if you do not go through what at first sight looks like a terrible ordeal, your results might well be, well, let's face it, crap! Ask yourself what is more discouraging: spending maybe several days on getting the spreadsheet right, or spending maybe several weeks on doing a simpler experiment and then having unusable results …

Warning/advice

You must be prepared for the fact that usually not all subjects answer all questions, give all the acceptability judgments you ask for, show up for both the first and the second test, etc. Thus, you should plan conservatively and try to get more subjects than you thought you would need in the first place. As mentioned above, you should still include these data in your table and mark them with NA. Also, it is often very useful to carefully examine the missing data for whether their patterning reveals something of interest (it would be very important if, say, one variable level combination accounted for 90% of the missing data or if 90% of the missing data were contributed by only two out of, say, 60 subjects).

5. The design of a factorial experiment: another example

Let us assume you want to investigate which variables determine how many elements a quantifier such as *some* refers to; consider (13):

(13) a. [$_{NP}$ some balls [$_{PP}$ in front of [$_{NP}$ the cat]]
b. [$_{NP}$ some balls [$_{PP}$ in front of [$_{NP}$ the table]]
c. [$_{NP}$ some cars [$_{PP}$ in front of [$_{NP}$ the building]]

Thus, the question is: are *some balls in front of the cat* as many balls as *some balls in* front *of the table*? Or: does *some balls in front of the table* mean as many balls as *some cars in front of the building* means cars? What – or more precisely, how many – does *some* mean? Your study of the literature may have shown that at least the following two variables influence the quantities that *some* denotes:

- OBJECT: the size of the object referred to by the first noun: *SMALL* (e.g. *ball*) vs. *LARGE* (e.g. *car*);
- REFPOINT: the size of the object introduced as a reference in the PP: *SMALL* (e.g. *cat*) vs. *LARGE* (e.g. *building*).[10]

Obviously, a study of *some* with these two variables results in a schematic token set with four variable level combinations, as in Table 16.

Table 16. Token sets (schematic + concrete) for OBJECT × REFPOINT

	REFPOINT: *SMALL*	REFPOINT: *LARGE*
OBJECT: *SMALL*	*SMALL* + *SMALL*: *some dogs next to a cat*	*SMALL* + *LARGE*: *some dogs next to a car*
OBJECT: *LARGE*	*LARGE* + *SMALL*: *some cars next to a cat*	*LARGE* + *LARGE*: *some cars next to a fence*

The (non-directional) hypotheses for this study are:

H_0: The average estimate of how many *some* denotes is independent of the sizes of the objects (OBJECT: *SMALL* vs. *LARGE*) and the sizes of the reference points (REFPOINT: *SMALL* vs. *LARGE*) in the utterances for which subjects provide estimates: $mean_{\text{SMALL+SMALL}} = mean_{\text{SMALL+LARGE}} = mean_{\text{LARGE+SMALL}} = mean_{\text{LARGE+LARGE}}$.

H_1: The average estimate of how many *some* denotes is dependent on the sizes of the objects (OBJECT: *SMALL* vs. *LARGE*) and/or the sizes of the reference points (REFPOINT: *SMALL* vs. *LARGE*) and/or some joint effect of the two: there is at least one ≠ in the above equation.

Let us now also assume you want to test these hypotheses with a questionnaire: subjects will be shown phrases such as those in Table 16 and

10 I will not discuss here how to decide what is 'small' and what is 'large'. In the study from which this example is taken, the sizes of the objects were determined on the basis of a pilot study prior to the real experiment.

then asked to provide estimates of how many elements a speaker of such a phrase would probably intend to convey – how many dogs were next to a cat etc. Since you have four variable level combinations, you need at least four concrete token sets (the rule in (5)), which are created according to the rule in (4). According to the rules in (6) and (7) this also means you need at least four subjects: you cannot have fewer because then some subject would see more than one stimulus from one concrete token set. You can then assign experimental stimuli to the subjects in a rotating fashion. The result of this is shown in the sheet <Phase 1> of the file <_input files/01-5_ExperimentalDesign.ods> (just like all files, this one too can be found on the companion website (see beginning of Chapter 2). The actual experimental stimuli are represented only schematically as a uniquely identifying combination of the number of the concrete token set and the variable levels of the two independent variables (in column E).

As you can easily see in the table on the right, the rotation ensures that every subject sees each variable level combination just once and each of these from a different concrete token set. However, we know you have to do more than that because in <Phase 1> every subject sees every variable level combination just once (which violates (8)) and every experimental item is seen by only one subject (which violates (9)). Therefore, you first re-use the experimental items in <Phase 1>, but put them in a different order so that the experimental items do not occur together with the very same experimental items (you can do that by rotating the subjects differently). One possible result of this is shown in the sheet <Phase 2>.

The setup in <Phase 2> does not yet conform to (8), though. For that, you have to do a little more. You must present more experimental items to, say, subject 1, but you cannot use the existing experimental items anymore without violating (6). Thus, you need four more concrete token sets, which are created and distributed across subjects as before. The result is shown in <Phase 3>. As you can see in the table on the right, every experimental item is now seen by two subjects (cf. the row totals), and in the columns you can see that each subjects sees each variable level combination in two different stimuli.

Now that every subjects receives eight experimental items, you must create enough distractors. In this example, let's use a ratio of experimental items to distractors of 1:2. Of course, 16 unique distractors are enough, which are presented to all subjects – there is no reason to create $8 \cdot 16 = 128$ distractors. Consider <Phase 4>, where the filler items have been added to the bottom of the table.

Now you must order the all stimuli – experimental items *and* distractors

– for every subject. To that end, you can add a column called "RND", which contains random numbers ranging between 0 and 1 (you can get those from R or by writing "=RAND()" (without double quotes, of course) into a cell in LibreOffice Calc and then double-clicking on the small black square on the bottom right corner you see when you click on that cell once, which will fill all cells below with random numbers.

As the next step, you will want to sort the whole spreadsheet (i) according to the column "SUBJ" and then (ii) according to the column "RAND". However, there is an important detail first: highlight that whole column, copy the contents into the clipboard, go to *Edit: Paste Special…*, and choose to paste back only the text and the numbers. This will make sure that the random numbers are not re-calculated after anything you do to the spreadsheet. Then sort as mentioned above so that all items of one subject are grouped together, and within each subject the order of items is random. This is required by (12) and represented in <Phase 5>.

When you look at <Phase 5>, you also see that the order of some elements must still be changed: red arrows in column H indicate problematic sequences of experimental items and blue arrives indicate potentially problematic sequences of identical schematic tokens. To take care of these cases, you can arbitrarily move things around. One possible result is shown in <Phase 6>, where the green arrows point to corrections. If we had used actual stimuli, you could now create a cover sheet with instructions for the subjects and a few examples (which in the case of, say, judgments would ideally cover the extremes of the possible judgments!), paste the experimental stimuli onto the following page(s), and hand out the questionnaires. Then, when you get the responses back, you enter them into <Phase 7> and proceed to analyze them statistically. For example, to evaluate this experiment, you would then have to compute a variety of means:

– the means for the two levels of OBJECT (i.e., $mean_{\text{OBJECT: } SMALL}$ and $mean_{\text{OBJECT: } LARGE}$);
– the means for the two levels of REFPOINT (i.e., $mean_{\text{REFPOINT: } SMALL}$ and $mean_{\text{REFPOINT: } LARGE}$);
– the four means for the interaction of OBJECT and REFPOINT.

We will discuss the method that is used to test these means for significant differences – a linear model – in Section 5.2.

Now you should do the exercises for Chapter 1 (which you can find on the website) …

Chapter 2
Fundamentals of R

When we say that a historian or a linguist is 'innumerate'
we mean that he cannot even begin to understand what
scientists and mathematicians are talking about
Oxford English Dictionary, 2nd ed., 1989, s.v. *numeracy*.
(cited from Keen 2010: 4)

1. Introduction and installation

In this chapter, you will learn about the basics of R that enable you to load, process, and store data as well as perform some simple data processing operations. Thus, this chapter prepares you for the applications in the following chapters. Let us begin with the first step: the installation of R.

1. The main R website is <http://www.r-project.org/>. From there you can go to the CRAN website at <http://cran.r-project.org/mirrors.html>. Click on the mirror Austria, then on the link(s) for your operating system;
2. for Windows you will then click on "base", and then on the link to the setup program to download the relevant setup program; for Mac OS X, you immediately get to a page with a link to a .pkg file; for Linux, you choose your distribution, maybe your distribution version, and then the relevant file(s) or, more conveniently, you may be able to install R and many frequently-used packages using a package manager such as Synaptic or Muon;
3. then, you run the installer;
4. start R by double-clicking on the icon on the desktop, the icon in the start menu, or the icon in the quick launch tool bar.

That's it. You can now start and use R. However, R has more to offer. Since R is an open-source software, there is a lively community of people who have written so-called packages for R. These packages are small additions to R that you can load into R to obtain commands (or functions, as we will later call them) that are not part of the default configuration.

5. In R, enter the following at the console `install.packages()¶` and then choose a mirror; I recommend always using *Austria*;
6. Choose all packages you think you will need; if you have a broadband connection, you could theoretically choose all of them, but that might be a bit of an overkill at this stage. I minimally recommend `amap`, `aod`, `car`, `cluster`, `effects`, `Hmisc`, `lattice`, `qcc`, `plotrix`, `rms`, `rpart`, and `vcd`. (You can also enter, say, `install.packages("car")¶` at the console to install said package and ideally do either with administrator/root rights; in Ubuntu, for example, start R with `sudo R¶`. On Linux systems, you will sometimes also need additional files such as gfortran, which you may need to install separately.)

Next, you should download the files with example files, all the code, exercises, and answer keys onto your hard drive. Ideally, you create one folder that will contain all the files from the book, such as <_sflwr> on your harddrive (for statistics for linguists with R). Then download all files from the companion website of this edition of the book (<http://tinyurl.com/StatForLingWithR>) and save/unzip them into:

– <_sflwr/_inputfiles>: this folder will contain all input files: text files with data for later statistical analysis, spreadsheets providing all files in a compact format, input files for exercises etc.; to unzip these files, you will need the password "hamste_R2";
– <_sflwr/_outputfiles>: this folder will contain output files from Chapters 2 and 5; to unzip these files, you will need the password "squi_R2rel";
– <_sflwr/_scripts>: this folder will contain all files with code from this book as well as the files with exercises and their answer keys; to unzip these files, you will need the password "otte_R2".

(By the way, I am using regular slashes here because you can use those in R, too, and more easily so than backslashes.) The companion website will also provide a file with errata. Lastly, I would recommend that you also get a text editor that has syntax highlighting for R or an IDE (integrated development environment). If you use a text editor, I recommend Notepad++ to Windows users and geany or the use of Notepad++ with Wine to Linux users. The probably best option, however, might be to go with RStudio (<http://www.rstudio.org/>), a truly excellent open source IDE for R, which offers easy editing of R code, sending code from the editor window

to the console with just using Ctrl+ENTER, plot histories, and many other things; you should definitely watch the screencast at RStudio's website.

After all this, you can view all scripts in <_scripts> with syntax-highlighting, which will make it easier for you to understand them. I strongly recommend to write all R scripts that are longer than, say, 2-3 lines in these editors / in the script window of the IDE and then paste them into R because the syntax high-lighting will help you avoid mistakes and you can more easily keep track of all the things you have entered into R.

R is not just a statistics program – it is also a programming language and environment which has at least some superficial similarity to Perl, Python, or Julia. The range of applications is breathtakingly large as R offers the functionality of spreadsheet software, statistics programs, a programming language, database functions etc. This introduction to statistics, however, is largely concerned with

- functions to generate and process simple data structures in R, and
- functions for probability distributions, statistical tests, and graphical evaluation.

We will therefore unfortunately not be able to deal with more complex data structures and many aspects of R as a programming language however interesting these may be. Also, I will not always use the simplest or most elegant way to perform a particular task but the way that is most useful from a pedagogical and methodological perspective (e.g., to highlight commonalities between different functions and approaches). Thus, this book is not really a general introduction to R, and I refer you to the recommendations for further study and the reference section for introductory books to R.

Now we have to address some typographical and other conventions. As already above, websites, folders, and files will be delimited by "<" and ">" as in, say, <_inputfiles/04-1-1-1_tense-aspect.csv>, where the numbering before the underscore refers to the section in which this file is used. Text you are supposed to enter into R is formatted like this `mean(c(1, 2, 3))¶`. This character "¶" instructs you to hit ENTER (I show these characters here because they can be important to show the exact structure of a line and because whitespace makes a big difference in character strings; the code files of course do not include those visibly unless you set your text editor to displaying them). Code will usually be given in grey blocks of several lines like this:

```
> a<-c(1, 2, 3)¶
> mean(a)¶
[1] 2
```

This also means for you: do not enter the two characters `>` . They are only provided for you to easily distinguish your input from R's output. You will also occasionally see lines that begin with "+". These plus signs, which you are not supposed to enter either, begin lines where R is still expecting further input before it begins to execute the function. For example, when you enter `2-¶`, then this is what your R interface will look like:

```
> 2-¶
+
```

R is waiting for you to complete the subtraction. When you enter the number you wish to subtract and press ENTER, then the function will be executed properly.

```
+ 3¶
[1] -1
```

Another example: if you wish to load the package `corpora` into R to access some of the functions that the computational linguists Marco Baroni and Stefan Evert contributed to the community, you can load this package by entering `library(corpora)¶`. (Note: this only works if you installed the package before as explained above.) However, if you forget the closing bracket, R will wait for you to complete the input:

```
> library(corpora¶
+ )¶
>
```

Unfortunately, R will not always be this forgiving. By the way, if you make a mistake in R, you often need to change only one thing in a line. Thus, rather than typing the whole line again, press the cursor-up key to get back to that line you wish to change or execute again; also, you need not move the cursor to the end of the line before pressing ENTER.

Corpus files or tables / data frames will be represented as in Figure 10, where "→" and "¶"denote tab stops and line breaks respectively. Menus, submenus, and commands in submenus in applications are given in italics in double quotes, and hierarchical levels within application menus are indicated with colons. So, if you open a document in, say, LibreOffice Writer,

you do that with what is given here as *File: Open* ...

```
PartOfSp  →  TokenFreq    →  TypeFreq   →  Class¶
ADJ       →  421          →  271        →  open¶
ADV       →  337          →  103        →  open¶
N         →  1411         →  735        →  open¶
CONJ      →  458          →  18         →  closed¶
PREP      →  455          →  37         →  closed¶
```

Figure 10. Representational format of corpus files and data frames

2. Functions and arguments

As you may remember from school, one often does not use numbers, but rather letters to represent variables that 'contain' numbers. In algebra class, for example, you had to find out from two equations such as the following which values *a* and *b* represent (here $a = {}^{23}/_{7}$ and $b = {}^{20}/_{7}$):

$a+2b = 9$ and
$3a-b = 7$

In R, you can solve such problems, too, but R is much more powerful, so variable names such as *a* and *b* can represent huge multidimensional elements or, as we will call them here, *data structures*. In this chapter, we will deal with the data structures that are most important for statistical analyses. Such data structures can either be entered into R at the console or, more commonly, read from files. I will present both means of data entry, but most of the examples below presuppose that the data are available in the form of a tab-delimited text file that has the structure discussed in the previous chapter and was created in a text editor or a spreadsheet software such as LibreOffice Calc. In the following sections, I will explain

- how to create data structures in R;
- how to load data structures into R and save them from R;
- how to edit data structures in R.

One of the most central things to understand about R is how you tell it to do something other than the simple calculations from above. A command in R virtually always consists of two elements: a *function* and, in parentheses, *arguments*. A function is an instruction to do something, and

the arguments to a function represent (i) what the instruction is to be applied to and (ii) how the instruction is to be applied to it. (Arguments can be null, in which case the function name is just followed by opening and closing parentheses.) Let us look at two simple arithmetic functions you know from school. If you want to compute the square root of 5 with R – without simply entering the instruction `5^0.5¶`, that is – you need to know the name of the function as well as how many and which arguments it takes. Well, the name of the function is `sqrt`, and it takes just one argument which R calls `x` by default, namely the figure of which you want the square root. Thus:

```
> sqrt(x=5)¶
[1] 2.236068
```

Note that R just outputs the result, but does not store it. If you want to store a result into a data structure, you must use the assignment operator `<-` (an arrow consisting of a less-than sign and a minus). The simplest way in the present example is to assign a name to the result of `sqrt(5)`. Note: R's handling of names, functions, and arguments is case-sensitive, and you can use letters, numbers, periods, and underscores in names as long as the name begins with a letter or a period (e.g., `my.result` or `my_result` or …):

```
> a<-sqrt(x=5)¶
```

R does not return anything, but the result of `sqrt(5)` has now been assigned to a data structure that is called a vector, which is called `a`. You can test whether the assignment was successful by looking at the content of `a`. One function to do that is `print`, and its minimally required argument is the data structure whose content you want to see, but most of the time, it is enough to simply enter the name of the relevant data structure:

```
> print(a)¶
[1] 2.236068
> a¶
[1] 2.236068
```

Three final comments before we discuss various data structures in more detail. First, R ignores everything in a line after a pound/number sign or hash, which you can use to put comments into your lines (to remind you what that line is doing). Second, the assignment operator can also be used to assign a new value to an existing data structure. For example,

```
> a<-sqrt(x=9) # assign the value of 'sqrt(9)' to a¶
> a # print a¶
[1] 3
> a<-a+2 # assign the value of 'a+2' to a¶
> a # print a¶
[1] 5
```

If you want to delete or clear a data structure, you can use the function `rm` (for *remove*). You can remove just a single data structure by using its name as an argument to `rm`, or you can remove all data structures at once.

```
> rm(a) # remove/clear a¶
> rm(list=ls(all=TRUE)) # clear memory of all data¶
```

Third, it will be very important later on to know that functions have default orders of their arguments and that many functions have default settings for their arguments. The former means that, if you provide arguments in their default order, you don't have to name them. That is, instead of `sqrt(x=9)¶` you could just write `sqrt(9)¶` because the (only) argument `x` is in its 'default position'. The latter means that if you use a function without specifying all required arguments, then R will use default settings, if those are provided by that function. Let us explore this on the basis of the very useful function `sample`. This function generates random or pseudo-random samples of elements and can take up to four arguments:

- `x`: a data structure – typically a vector – containing the elements from which you want to sample;
- `size`: a positive integer giving the size of the sample;
- the assignment `replace=FALSE` (if each element of the vector can only be sampled once, the default setting) or `replace=TRUE` (if the elements of the vector can be sampled multiple times, sampling with replacement);
- `prob`: a vector with the probabilities of each element to be sampled; the default setting is `NULL`, which means that all elements are equally likely to be sampled.

Let us look at a few examples, which will make successively more use of default orders and argument settings. First, you generate a vector with the numbers from 1 to 10 using the function `c` (for *concatenate*); the colon here generates a sequence of integers between the two numbers:

```
> some.data<-c(1:10)¶
```

If you want to sample 5 elements from this vector equiprobably and with replacement, you can enter the following:[11]

```
> sample(x=some.data, size=5, replace=TRUE, prob=NULL)¶
[1] 5 9 9 9 2
```

But if you list the arguments of a function in their standard order (as we do here), then you can leave out their names:

```
> sample(some.data, 5, TRUE, NULL)¶
[1] 3 8 4 1 7
```

Also, `prob=NULL` is the default, so you can leave that out, too:

```
> sample(some.data, 5, TRUE)¶
[1]  2  1  9  9 10
```

With this, you sample 5 elements equiprobably *without* replacement:

```
> sample(some.data, 5, FALSE)¶
[1]  1 10  6  3  8
```

But since `replace=FALSE` is the default, you can leave that out, too:

```
> sample(some.data, 5)¶
[1] 10  5  9  3  6
```

Sometimes, you can even leave out the `size` argument, namely when you just want all elements of the given vector in a random order:

```
> some.data¶
[1]  1  2  3  4  5  6  7  8  9 10
> sample(some.data)¶
[1]  2  4  3 10  9  8  1  6  5  7
```

And if you only want the numbers from 1 to 10 in a random order, you can even do away with the vector `some.data`:

```
> sample(10)¶
[1]  5 10  2  6  1  3  4  9  7  8
```

11. Your results will be different, after all this is *random* sampling.

In extreme cases, the property of default settings may result in function calls without any arguments. Consider the function `q` (for *quit*). This function shuts R down and usually requires three arguments:

- `save`: a character string indicating whether the R workspace should be saved or not or whether the user should be prompted to make that decision (the default);
- `status`: the (numerical) error status to be returned to the operating system, where relevant; the default is 0, indicating 'successful completion';
- `runLast`: a logical value (`TRUE` or `FALSE`), stating whether a function called `Last` should be executed before quitting R; the default is `TRUE`.

Thus, if you want to quit R with these settings, you just enter:

```
> q()¶
```

R will then ask you whether you wish to save the R workspace or not and, when you answered that question, executes the function `Last` (only if one is defined), shuts down R and sends "0" to your operating system.

As you can see, defaults can be a very useful way of minimizing typing effort. However, especially at the beginning, it is probably wise to try to strike a balance between minimizing typing on the one hand and maximizing code transparency on the other. While this may ultimately boil down to a matter of personal preference, I recommend using more explicit code at the beginning in order to be maximally aware of the options your R code uses; you can then shorten your code as you become more proficient.

Recommendation(s) for further study

the functions `?` or `help`, which provide the help file for a function (try `?sample¶` or `help(sample)¶`), and the functions `args` and `formals`, which provide the arguments a function needs, their default settings, and their default order (try `formals(sample)¶` or `args(sample)¶`)

3. Vectors

3.1. Generating vectors

The most basic data structure in R is a vector. Vectors are one-dimensional, sequentially ordered sequences of elements (such as numbers or character

strings (such as words)). While it may not be completely obvious why vectors are important here, we must deal with them in some detail since many other data structures in R can ultimately be understood in terms of vectors. As a matter of fact, we have already used vectors when we computed the square root of 5:

```
> sqrt(5)¶
[1] 2.236068
```

The "[1]" before the result indicates that the first (and, here, only) element printed as the output is element number 1, namely 2.236068. You can test this with R: first, you assign the result of `sqrt(5)` to a data structure.

```
> a<-sqrt(5)¶
```

The function `is.vector` tests whether its argument is a vector or not and returns the result of its test, here R's version of "yes":

```
> is.vector(a)¶
[1] TRUE
```

And the function `length` returns the number of elements of the data structure provided as its argument:

```
> length(a)¶
[1] 1
```

Of course, you can also create vectors that contain character strings – the only difference is that the character strings are put into double quotes:

```
> a.name<-"John"; a.name¶
[1] "John"
```

In this book, we only deal with logical vectors as well as vectors of numbers or character strings. Vectors usually only become interesting when they contain more than one element. You already know the function to create such vectors, `c`, and the arguments it takes are just the elements to be concatenated in the vector, separated by commas. For example:

```
> numbers<-c(1, 2, 3); numbers¶
[1] 1 2 3
```

or

```
> some.names<-c("al", "bill", "chris"); some.names¶
[1] "al"    "bill"  "chris"
```

Note that, since individual numbers or character strings are also vectors (just vectors of length 1), the function `c` can not only combine individual numbers or character strings but also vectors with 2+ elements:

```
> numbers1<-c(1, 2, 3); numbers2<-c(4, 5, 6) # generate two
   vectors¶
> numbers1.and.numbers2<-c(numbers1, numbers2) # combine
   vectors¶
> numbers1.and.numbers2¶
[1] 1 2 3 4 5 6
```

A similar function is `append`, which takes two or three arguments:

- `x`: a vector to which something should be appended;
- `values`: the vector to be appended;
- `after`: the position in the first argument where the elements of the second argument are to be appended; the default setting is at the end.

Thus, with `append`, the above example would look like this:

```
> numbers1.and.numbers2<-append(numbers1, numbers2)¶
> numbers1.and.numbers2¶
[1] 1 2 3 4 5 6
```

An example of how `append` is more typically used is the following, where an existing vector is modified:

```
> evenmore<-c(7, 8)¶
> numbers1.and.numbers2<-append(numbers1.and.numbers2,
   evenmore)¶
> numbers¶
[1] 1 2 3 4 5 6 7 8
```

It is important to note that – unlike arrays in Perl – vectors can only store elements of one data type. For example, a vector can contain numbers *or* character strings, but not really both: if you try to force character strings into a vector together with numbers, R will change the data type of one kind of element to homogenize the kinds of vector elements, and since you

can interpret numbers as characters but not vice versa, R changes the numbers into character strings and then concatenates them into a vector of character strings:

```
> mixture<-c("al", 2, "chris"); mixture¶
[1] "al"    "2"     "chris"
```

and

```
> numbers.num<-c(1, 2, 3); numbers.char<-c("four", "five",
   "six")¶
> nums.and.chars<-c(numbers.num, numbers.char)¶
> nums.and.chars¶
[1] "1"    "2"    "3"    "four" "five" "six"
```

The double quotes around 1, 2, and 3 indicate that these are now understood as character strings, which means that you cannot use them for calculations anymore (unless you change their data type back). We can identify the type of a vector (or the data types of other data structures) with `str` (for "structure") which takes as an argument the name of a data structure:

```
> str(numbers.num)¶
 num [1:3] 1 2 3¶
> str(nums.and.chars)¶
 chr [1:6] "1" "2" "3" "four" "five" "six"
```

The first vector consists of three numerical elements, namely 1, 2, and 3. The second vector consists of the six character strings (from *character*) that are printed.

As you will see later, it is often necessary to create quite long vectors in which (sequences of) elements are repeated. Instead of typing those into R manually, you can use two very useful functions, `rep` and `seq`. In a simple form, the function rep (for *repetition*) takes two arguments: the element(s) to be repeated, and the number of repetitions. To create, say, a vector `x` in which the number sequence from 1 to 3 is repeated four times, you enter:

```
> numbers<-c(1, 2, 3)¶
> x<-rep(numbers, 4)¶
```

or

```
> x<-rep(c(1, 2, 3), 4); x¶
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

To create a vector in which the numbers from 1 to 3 are individually repeated four times – not in sequence – then you use the argument `each`:

```
> x<-rep(c(1, 2, 3), each=4); x¶
[1] 1 1 1 1 2 2 2 2 3 3 3 3
```

(The same would be true of vectors of character strings.) With whole numbers, you can also often use the `:` as a range operator:

```
> x<-rep(c(1:3), 4)¶
```

The function `seq` (for *sequence*) is used a little differently. In one form, `seq` takes three arguments:

- `from`: the starting point of the sequence;
- `to`: the end point of the sequence;
- `by`: the increment of the sequence.

Thus, instead of entering `numbers<-c(1:3)¶`, you can also write:

```
> numbers<-seq(1, 3, 1)¶
```

Since 1 is the default increment, the following would suffice:

```
> numbers<-seq(1, 3)¶
```

In fact, you can even just write this:

```
> numbers<-seq(3)¶
```

If the numbers in the vector to be created do not increment by 1, you can set the increment to whatever value you need. The following lines generate a vector `x` in which the even numbers between 1 and 10 are repeated six times in sequence. Try it out (and look at `x`):

```
> numbers<-seq(2, 10, 2)¶
> x<-rep(numbers, 6)¶
```

or

```
> x<-rep(seq(2, 10, 2), 6)¶
```

Finally, instead of providing the increment, you can also let R figure out it for you, as when you know how long your sequence should be and just want equal increments everywhere. You can then use the argument `length.out`. The following generates a 7-element sequence from 1 to 10 with equal increments and assigns it to `numbers`:

```
> numbers<-seq(1, 10, length.out=7); numbers¶
[1] 1.0  2.5  4.0  5.5  7.0  8.5 10.0
```

With `c`, `append`, `rep`, and `seq`, even long and complex vectors can often be created fairly easily. Another useful feature is that you can not only name vectors, but also elements of vectors:

```
> numbers<-c(1, 2, 3); names(numbers)<-c("one", "two",
   "three")¶
> numbers¶
  one   two three
    1     2     3
```

Before we turn to loading and saving vectors, let me briefly mention an interactive way to enter vectors into R. If you assign to a data structure just `scan()¶` (for numbers) or `scan(what=character(0))¶` (for character strings), then you can enter the numbers or character strings separated by ENTER until you complete the data entry by pressing ENTER twice:

```
> x<-scan()¶
1: 1¶
2: 2¶
3: 3¶
4:¶¶
Read 3 items
> x¶
[1] 1 2 3
```

Recommendation(s) for further study
the functions `as.numeric` and `as.character` to change the type of vectors

3.2. Loading and saving vectors

Since data for statistical analysis will usually not be entered into R manually, we now turn to reading vectors from files. First a general remark: R can read data of different formats, but we only discuss data saved as text files,

i.e., files that often have the extension: <.txt> or <.csv>. Thus, if the data file has not been created with a text editor but a spreadsheet software such as LibreOffice Calc, then you must first export these data into a text file (with *File: Save As ...* and *Save as type: Text CSV (.csv)*).

A very powerful function to load vector data into R is the function `scan`, which we already used to enter data manually. This function can take many different arguments so you should list arguments with their names. The most important arguments of `scan` for our purposes together with their default settings are as follows:

- `file=""`: the path of the file you want to load as a character string, e.g. `"_inputfiles/02-3-2_vector1.txt"`, but most of the time it is probably easier to just use the function `file.choose()`, which will prompt you to choose the relevant file directly; note, the `file` argument can also be `"clipboard"`;
- `what=""`: the kind of input `scan` is supposed to read. The most important settings are `what=double(0)` (for numbers, the omissible default) and `what=character(0)` (for character strings);
- `sep=""`: the character that separates individual entries in the file. The default setting, `sep=""`, means that any whitespace character will separate entries, i.e. spaces, tabs (represented as `"\t"`), and newlines (represented as `"\n"`). Thus, if you want to read in a text file into a vector such that each line is one element of the vector, you write `sep="\n"`;
- `dec=""`: the decimal point character; `dec="."` is the default; if you want to use a comma instead of the default period, just enter that here as `dec=","`.

To read the file <_inputfiles/02-3-2_vector1.txt>, which contains what is shown in Figure 11, into a vector `x`, you could enter this.

```
1¶
2¶
3¶
4¶
5¶
```

Figure 11. An example file

```
> x<-scan(file=file.choose(), sep="\n")¶
Read 5 items
```

Then you can print out the contents of `x`:

```
> x¶
[1] 1 2 3 4 5
```

Reading in a file with character strings (like the one in Figure 12) is just as easy; here you just have to tell R that you are reading in a file of character strings and that the character strings are separated by spaces:

```
alpha·bravo·charly·delta·echo¶
```

Figure 12. Another example file

```
> x<-scan(file.choose(), what=character(0), sep=" ")¶
```

You get:

```
> x¶
[1] "alpha"  "bravo"  "charly" "delta"  "echo"
```

Now, how do you save vectors into files. The required function – basically the reverse of `scan` – is `cat` and it takes very similar arguments:

- the vector(s) to be saved;
- `file=""`: the path to the file into which the vector is to be saved or again just `file.choose()`);
- `sep=""`: the character that separates the elements of the vector from each other: `sep=""` or `sep=" "` for spaces (the default), `sep="\t"` for tabs, `sep="\n"` for newlines;
- `append=TRUE` or `append=FALSE` (the default): if the output file already exists and you set `append=TRUE`, then the output will be appended to the output file, otherwise the output will overwrite the existing file.

Thus, to append two names to the vector `x` and then save it under some other name, you can enter the following:

```
> x<-append(x, c("foxtrot", "golf"))¶
> cat(x, file=file.choose())¶
```

Recommendation(s) for further study
the function `write`, `save`, and `dput` to save vectors (or other structures)

3.3. Editing vectors

Now that you can generate, load, and save vectors, we must deal with how you can edit them. The functions we will be concerned with allow you to access particular parts of vectors to output them, to use them in other functions, or to change them. First, a few functions to edit numerical vectors. One such function is `round`. Its first argument is the vector with numbers to be rounded, its second the desired number of decimal places. (Note, R rounds according to an IEEE standard: 3.55 does not become 3.6, but 3.5.)

```
> a<-seq(3.4, 3.6, 0.05); a¶
[1] 3.40 3.45 3.50 3.55 3.60
> round(a, 1)¶
[1] 3.4 3.4 3.5 3.5 3.6
```

The function `floor` returns the largest integers not greater than the corresponding elements of the vector provided, `ceiling` returns the smallest integers not less than the corresponding elements of the vector provided, and `trunc` simply truncates the elements toward 0:

```
> floor(c(-1.8, 1.8))¶
[1] -2  1
> ceiling(c(-1.8, 1.8))¶
[1] -1  2
> trunc(c(-1.8, 1.8))¶
[1] -1  1
```

The most important way to access parts of a vector (or other data structures) in R involves *subsetting* with square brackets. In the simplest form, this is how you access an individual vector element (here, the third):

```
> x<-c("a", "b", "c", "d", "e")¶
> x[3]¶
[1] "c"
```

Since you already know how flexible R is with vectors, the following uses of square brackets should not come as big surprises:

```
> y<-3; x[y]¶
[1] "c"
> z<-c(1, 3); x[z]¶
[1] "a" "c"
> z<-c(1:3); x[z]¶
[1] "a" "b" "c"
```

With negative numbers, you can leave out elements:

```
> x[-2]¶
[1] "a" "c" "d" "e"
```

However, there are many more powerful ways to access parts of vectors. For example, you can let R determine which elements of a vector fulfill a certain condition. One way is to present R with a *logical expression*:

```
> x=="d"¶
[1] FALSE FALSE FALSE  TRUE FALSE
```

This means, R checks for each element of `x` whether it is "d" or not and returns its findings. The only thing requiring a little attention here is that the logical expression uses two equal signs, which distinguishes logical expressions from assignments such as `file=""`. Other logical operators are:

`&`	and	`|`	or
`>`	greater than	`<`	less than
`>=`	greater than or equal to	`<=`	less than or equal to
`!`	not	`!=`	not equal to

Here are some examples:

```
> x<-c(10:1)¶
> x¶
[1] 10  9  8  7  6  5  4  3  2  1
> x==4¶
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
   FALSE
> x<=7¶
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
   TRUE
> x!=8¶
[1]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
   TRUE
> (x>8 | x<3)¶
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
   TRUE
```

Since `TRUE` and `FALSE` in R correspond to 1 and 0, you can easily determine how often a particular logical expression is true in a vector:

```
> sum(x==4)¶
[1] 1
```

```
> sum(x>8 | x<3)¶
[1] 4
```

The very useful function `table` counts how often vector elements (or combinations of vector elements) occur. For example, with `table` we can immediately determine how many elements of `x` are greater than 8 or less than 3. (Note: `table` ignores missing data – if you want to count those, too, you must write `table(…, exclude=NULL)`.)

```
> table(x>8 | x<3)¶
FALSE  TRUE
    6     4
```

It is, however, obvious that the above examples are not particularly elegant ways to identify the position(s) of elements. However many elements of `x` fulfill a logical condition, you always get 10 logical values (one for each element of `x`) and must locate the `TRUE`s by hand – what do you do when a vector contains 10,000 elements? Another function can do that for you, though. This function is `which`, and it takes a logical expression of the type discussed above:

```
> which(x==4) # which elements of x are 4?¶
[1] 7
```

As you can see, this function looks nearly like English: you ask R "which element of `x` is 4?", and you get the response 'the seventh'. The following examples are similar to the ones above but now use `which`:

```
> which(x<=7) which elements of x are <= 7?¶
[1]  4  5  6  7  8  9 10
> which(x!=8) # which elements of x are not 8?¶
[1]  1  2  4  5  6  7  8  9 10
> which(x>8 | x<3) which elements of x are >8 or <3?¶
[1]  1  2  9 10
```

It should go without saying that you can assign such results to data structures, i.e. vectors:

```
> y<-which(x>8 | x<3); y¶
[1]  1  2  9 10
```

Note: do not confuse the *position of an element* in a vector with the *element* of the vector. The function `which(x==4)¶` does not return the element

4, but the position of the element 4 in x, which is 7; and the same is true for the other examples. You can probably guess how you can now get the elements themselves and not just their positions. You only need to remember that R uses vectors. The data structure you just called y is also a vector:

```
> is.vector(y)¶
[1] TRUE
```

Above, you saw that you can use vectors in square brackets to access parts of a vector. Thus, when you have a vector x and do not just want to know where to find numbers which are larger than 8 or smaller than 3, but also which numbers these are, you first use which and then square brackets, or you immediately combine these two steps:

```
> y<-which(x>8 | x<3)¶
> x[y]¶
[1] 10  9  2  1
> x[which(x>8 | x<3)]¶
[1] 10  9  2  1
```

Or you use this, which uses the fact that, when you subset with a logical vector of TRUEs and FALSEs, R returns the elements subset by TRUEs:

```
> x[x>8 | x<3]¶
[1] 10  9  2  1
```

You use a similar approach to see how often a logical expression is true:

```
> length(which(x>8 | x<3))¶
[1] 4
```

Sometimes you may want to test for several elements at once (e.g., the numbers 1, 6, and 11), which which can't do, but you can use the very useful operator %in%:

```
> c(1, 6, 11) %in% x¶
[1]  TRUE  TRUE FALSE
```

The output of %in% is a logical vector which says for each element of the vector before %in% whether it occurs in the vector after %in%. If you also would like to know the exact position of the first (!) occurrence of each of the elements of the first vector in the second, you can use match:

```
> match(c(1, 6, 11), x)¶
[1] 10  5 NA
```

That is to say, the first element of the first vector – the 1 – occurs the first (and only) time at the tenth position of `x`; the second element of the first vector – the 6 – occurs the first (and only) time at the fifth position of `x`; the last element of the first vector – the 11 – does not occur in `x`.

I hope it becomes obvious that the fact that much of what R does involves vectors is a big strength of R. Since nearly everything we have done so far is based on vectors (often of length 1), you can use functions flexibly and even embed them into each other freely. For example, now that you have seen how to access parts of vectors, you can also change those. Maybe you would like to change the values of `x` that are greater than 8 into 12:

```
> x # show x again¶
 [1] 10  9  8  7  6  5  4  3  2  1
> y<-which(x>8)¶
> x[y]<-12¶
> x¶
[1] 12 12  8  7  6  5  4  3  2  1
```

As you can see, since you want to replace more than one element in `x` but provide only one replacement (12), R recycles the replacement as often as needed (cf. below for more on that feature). This is a shorter way to do the same thing:

```
> x<-10:1¶
> x[which(x>8)]<-12¶
> x¶
 [1] 12 12  8  7  6  5  4  3  2  1
```

And this one is even shorter:

```
> x<-10:1¶
> x[x>8]<-12¶
> x¶
 [1] 12 12  8  7  6  5  4  3  2  1
```

R also offers several set-theoretical functions – `setdiff`, `intersect`, and `union` – which take two vectors as arguments. The function `setdiff` returns the elements of the first vector that are not in the second vector. The function `intersect` returns the elements of the first vector that are also in the second vector. And the function `union` returns all elements that occur in at least one of the two vectors.

```
> x<-c(10:1); y<-c(2, 5, 9, 12)¶
> setdiff(x, y)¶
[1] 10  8  7  6  4  3  1
> setdiff(y, x)¶
[1] 12
> intersect(x, y)¶
[1] 9 5 2
> intersect(y, x)¶
[1] 2 5 9
> union(x, y)¶
[1] 10  9  8  7  6  5  4  3  2  1 12
> union(y, x)¶
[1]  2  5  9 12 10  8  7  6  4  3  1
```

Another useful function is `unique`, which can be explained particularly easily to linguists: `unique` goes through all the elements of a vector (tokens) and returns all elements that occur at least once (types).

```
> x<-c(1, 2, 3, 2, 3, 4, 3, 4, 5)¶
> unique(x)
[1] 1 2 3 4 5
```

In R you can also very easily apply a mathematical function or operation to many or all elements of a numerical vector. Mathematical operations that are applied to a vector are applied to all elements of the vector:

```
> x<-c(10:1)¶
> x¶
[1] 10  9  8  7  6  5  4  3  2  1
> y<-x+2¶
> y¶
[1] 12 11 10  9  8  7  6  5  4  3
```

If you add two vectors (or multiply them with each other, or …), three different things can happen. First, if the vectors are equally long, the operation is applied to all pairs of corresponding vector elements:

```
> x<-c(2, 3, 4); y<-c(5, 6, 7)¶
> x*y¶
[1] 10 18 28
```

Second, the vectors are not equally long, but the length of the longer vector can be divided by the length of the shorter vector without a remainder. Then, the shorter vector will again be recycled as often as is needed to perform the operation in a pairwise fashion; as you saw above, often the length of the shorter vector is 1.

```
> x<-c(2, 3, 4, 5, 6, 7); y<-c(8, 9)¶
> x*y¶
[1] 16 27 32 45 48 63
```

Third, the vectors are not equally long and the length of the longer vector is not a multiple of the length of the shorter vector. In such cases, R will recycle the shorter vector as necessary, but will also issue a warning:

```
> x<-c(2, 3, 4, 5, 6); y<-c(8, 9)¶
> x*y¶
[1] 16 27 32 45 48
Warning message:
In x * y : longer object length is not a multiple of shorter
   object length
```

Finally, two functions to change the ordering of elements of vectors. The first of these functions is called `sort`, and its most important argument is of course the vector whose elements are to be sorted; another important argument defines the sorting style: `decreasing=FALSE` (the default) or `decreasing=TRUE`.

```
> x<-c(1, 3, 5, 7, 9, 2, 4, 6, 8, 10)¶
> y<-sort(x)¶
> z<-sort(x, decreasing=TRUE)¶
> y; z¶
[1]  1  2  3  4  5  6  7  8  9 10
[1] 10  9  8  7  6  5  4  3  2  1
```

The second function is `order`. It takes one or more vectors as arguments as well as the argument `decreasing=…` – but it returns something that may not be immediately obvious. Can you see what `order` does?

```
> z<-c("a", "c", "e", "d", "b")¶
> order(z, decreasing=FALSE)¶
[1] 1 5 2 4 3
```

THINK
BREAK

The output of `order` when applied to a vector `z` is a vector which tells you in which order to put the elements of `z` to sort them as specified. Let us clarify this rather opaque characterization: If you want to sort the values of

`z` in increasing order, you first have to take `z`'s first value. Thus, the first value of `order(z, decreasing=FALSE)¶` is 1. The next value you have to take is the fifth value of `z`. The next value you take is the second value of `z`, etc. (If you provide `order` with more than one vector, additional vectors are used to break ties.) As we will see below, this function will turn out to be useful when applied to data frames.

Recommendations for further study

- the functions `any` and `all` to test whether any or all elements of a vector fulfill a particular condition
- the function `abs` to obtain the absolute values of a numerical vector
- the functions `min` and `max` to obtain the minimum and the maximum values of numeric vectors respectively

4. Factors

At a superficial glance at least, factors are similar to vectors of character strings. Apart from the few brief remarks in this section, they will mainly be useful when we read in data frames and want R to recognize that some of their columns are nominal or categorical variables.

4.1. Generating factors

As I just mentioned, factors are mainly used to code nominal or categorical variables, i.e. in situations where a variable has two or more (but usually not very many) qualitatively different levels. The simplest way to create a factor is to generate a vector and then change it into a factor using the function `factor`. That function usually takes one or two arguments. The first is mostly the vector you wish to change into a factor. The second argument is `levels=…` and will be discussed in more detail in Section 2.4.3 below.

```
> rm(list=ls(all=TRUE))¶
> x<-c(rep("male", 5), rep("female", 5))¶
> y<-factor(x); y¶
 [1] male   male   male   male   male   female female female
   female female
Levels: female male
> is.factor(y)
 [1] TRUE
```

When you output a factor, you can see one difference between factors and vectors because the output includes a by default alphabetically sorted list of all levels of that factor.

One other very useful way in which one sometimes generates factors is based on the function `cut`. In its simplest implementation, it takes a numeric vector as its first argument (`x`) and a number of intervals as its second (`breaks`), and then it divides `x` into `breaks` intervals:

```
> cut(1:9, 3)¶
[1] (0.992,3.66] (0.992,3.66] (0.992,3.66] (3.66,6.34]
   (3.66,6.34]  (3.66,6.34]  (6.34,9.01]  (6.34,9.01]
   (6.34,9.01]
Levels: (0.992,3.66] (3.66,6.34] (6.34,9.01]
```

As you can see, the vector with the numbers from 1 to 9 has now been recoded as a factor with three levels that provide the intervals R used for cutting up the numeric vector.

- $0.992 <$ interval/level 1 ≤ 3.66;
- $3.66 <$ interval/level 1 ≤ 6.34;
- $6.34 <$ interval/level 1 ≤ 9.01.

This function has another way of using `breaks` and some other useful arguments so you should explore those in more detail: `?cut`.

4.2. Loading and saving factors

We do not really need to discuss how you load factors – you do it in the same way as you load vectors, and then you convert the loaded vector into a factor as illustrated above. Saving a factor, however, is a little different. Imagine you have the following factor `a`.

```
> a<-factor(c("alpha", "charly", "bravo")); a¶
[1] alpha  charly bravo
Levels: alpha bravo charly
```

If you now try to save this factor into a file as you would with a vector,

```
> cat(a, sep="\n", file=file.choose())¶
```

your output file will look like Figure 13.

```
1¶
3¶
2¶
```

Figure 13. Another example file

This is because R represents factors internally in the form of numbers (which represent the factor levels), and therefore R also only outputs these numbers into a file. Since you want the words, however, you simply force R to treat the factor as a vector, which will produce the desired result.

```
> cat(as.vector(a), sep="\n", file=file.choose())¶
```

4.3. Editing factors

Editing factors is similar to editing vectors, but a bit more cumbersome when you want to introduce new levels. Let's create a factor `x`:

```
> x<-factor(rep(c("long", "intmed", "short"), 1:3)); x¶
[1] long   intmed intmed short  short  short
Levels: intmed long short
```

Note how the alphabetical ordering of the levels is not particularly useful since it does not coincide with an ascending or descending order of the meaning of the levels. The easiest thing you may have to do is change the first level from the alphabetically first level to another one (which will be important in Chapters 4 and 5). For example, you may want to make `short` the first level. For that, you can use the function `relevel`, which requires the factor to be changed and the new reference level:

```
> x<-relevel(x, "short"); x¶
[1] long   intmed intmed short  short  short
Levels: short intmed long
```

As you can see, the factor content per se has not changed, only the order in which the levels are listed and now we have a nice ordering of the levels.

If you want to change the order of levels more substantively – for instance reversing their order – you can use the function `factor` again and assign the levels in the desired way. Again, the content of the factor is the same, but the order of the levels is now reversed.

```
> x<-factor(x, levels=levels(x)[3:1]); x¶
[1] long   intmed intmed short  short  short
Levels: long intmed short
```

Now, what about changing the content of factors? You may want to just chang the name of a level in that factor to something else. You can do that by just setting a new level, e.g., changing `intmed` to `intermed`:

```
> levels(x)[2]<-"intermed"; x¶
[1] long     intermed intermed short    short    short
Levels: long intermed short
```

Then, you may wish to change a particular element to some other level that is already attested in the factor. In that case, you can treat factors as you would vectors:

```
> x[3]<-"short"; x¶
[1] long     intermed short    short    short    short
Levels: long intermed short
```

A difficulty arises when you want to assign a brand new level:

```
> x[6]<-"supershort"¶
Warning message:
In `[<-.factor`(`*tmp*`, 6, value = "supershort") :
  invalid factor level, NAs generated
> x¶
[1] long     intermed short    short    short    <NA>
Levels: long intermed short
```

Thus, if you want to assign a new level, you must proceed differently: Let's re-create `x` and then first define the new (fourth) level with `levels`:

```
x<-factor(rep(c("long", "intermed", "short"), c(1, 1, 4)),
   levels=c("long", "intermed", "short"))¶
> x<-factor(x, levels=c(levels(x), "supershort")); x¶
[1] long     intermed short    short    short    short
Levels: long intermed short supershort
```

Note: the factor content has not changed yet, you only have one more level than before. This also illustrates a factor can have levels that are not attested in its content. However, now that `x` has all the levels you need, you can proceed as above and assign the new value as you would with a vector:

```
> x[6]<-"supershort"; x¶
[1] long       intermed   short      short      short
   supershort
Levels: long intermed short supershort
```

Now, let's just assume you changed your mind and changed the sixth data point back to `short`:

```
> x[6]<-"short"; x¶
 [1] long     intermed short    short    short    short
Levels: long intermed short supershort
```

Now it is of course not nice to have this level `supershort` listed in the levels if it is not really attested especially because later we will use functions that would return output for these levels even if they are unattested. Thus, let us get rid of this level. Thankfully, that is easy: you can just apply the function `factor` again, which will then drop unused levels:

```
> x<-factor(x); x # also see ?droplevels¶
[1] long     intermed short    short    short    short
Levels: long intermed short
```

Sometimes one wants to conflate factor levels, e.g. to test whether all levels of a factor that corpus data were annotated for are actually required. Let's assume, you decide that you really only want to distinguish 'short' from 'not short'. This is how you change the levels and the factor accordingly, essentially by overwriting the first two levels with the new level.

```
> levels(x)<-c("not_short", "not_short", "short"); x¶
[1] not_short not_short short     short     short     short
Levels: not_short short
```

Finally, as I mentioned above, R stores factors as numbers and there are situations (esp. in the context of plotting, see Ch. 5) where it is useful to have access to these numbers. The function `as.numeric` provides these:

```
> as.numeric(x)¶
[1] 1 1 2 2 2 2
```

Recommendation(s) for further study

– the function `is.factor` to test whether a data structure is a factor
– the functions `gl` and `reorder` to create factors and reorder levels

5. Data frames

The data structure that is most relevant to nearly all statistical methods in this book is the data frame. The data frame, basically what we would colloquially call a table, is actually only a specific type of another data structure, a list, but since data frames are the single most frequent input format for statistical analyses (within R, but also for other statistical programs and of course spreadsheet software), we will concentrate only on data frames per se and disregard lists for now.

5.1. Generating data frames

Given the centrality of vectors in R, you can generate data frames easily from vectors (and factors). Imagine you collected three different kinds of information for five parts of speech and wanted to generate the data frame in Figure 14:

- the variable TOKENFREQUENCY, i.e. the frequency of words of a particular part of speech in a corpus X;
- the variable TYPEFREQUENCY, i.e. the number of different words of a particular part of speech in the corpus X;
- the variable CLASS, which represents whether the part of speech is from the group of open-class words or closed-class words.

```
POS     →     TOKENFREQ     →     TYPEFREQ     →     CLASS¶
adj     →     421           →     271          →     open¶
adv     →     337           →     103          →     open¶
n       →     1411          →     735          →     open¶
conj    →     458           →     18           →     closed¶
prep    →     455           →     37           →     closed¶
```

Figure 14. An example data frame

Step 1: you generate four vectors, one for each column:

```
> rm(list=ls(all=TRUE))¶
> POS<-c("adj", "adv", "n", "conj", "prep")¶
> TOKENFREQ<-c(421, 337, 1411, 458, 455)¶
> TYPEFREQ<-c(271, 103, 735, 18, 37)¶
> CLASS<-c("open", "open", "open", "closed", "closed")¶
```

Step 2: The first row in the desired table does not contain data points but the header with the column names. You must now decide whether the first column contains data points or also 'just' the names of the rows. In the first case, you can just create your data frame with the function `data.frame`, which takes as arguments the relevant vectors; the order of vectors determines the order of columns. Now you can look at the data frame.

```
> x<-data.frame(POS, TOKENFREQ, TYPEFREQ, CLASS)¶
> x¶
   POS TOKENFREQ TYPEFREQ  CLASS
1  adj       421      271   open
2  adv       337      103   open
3    n      1411      735   open
4 conj       458       18 closed
5 prep       455       37 closed
> str(x)¶
'data.frame': 5 obs. of  4 variables:
 $ POS      : Factor w/ 5 levels "adj","adv",..: 1 2 4 3 5
 $ TOKENFREQ: num  421 337 1411 458 455
 $ TYPEFREQ : num  271 103 735 18 37
 $ CLASS    : Factor w/ 2 levels "closed","open": 2 2 2 1 1
```

Within the data frame, R has changed the vectors of character strings into factors and represents them with numbers internally (e.g., `closed` is 1 and `open` is 2). It is very important in this connection that R only changes variables into factors when they contain character strings (and not just numbers). If you have a data frame in which nominal or categorical variables are coded with numbers, then R will neither know nor guess that these are factors and will treat the variables as numeric and thus as interval/ratio variables in statistical analyses. Thus, you should either use meaningful character strings as factor levels in the first place (as recommended in Chapter 1 anyway) or must characterize the relevant variable(s) as factors at the point of time you create the data frame: `factor(vectorname)`. Also, you did not define row names, so R automatically numbers the rows. If you want to use the parts of speech as row names, you need to say so explicitly:

```
> x<-data.frame(TOKENFREQ, TYPEFREQ, CLASS, row.names=POS)
> x¶
     TOKENFREQ TYPEFREQ  CLASS
adj        421      271   open
adv        337      103   open
n         1411      735   open
conj       458       18 closed
prep       455       37 closed
> str(x)¶
'data.frame': 5 obs. of  3 variables:
 $ TOKENFREQ: num  421 337 1411 458 455
```

```
 $ TYPEFREQ : num  271 103 735 18 37
 $ CLASS    : Factor w/ 2 levels "closed","open": 2 2 2 1 1
```

As you can see, there are now only three variables left because `POS` now functions as row names. Note that this is only possible when the column with the row names contains no element twice.

A second way of creating data frames that is much less flexible, but extremely important for Chapter 5 involves the function `expand.grid`. In its simplest use, the function takes several vectors or factors as arguments and returns a data frame the rows of which contain all possible combinations of vector elements and factor levels. Sounds complicated but is very easy to understand from this example and we will use this many times:

```
> expand.grid(COLUMN1=c("a", "b"), COLUMN2=1:3)¶
  COLUMN1 COLUMN2
1       a       1
2       b       1
3       a       2
4       b       2
5       a       3
6       b       3
```

5.2. Loading and saving data frames

While you can generate data frames as shown above, this is certainly not the usual way in which data frames are entered into R. Typically, you will read in files that were created with a spreadsheet software. If you create a table in, say LibreOffice Calc and want to work on it within R, then you should first save it as a comma-separated text file. There are two ways to do this. Either you copy the whole file into the clipboard, paste it into a text editor (e.g., geany or Notepad++), and then save it as a tab-delimited text file, or you save it directly out of the spreadsheet software as a CSV file (as mentioned above with *File: Save As ...* and *Save as type: Text CSV (.csv)*; then you choose tabs as field delimiter and no text delimiter, and don't forget to provide the file extension. To load this file into R, you use the function `read.table` and some of its arguments:

- `file="…"`: the path to the text file with the table (on Windows PCs you can use `choose.files()` here, too; if the file is still in the clipboard, you can also write `file="clipboard"`;
- `header=TRUE`: an indicator of whether the first row of the file contains

column headers (which it should always have) or `header=FALSE` (the default);
- `sep=""`: between the double quotes you put the single character that delimits columns; the default `sep=""` means space or tab, but usually you should set `sep="\t"` so that you can use spaces in cells of the table;
- `dec="."` or `dec=","`: the decimal separator;
- `row.names=…`, where … is the number of the column containing the row names;
- `quote=…`: the default is that quotes are marked with single or double quotes, but you should nearly always set `quote=""`;
- `comment.char=…`: the default is that comments are separated by "#", but we will always set `comment.char=""`.

Thus, if you want to read in the above table from the file <_inputfiles/02-5-2_dataframe1.csv> – once without row names and once with row names – then this is what you could type:

```
> a1<-read.table(file.choose(), header=TRUE, sep="\t",
   quote="", comment.char="") # R numbers rows¶
```

or

```
> a2<-read.table(file.choose(), header=TRUE, sep="\t",
   quote="", comment.char="", row.names=1) # row names¶
```

By entering `a1¶` or `str(a1)¶` (same with `a2`), you can check whether the data frames have been loaded correctly.

While the above is the most explicit and most general way to load all sorts of different data frames, when you have set up your data as recommended above, you can often use a shorter version with `read.delim`:, which has `header=TRUE` and `sep="\t"` as defaults and should, therefore, work most of the time:

```
> a3<-read.delim(file.choose())¶
```

If you want to save a data frame from R, then you can use `write.table`. Its most important arguments are:

- `x`: the data frame you want to save;
- `file`: the path to the file into which you wish to save the data frame;

typically, using `file.choose()` is easiest;
- `append=FALSE` (the default) or `append=TRUE`: the former generates or overwrites the defined file, the latter appends the data frame to that file;
- `quote=TRUE` (the default) or `quote=FALSE`: the former prints factor levels with double quotes; the latter prints them without quotes;
- `sep=""`: between the double quotes you put the single character that delimits columns; the default `" "` means a space, what you should use is `"\t"`, i.e. tabs;
- `eol="\n"`: between the double quotes you put the single character that separates lines from each other (eol for *end of line*); the default `"\n"` means newline;
- `dec="."` (the default): the decimal separator;
- `row.names=TRUE` (the default) or `row.names=FALSE`: whether you want row names or not;
- `col.names=TRUE` (the default) or `col.names=FALSE`: whether you want column names or not.

Given these default settings and under the assumption that your operating system uses an English locale, you would save data frames as follows:

```
> write.table(a1, file.choose(), quote=FALSE, sep="\t",
   col.names=NA)¶
```

5.3. Editing data frames

In this section, we will discuss how you can access parts of data frames and then how you can edit and change data frames.

Further below, we will discuss many examples in which you have to access individual columns or variables of data frames. You can do this in several ways. The first of these you may have already guessed from looking at how a data frame is shown in R. If you load a data frame with column names and use `str` to look at the structure of the data frame, then you see that the column names are preceded by a "$". You can use this syntax to access columns of data frames, as in this example using the file <_inputfiles/02-5-3_dataframe.csv>.

```
> rm(list=ls(all=TRUE))¶
> a<-read.delim(file.choose())¶
> a¶
   POS TOKENFREQ TYPEFREQ  CLASS
```

```
1  adj       421      271   open
2  adv       337      103   open
3    n      1411      735   open
4 conj       458       18 closed
5 prep       455       37 closed
> a$TOKENFREQ¶
[1]  421  337 1411  458  455
> a$CLASS¶
[1] open   open   open   closed closed
Levels: closed open
```

You can now use these just like any other vector or factor. For example, the following line computes token/type ratios of the parts of speech:

```
> ratio<-a$TOKENFREQ/a$TYPEFREQ; ratio¶
[1]  1.553506  3.271845  1.919728 25.444444 12.297297
```

You can also use indices in square brackets for subsetting. Vectors and factors as discussed above are one-dimensional structures, but R allows you to specify arbitrarily complex data structures. With two-dimensional data structures, you can also use square brackets, but now you must of course provide values for both dimensions to identify one or several data points – just like in a two-dimensional coordinate system. This is very simple and the only thing you need to memorize is the order of the values – rows, then columns – and that the two values are separated by a comma. Here are some examples:

```
> a[2,3]¶
[1] 103
> a[2,]¶
  POS TOKENFREQ TYPEFREQ CLASS
2 adv       337      103  open
> a[,3]¶
[1] 271 103 735  18  37
> a[2:3,4]¶
[1] open open
Levels: closed open
> a[2:3,3:4]¶
   TYPEFREQ CLASS
2       103  open
3       735  open
```

Note that row and columns names are not counted. Also note that all functions applied to vectors above can be used with what you extract out of a column of a data frame:

```
> which(a[,2]>450)¶
```

```
[1] 3 4 5
> a[,3][which(a[,3]>100)]¶
[1] 271 103 735
> a[,3][ a[,3]>100]¶
[1] 271 103 735
```

The most practical way to access individual columns, however, involves the function `attach` (and gets undone with `detach`). I will not get into the ideological debate about whether one should use `attach` or rather `with`, etc. – if you are interested in that, go to the R-Help list or read `?with`… You get no output, but you can now access any column with its name:

```
> attach(a)¶
> Class¶
[1] open   open   open   closed closed
Levels: closed open
```

Note two things. First, if you attach a data frame that has one or more names that have already been defined as data structures or as columns of previously attached data frames, you will receive a warning; in such cases, make sure you are really dealing with the data structures or columns you want and consider using `detach` to un-attach the earlier data frame. Second, when you use `attach` you are strictly speaking using 'copies' of these variables. You can change those, but these changes do not affect the data frame they come from.

```
> CLASS[4]<-NA; CLASS¶
[1] open   open   open   <NA>   closed
Levels: closed open
> a¶
   POS TOKENFREQ TYPEFREQ  CLASS
1  adj       421      271   open
2  adv       337      103   open
3    n      1411      735   open
4 conj       458       18 closed
5 prep       455       37 closed
```

Let's change `CLASS` back to its original state:

```
> CLASS[4]<-"closed"¶
```

If you want to change the data frame `a`, then you must make your changes in `a` directly, e.g. with `a$CLASS[4]<-NA¶` or `a$TOKENFREQ[2]<-338¶`. Given what you have seen in Section 2.4.3, however, this is only easy with vector or with factors where you do not add a new level – if you

want to add a new factor level, you must define that level first.

Sometimes you will need to investigate only a part of a data frame – maybe a set of rows, or a set of columns, or a matrix within a data frame. Also, a data frame may be so huge that you only want to keep one part of it in memory. As usual, there are several ways to achieve that. One uses indices in square brackets with logical conditions or `which`. Either you have already used `attach` and can use the column names directly or not:

```
> b<-a[CLASS=="open",]; b¶
  POS TOKENFREQ TYPEFREQ CLASS
1 adj       421      271  open
2 adv       337      103  open
3   n      1411      735  open

> b<-a[a[,4]=="open",]; b¶
  POS TOKENFREQ TYPEFREQ CLASS
1 adj       421      271  open
2 adv       337      103  open
3   n      1411      735  open
```

(Of course you can also write `b<-a[a$Class=="open",]¶`.) That is, you determine all elements of the column called `CLASS` / the fourth column that are `open`, and then you use that information to access the desired rows and all columns (hence the comma before the closing square bracket). There is a more elegant way to do this, though, the function `subset`. This function takes two arguments: the data structure of which you want a subset and the logical condition(s) describing which subset you want. Thus, the following line creates the same structure `b` as above:

```
> b<-subset(a, CLASS=="open")¶
```

The formulation "condition(s)" already indicates that you can of course use several conditions at the same time.

```
> b<-subset(a, CLASS=="open" & TOKENFREQ<1000); b¶
  POS TOKENFREQ TYPEFREQ CLASS
1 adj       421      271  open
2 adv       337      103  open
> b<-subset(a, POS %in% c("adj", "adv")); b¶
  POS TOKENFREQ TYPEFREQ CLASS
1 adj       421      271  open
2 adv       337      103  open
```

As I mentioned above, you will usually edit data frames in a spreadsheet software or, because the spreadsheet software does not allow for as many

rows as you need, in a text editor. For the sake of completeness, let me mention that R of course also allows you to edit data frames in a spreadsheet-like format. The function `fix` takes as argument a data frame and opens a spreadsheet editor in which you can edit the data frame; you can even introduce new factor levels without having to define them first. When you close the editor, R will do that for you.

Finally, let us look at ways in which you can sort data frames. Recall that the function `order` creates a vector of positions and that vectors can be used for sorting. Imagine you wanted to search the data frame `a` according to the column `CLASS` (in alphabetically ascending order), and within `Class` according to `TOKENFREQ` (in descending order). How can you do that?

THINK
BREAK

The problem is both sorting styles are different: one is `decreasing=FALSE`, the other is `decreasing=TRUE`. What you can do is apply `order` not to `TOKENFREQ`, but to the negative values of `TOKENFREQ`.

```
> order.index<-order(CLASS, -TOKENFREQ); order.index¶
[1] 4 5 3 1 2
```

After that, you can use the vector `order.index` to sort the data frame:

```
> a[order.index,]¶
   POS TOKENFREQ TYPEFREQ  CLASS
4 conj       458       18 closed
5 prep       455       37 closed
3    n      1411      735   open
1  adj       421      271   open
2  adv       337      103   open
```

Of course you can do that in just one line:[12]

```
> a[order(CLASS, -TOKENFREQ),]¶
```

You can now also use the function `sample` to sort the rows of a data frame randomly (for example, to randomize tables with experimental items;

12. Note that R is superior to many other programs here because the number of sorting parameters is in principle unlimited.

cf. above). You first determine the number of rows to be randomized (e.g., with `nrow` or `dim`) and then combine `sample` with `order`. Your data frame will probably be different because we used a random sampling.

```
> no.rows<-nrow(a)¶
> order.index<-sample(no.rows); order.index¶
[1] 3 4 1 2 5
> a[order.index,]¶
   POS TOKENFREQ TYPEFREQ  CLASS
3    n      1411      735   open
4 conj       458       18 closed
1  adj       421      271   open
2  adv       337      103   open
5 prep       455       37 closed

> a[sample(nrow(a)),] # in just one line¶
```

But what do you do when you need to sort a data frame according to several factors – some in ascending and some in descending order? You can of course not use negative values of factor levels – what would `-open` be? Thus, you first use the function `rank`, which rank-orders factor levels, and then you can use negative values of these ranks:

```
> order.index<-order(-rank(CLASS), -rank(POS))¶
> a[order.index,]¶
   POS TOKENFREQ TYPEFREQ  CLASS
3    n      1411      735   open
2  adv       337      103   open
1  adj       421      271   open
5 prep       455       37 closed
4 conj       458       18 closed
```

Recommendation(s) for further study

- the function `is.data.frame` to test if a data structure is a data frame
- the function `dim` for the number of rows and columns of a data frame
- the functions `read.csv` and `read.csv2` to read in tab-delimited files
- the function `save` to save data structures in a compressed binary format
- the function `with` to access columns of a data frame without `attach`
- the functions `cbind` and `rbind` to combine vectors and factors in a columnwise or rowwise way
- the function `merge` to combine different data frames
- the function `complete.cases` to test which rows of a data frame contain missing data / `NA`

6. Some programming: conditionals and loops

So far, we have focused on simple and existing functions but we have done little to explore the programming-language character of R. This section will introduce a few very powerful notions that allow you to make R decide which of two or more user-specified things to do and/or do something over and over again. In Section 2.6.1, we will explore the former, Section 2.6.2 then discusses the latter, but the treatment here can only be very brief and I advise you to explore some of the reading suggestions for more details.

6.1. Conditional expressions

Later, you will often face situations where you want to pursue one of several possible options in a statistical analysis. In a plot, for example, the data points for male subjects should be plotted in blue and the data points for female subjects should be plotted in pink. Or, you actually only want R to generate a plot when the result is significant but not, when it is not. In general, you can of course always do these things stepwise yourself: you could decide for each analysis yourself whether it is significant and then generate a plot when it is. However, a more elegant way is to write R code that makes decisions for you, that you can apply to any data set, and that, therefore, allows you to recycle code from one analysis to the next. Conditional expressions are one way – others are available and sometimes more elegant – to make R decide things. This is what the syntax can look like in a notation often referred to as pseudo code (so, no need to enter this into R!):

```
if (some logical expression testing a condition) {
   what to do if this logical expression evaluates to TRUE
   (this can be more than one line)
} else if (some other logical expression) {
   what to do if this logical expression evaluates to FALSE
   (this can be more than one line)
} else {
   what to do if all logical expressions above evaluate to
   FALSE
}
```

That's it, and the part after the first `}` is even optional. Here's an example with real code (recall, `"\n"` means 'a new line'):

```
> pvalue<-0.06¶
> if (pvalue>=0.05) {¶
```

```
+    cat("Not significant, p =", pvalue, "\n")¶
+ } else {¶
+    cat("Significant, p =", pvalue, "\n")¶
+ }¶
Not significant, p = 0.06
```

The first line defines a *p*-value, which you will later get from a statistical test. The next line tests whether that *p*-value is greater than or equal to 0.05. It is, which is why the code after the first opening `{` is executed and why R then never gets to see the part after `else`.

If you now set `pvalue` to 0.04 and run the `if` expression again, then this happens: Line 2 from above tests whether 0.04 is greater than or equal to 0.05. It is not, which is why the block of code between `{` and `}` before `else` is skipped and why the second block of code is executed. Try it.

A short version of this can be extremely useful when you have many tests to make but only one instruction for both when a test returns `TRUE` or `FALSE`. It uses the function `ifelse`, here represented schematically again:

```
ifelse(logical expression, what when TRUE, what when FALSE)
```

And here's an application:

```
> pvalues<-c(0.02, 0.00096, 0.092, 0.4)¶
> decisions<-ifelse (pvalues<0.05, "*", "ns")¶
> decisions¶
[1] "*"  "*"  "ns" "ns"
```

As you can see, `ifelse` tested all four values of `pvalues` against the threshold value of 0.05, and put the correspondingly required values into the new vector `decisions`. We will use this a lot to customize graphs.

6.2. Loops

Loops are useful to have R execute one or (many) more functions multiple times. Like many other programming languages, R has different types of loops, but I will only discuss `for`-loops here. This is the general syntax in pseudo code:

```
for (some.name in a.sequence) {
   what to do as often often as a.sequence has elements
   (this can be more than one line)
}
```

Let's go over this step by step. The data structure `some.name` stands for any name you might wish to assign to a data structure that is processed in the loop, and `a.sequence` stands for anything that can be interpreted as a sequence of values, most typically a vector of length 1 or more. This sounds more cryptic than it actually is, here's a very easy example:

```
> for (counter in 1:3) {¶
+    cat("This is iteration number", counter, "\n")¶
+ }¶
This is iteration number 1
This is iteration number 2
This is iteration number 3
```

When R enters the `for`-loop, it assigns to `counter` the first value of the sequence 1:3, i.e. 1. Then, in the only line in the loop, R prints some sentence and ends it with the current value of `counter`, 1, and a line break. Then R reaches the `}` and, because `counter` has not yet iterated over all values of `a.sequence`, re-iterates, which means it goes back to the beginning of the loop, this time assigning to `counter` the next value of `a.sequence`, i.e., 2, and so on. Once R has printed the third line, it exits the loop because `counter` has now iterated over all elements of `a.sequence`.

Here is a more advanced example, but one that is typical of what we're going to use loops for later. Can you see what it does just from the code?

```
> some.numbers<-1:100¶
> collector<-vector(length=10)¶
> for (i in 1:10) {¶
+    collector[i]<-mean(sample(some.numbers, 50))¶
+ }¶
> collector¶
[1] 50.78 51.14 45.04 48.04 55.30 45.90 53.02 48.40 50.38
   49.88
```

THINK
BREAK

The first line generates a vector `some.numbers` with the values from 1 to 100. The second line generates a vector called `collector` which has 10 elements and which will be used to collect results from the looping. Line 3 begins a loop of 10 iterations, using a vector called `i` as the counter. Line 4 is the crucial one now: In it, R samples 50 numbers randomly without replacement from the vector `some.numbers`, computes the mean of these 50

numbers, and then stores that mean in the `i`-th slot of collector. On the first iteration, `i` is of course 1 so the first mean is stored in the first slot of `collector`. Then R iterates, `i` becomes 2, R generates a second random sample, computes its mean, and stores it in the – now – 2nd slot of `collector`, and so on, until R has done the sampling, averaging, and storing process 10 times and exits the loop. Then, the vector `collector` is printed on the screen.

In Chapter 4, we will use an approach like this to help us explore data that violate some of the assumptions of common statistical tests. However, it is already worth mentioning that loops are often not the best way to do things like the above in R: in contrast to some other programming languages, R is designed such that it is often much faster and more memory-efficient to do things not with loops but with members of the `apply` family of functions, which you will get to know a bit later. Still, being able to quickly write a loop and test something is often a very useful skill.

Recommendation(s) for further study
– the functions `next` and `break` to control behavior of/in loops

7. Writing your own little functions

The fact that R is not just a statistics software but a full-fledged programming language is something that can hardly be overstated enough. It means that nearly anything is possible: the limit of what you can do with R is not defined by what the designers of some other software thought you may want to do – the limit is set pretty much only by your skills and maybe your RAM/processor (which is one reason why I recommend using R for corpus-linguistic analyses, see Gries 2009a). One aspect making this particularly obvious is how you can very easily write your own functions to facilitate and/or automate tedious and/or frequent tasks. In this section, I will give a few very small examples of the logic of how to write your own functions, mainly because we haven't dealt with any statistical functions yet. Don't despair if you don't understand these programming issues immediately – for most of this book, you will not need them, but these capabilities can come in very handy when you begin to tackle more complex data. Also, in Chapter 3 and 4 I will return to this topic so that you get more practice in this and end up with a list of useful functions for your own work.

The first example I want to use involves looking at a part of a data structure. For example, let's assume you loaded a really long vector (let's

say, 10,000 elements long) and want to check whether you imported it into R properly. Just printing that onto the screen is somewhat tedious since you can't possibly read all 10,000 items (let alone at the speed with which they are displayed), nor do you usually need all 10,000 items – the first *n* are usually enough to see whether your data import was successful. The same holds for long data frames: you don't need to see all 1600 rows to check whether loading it was successful, maybe the first 5 or 6 are sufficient. Let's write a function `peek` that by default shows you the first 6 elements of each of the data structures you know about: one-dimensional vectors or factors and two-dimensional data frames.

One good way to approach the writing of functions is to first consider how you would solve that problem just for a particular data structure, i.e. outside of the function-writing context, and then make whatever code you wrote general enough to cover not just the one data structure you just addressed, but many more. To that end, let's first load a data frame for this little example (from <_inputfiles/02-7_dataframe1.csv>):

```
> into.causatives<-read.delim(file.choose())¶
> str(into.causatives)¶
'data.frame':1600 obs. of  5 variables:
$ BNC       : Factor w/ 929 levels "A06","A08","A0C",..:
   1 2 3 4 ...
$ TAG_ING   : Factor w/ 10 levels "AJ0-NN1","AJ0-VVG",..:
   10 7 10 ...
$ ING       : Factor w/ 422 levels "abandon-
   ing","abdicating",..: 354 49 382 ...
$ VERB_LEMMA: Factor w/ 208 levels "activate","aggravate",..:
   76 126 186 ...
$ ING_LEMMA : Factor w/ 417 levels "abandon","abdicate",..:
   349 41 377 ...
```

Now, you want to work with one-dimensional and two-dimensional vectors, factors, and data frames. How would you get the first six elements of each of these? That you already know. For vectors or factors you'd write:

```
vector.or.factor[1:6]
```

and for data frames you'd write:

```
data.frame[1:6,]
```

So, essentially you need to decide what the data structure is of which R is supposed to display the first *n* elements (by default 6) and then you subset with either `[1:6]` or `[1:6,]`. Since, ultimately, the idea is to have R –

not you – decide on the right way of subsetting (depending on the data structure), you use a conditional expression:

```
> if (is.data.frame(into.causatives)) {¶
>    into.causatives[1:6,]¶
> } else {¶
>    into.causatives[1:6]¶
> }¶
  BNC TAG_ING      ING VERB_LEMMA ING_LEMMA
1 A06     VVG speaking      force     speak
2 A08     VBG    being      nudge        be
3 A0C     VVG   taking       talk       tak
4 A0F     VVG   taking      bully      take
5 A0H     VVG   trying  influence       try
6 A0H     VVG thinking     delude     think
```

To turn this into a function, you wrap a function definition (naming the function `peek`) around this piece of code. However, if you use the above code as is, then this function will use the name `into.causatives` in the function definition, which is not exactly very general. As you have seen, many R functions use `x` for the main obligatory variable. Following this tradition, you could write this:

```
> peek<-function (x) {¶
>    if (is.data.frame(x)) {¶
>       x[1:6,]¶
>    } else {¶
>       x[1:6]¶
>    }¶
> }¶
> peek(into.causatives)¶
```

This means, R defines a function called `peek` that requires an argument, and that argument is function-internally called `x`. When you call `peek` with some argument – e.g., `into.causatives` – then R will take the content of that data structure and, for the duration of the function execution, assign it to `x`. Then, within the function R will carry out all of `peek` with `x` and return/output the result, which is the first 6 rows of `into.causatives`.

It seems like we're done. However, some things are missing. When you write a function, it is crucial you make sure it covers all sorts of possibilities or data you may throw at it. After all, you're writing a function to make your life easier, to allow you not to have to worry about stuff anymore after you have thought about it once, namely when you wrote the function. There are three ways in which the above code should be improved:

– what if the data structure you use `peek` with is not a vector or a factor or

a data frame?
- what if you want to be able to see not 6 but *n* elements?
- what if the data structure you use `peek` with has fewer than *n* elements or rows?

To address the first possibility, we just add another conditional expression. So far we only test whether whatever we use `peek` with is a data frame – now we also need to check whether, if it is not a data frame, whether it then is a vector or a factor, and ideally we return some warning if the data structure is none of the three.

To address the second possibility, we need to be able to tell the function flexibly how many parts of `x` we want to see, and the way we tell this to a function is of course by its arguments. Thus, we add an argument, let's call it `n`, that says how much we want to see of `x`, but we make 6 the default.

To address the final possibility, we have to make sure that R realizes how many elements `x` has: if it has more than `n`, R should show `n`, but if it has fewer than `n`, R should show as many as it can, i.e., all of them.

This version of `peek` addresses all of these issues:

```
> peek< function (x, n=6) {¶
>    if (is.data.frame(x)) {¶
>       return(x[1:min(nrow(x), n),])¶
>    } else if (is.vector(x) | is.factor(x)) {¶
>       return(x[1:min(length(x), n)])¶
>    } else {¶
>       cat("Not defined for other data structures ...\n")¶
>    }¶
> }¶
```

Issue number one is addressed by adding a second conditional with the `else if` test – recall the use of | to mean 'or' – and outputting a message if `x` is neither a vector, factor, or a data frame.

Issue number two is addressed by adding the argument `n` to the function definition and using `n` in the body of the function. The argument `n` is set to 6 by default, so if the user does not specify `n`, 6 is used, but the user can also override this with another number.

The final issue is addressed by tweaking the subsetting: instead of using just `n`, we use `1:` the minimum of `n` or the number of elements `x` has. Thus, if `x` has more than `n` elements, then `n` will be the minimum and we get to see `n` elements, and if `x` has less than `n` elements, then that number of elements will be the minimum and we get to see them all.

Finally, also note that I am now using the function `return` to specify

exactly what `peek` should return and output to the user when it's done. Try the following lines (output not shown here and see the comments in the code file) to see that it works:

```
> peek(into.causatives)¶
> peek(into.causatives, 3)¶
> peek(into.causatives, 9)¶
> peek(21:50, 10)¶
> peek(into.causatives$BNC, 12)¶
> peek(as.matrix(into.causatives))¶
```

While all this may not seem easy and worth the effort, we will later see that being able to write your own functions will facilitate quite a few statistical analyses below. Let me also note that this was a tongue-in-cheek example: there is actually already a function in R that does what `peek` does (and more, because it can handle more data structures) – look up `head` and also `tail` ;-).

Now you should do the exercise(s) for Chapter 2 …

Recommendation(s) for further study

- the functions `NA`, `is.na`, `NaN`, `is.nan`, `na.action`, `na.omit`, and `na.fail` on how to handle missing data
- Ligges (2005), Crawley (2007), Braun and Murdoch (2008), Spector (2008), Gentleman (2009), and Gries (2009a) for more information on R: Ligges (2005), Braun and Murdoch (2008), and Gentleman (2009) on R as a (statistical) programming language, Crawley as a very comprehensive overview, Spector (2008) on data manipulation in R, and Gries (2009a) on corpus-linguistic methods with R

Chapter 3
Descriptive statistics

> Any 21st century linguist will be required to read about and understand mathematical models as well as understand statistical methods of analysis. Whether you are interested in Shakespearean meter, the sociolinguistic perception of identity, Hindi verb agreement violations, or the perception of vowel duration, the use of math as a tool of analysis is already here and its prevalence will only grow over the next few decades. If you're not prepared to read articles involving the term *Bayesian*, or *(p<.01)*, k-*means clustering*, *confidence interval*, *latent semantic analysis*, *bimodal and unimodal distributions*, N-*grams*, etc, then you will be but a shy guest at the feast of linguistics.
>
> (<http://thelousylinguist.blogspot.com/2010/01/why-linguists-should-study-math.html>)

In this chapter, I will explain how you obtain descriptive results. In section 3.1, I will discuss univariate statistics, i.e. statistics that summarize the distribution of one variable, of one vector, of one factor. Section 3.2 then is concerned with bivariate statistics, statistics that characterize the relation of two variables, two vectors, two factors to each other. Both sections also introduce ways of representing the data graphically; many additional graphs will be illustrated in Chapters 4 and 5.

1. Univariate statistics

1.1. Frequency data

The probably simplest way to describe the distribution of data points are frequency tables, i.e. lists that state how often each individual outcome was observed. In R, generating a frequency table is extremely easy. Let us look at a psycholinguistic example. Imagine you extracted all occurrences of the disfluencies *uh*, *uhm*, and 'silence' and noted for each disfluency whether it was produced by a male or a female speaker, whether it was produced in a monolog or in a dialog, and how long in milliseconds the disfluency lasted. First, we load these data from the file <_inputfiles/03-1_uh(m).csv>.

```
> UHM<-read.delim(file.choose())¶
> str(UHM)¶
'data.frame':   1000 obs. of  5 variables:
 $ CASE  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ SEX   : Factor w/ 2 levels "female","male": 2 1 1 1 2  ...
 $ FILLER: Factor w/ 3 levels "silence","uh",..: 3 1 1 3  ...
 $ GENRE : Factor w/ 2 levels "dialog","monolog": 2 2 1 1 ...
 $ LENGTH: int  1014 1188 889 265 465 1278 671 1079 643 ...
> attach(UHM)¶
```

To see which disfluency or filler occurs how often, you use the function `table`, which creates a frequency list of the elements of a vector or factor:

```
> table(FILLER)¶
FILLER
silence      uh     uhm
    332     394     274
```

If you also want to know the percentages of each disfluency, then you can either do this rather manually or you use the function `prop.table`, whose argument is a table generated with `table` and which returns the percentages of the frequencies in that table (cf. also below).

```
> table(FILLER)/length(FILLER)¶
FILLER
silence      uh     uhm
  0.332   0.394   0.274
> prop.table(table(FILLER))¶
FILLER
silence      uh     uhm
  0.332   0.394   0.274
```

Often, it is also useful to generate a cumulative frequency table of the observed values or of the percentages. R has a function `cumsum`, which successively adds the values of a vector and returns all sums, which is exemplified in the following two lines:

```
> 1:5¶
[1] 1 2 3 4 5
> cumsum(1:5)¶
[1]  1  3  6 10 15
```

And of course you can apply `cumsum` to our tables:

```
> cumsum(table(FILLER))¶
silence      uh     uhm
    332     726    1000
```

```
> cumsum(prop.table(table(FILLER)))¶
silence      uh     uhm
  0.332   0.726   1.000
```

Usually, it is instructive to represent the observed distribution graphically and the sections below introduce a few graphical formats. For reasons of space, I only discuss some ways to tweak graphs, but you can turn to the help pages of these functions (using `?…`) and Murrell (2011) for more info.

1.1.1. Scatterplots and line plots

Before we begin to summarize vectors and factors graphically in groups of elements, we discuss how the data points of a vector are plotted individually. The simplest approach just requires the function `plot`. This is a very versatile function, which, depending on the arguments you use with it, creates many different graphs. (This may be a little confusing at first, but allows for an economical style of working, as you will see later.) If you provide just one numerical vector as an argument, then R plots a scatterplot, i.e., a two-dimensional coordinate system in which the values of the vector are interpreted as coordinates of the *y*-axis, and the order in which they appear in the vector are the coordinates of the *x*-axis. Here's an example:

```
> a<-c(1, 3, 5, 2, 4); b<-1:5¶
> plot(a) # left panel of Figure 15¶
```

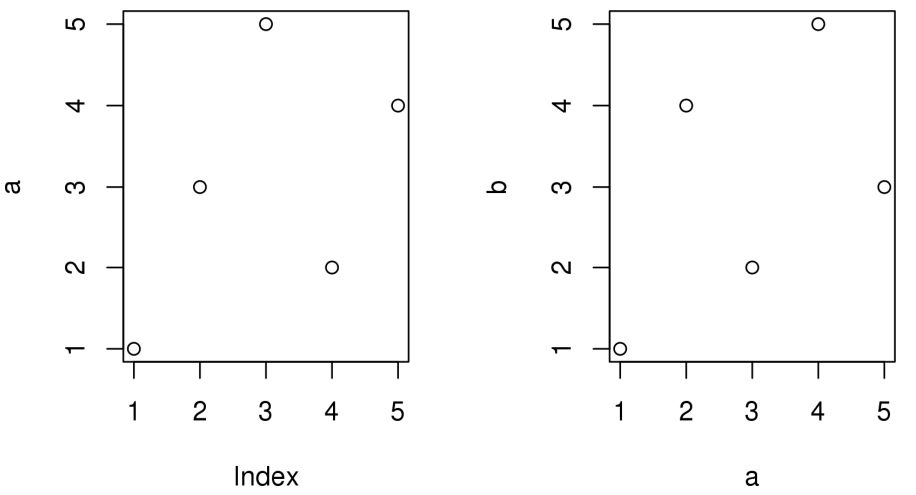

Figure 15. Simple scatterplots

But if you give two vectors as arguments, then the values of the first and the second are interpreted as coordinates on the *x*-axis and the *y*-axis respectively (and the names of the vectors will be used as axis labels):

```
> plot(a, b) # right panel of Figure 15¶
```

With the argument `type=…`, you can specify the kind of graph you want. The default, which was used because you did not specify anything else, is `type="p"` (for *points*). If you use `type="b"` (for *both*), you get points and lines connecting the points; if you use `type="l"` (for *lines*), you get a line plot; cf. Figure 16. (With `type="n"`, nothing gets plotted into the main plotting area, but the coordinate system is set up.)

```
> plot(b, a, type="b") # left panel of Figure 16¶
> plot(b, a, type="l") # right panel of Figure 16¶
```

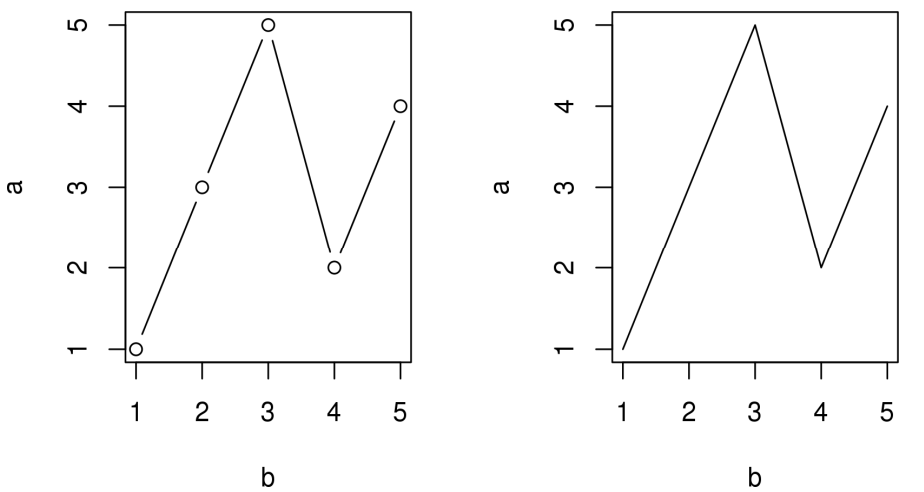

Figure 16. Simple line plots

Other simple but useful ways to tweak graphs involve defining labels for the axes (`xlab="…"` and `ylab="…"`), a bold heading for the whole graph (`main="…"`), the ranges of values of the axes (`xlim=…` and `ylim=…`), and the addition of a grid (`grid()¶`). With `col="…"`, you can also set the color of the plotted element, as you will see more often below.

```
> plot(b, a, xlab="A vector b", ylab="A vector a", xlim=c(0,
   8), ylim=c(0, 8), type="b"); grid() # Figure 17¶
```

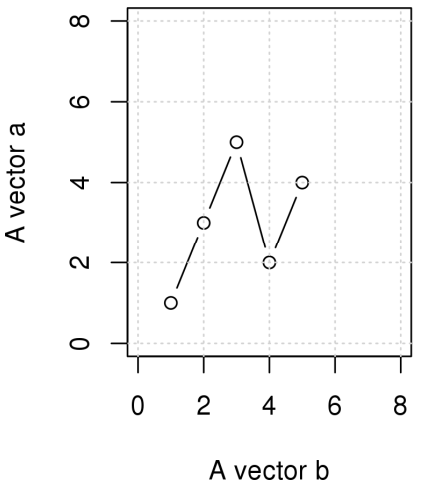

Figure 17. A scatterplot exemplifying a few simple plot settings

An important rule of thumb is that the ranges of the axes must be chosen such that the distribution of the data is represented most meaningfully. It is often useful to include the point (0, 0) within the ranges of the axes and to make sure that graphs to be compared have the same and sufficient axis ranges. For example, if you want to compare the ranges of values of two vectors `x` and `y` in two graphs, then you usually may not want to let R decide on the ranges of axes. Consider the upper panel of Figure 18.

The clouds of points look very similar and you only notice the distributional difference between `x` and `y` when you specifically look at the range of values on the *y*-axis. The values in the upper left panel range from 0 to 2 but those in the upper right panel range from 0 to 6. This difference between the two vectors is immediately obvious, however, when you use `ylim=…` to manually set the ranges of the *y*-axes to the same range of values, as I did for the lower panel of Figure 18.

Note: whenever you use `plot`, by default a new graph is created and the old graph is lost (In RStudio, you can go back to previous plots, however, with the arrow button or the menu *Plots: ...*) If you want to plot two lines into a graph, you first generate the first with `plot` (and `type="l"` or `type="b"`) and then add the second one with `points` (or `lines`; sometimes you can also use the argument `add=TRUE`). That also means that you must define the ranges of the axes in the first plot in such a way that the values of the second graph can also be plotted into it.

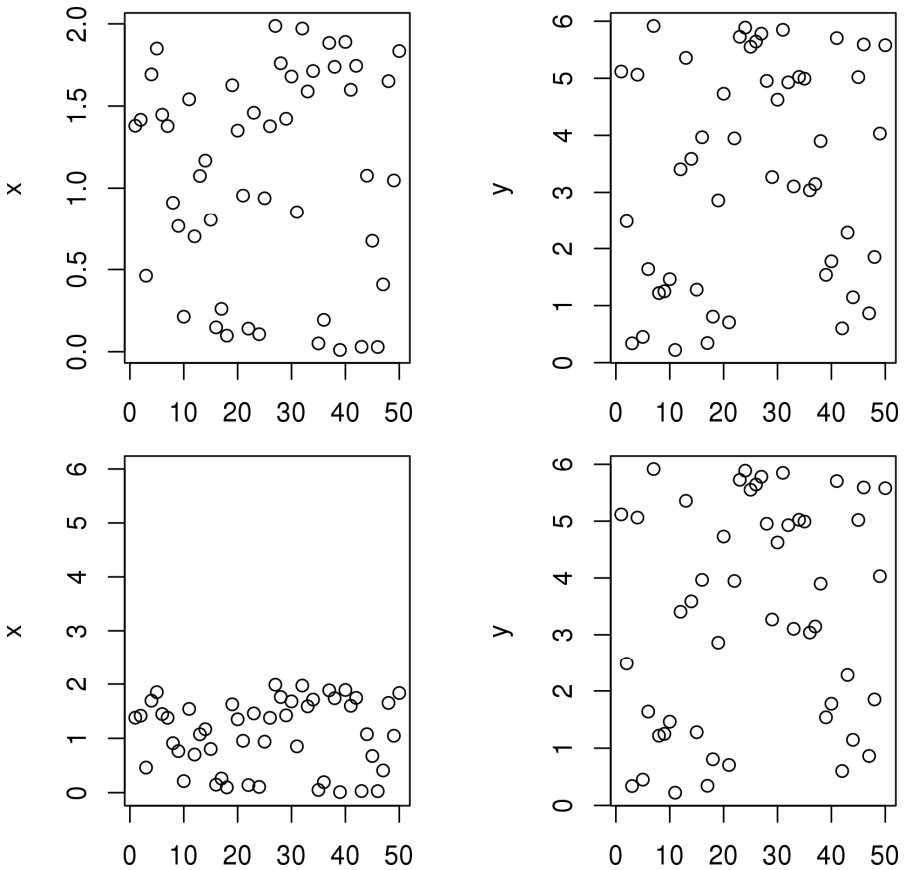

Figure 18. Scatterplots and the importance of properly-defined ranges of axes

An example will clarify that point. If you want to plot the points of the vectors `m` and `n`, and then want to add into the same plot the points of the vectors `x` and `y`, then this does *not* work, as you can see in the left panel of Figure 19.

```
> m<-1:5; n<-5:1¶
> x<-6:10; y<-6:10¶
> plot(m, n, type="b"); points(x, y, type="b"); grid()¶
```

The left panel of Figure 19 shows the points defined by `m` and `n`, but not those of `x` and `y` because the ranges of the axes that R used to plot `m` and `n` are too small for `x` and `y`, which is why you must define those manually while creating the first coordinate system. One way to do this is to use the

function `max`, which returns the maximum value of a vector (and `min` returns the minimum). The right panel of Figure 19 shows that this does the trick. (In this line, the minimum is set to 0 manually – of course, you could also use `min(m, x)` and `min(n, y)` for that, but I wanted to include (0, 0) in the graph.)

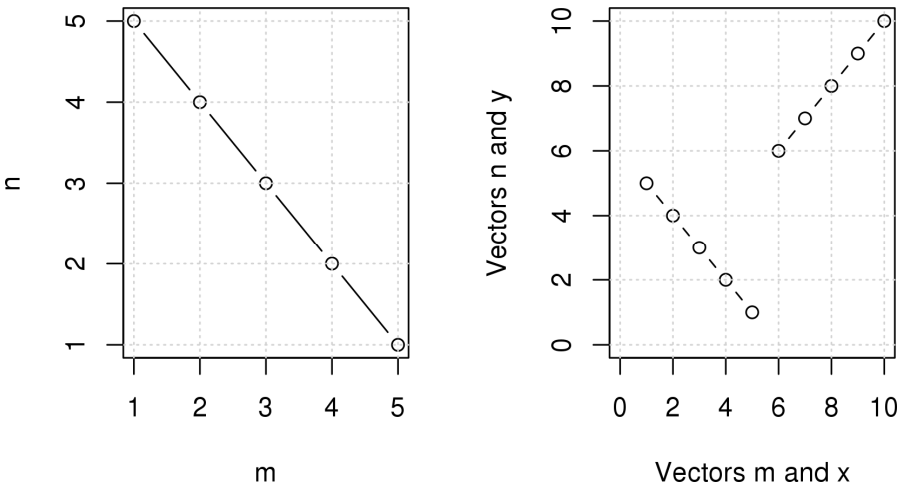

Figure 19. Scatterplots and the importance of properly-defined ranges of axes

```
> plot(m, n, type="b", xlim=c(0, max(m, x)), ylim=
   c(0, max(n, y)), xlab="Vectors m and x",
   ylab="Vectors n and y"); grid()¶
> points(x, y, type="b")¶
```

Recommendation(s) for further study
the functions `pmin` and `pmax` to determine the minima and maxima at each position of different vectors (try `pmin(c(1, 5, 3), c(2, 4, 6))¶`)

1.1.2. Pie charts

The function to generate a pie chart is `pie`. Its most important argument is a table generated with `table`. You can either just leave it at that or, for example, change category names with `labels=…` or use different colors with `col=…` etc.:

```
> pie(table(FILLER), col=c("grey20", "grey50", "grey80"))¶
```

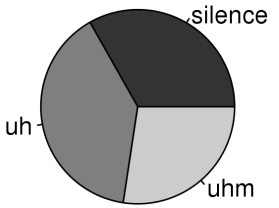

Figure 20. A pie chart with the frequencies of disfluencies

One thing that's a bit annoying about this is that, to use different colors with `col=…` as above, you have to know how many colors there are and assign names to them, which becomes cumbersome with many different colors and/or graphs. For situations like these, the function `rainbow` can be very useful. In its simplest use, it requires only one argument, namely the number of different colors you want. Thus, how would you re-write the above line for the pie chart in such a way that you let R find out how many colors are needed rather than saying `col=rainbow(3)`?

THINK
BREAK

Let R use as many colors as the table you are plotting has elements:

```
> pie(table(FILLER), col=rainbow(length(table(FILLER))))¶
```

Note that pie charts are usually not a good way to summarize data because humans are not very good at inferring quantities from angles. Thus, `pie` is not a function you should use too often – the function `rainbow`, on the other hand, is one you should definitely bear in mind.

1.1.3. Bar plots

To create a bar plot, you can use the function `barplot`. Again, its most important argument is a table generated with `table` and again you can create either a standard version or more customized ones. If you want to define your own category names, you unfortunately must use `names.arg=…`, not `labels=…` (cf. Figure 21 below).

```
> barplot(table(FILLER)) # left panel of Figure 21¶
> barplot(table(FILLER), col=c("grey20", "grey40",
   "grey60")) # right panel of Figure 21¶
```

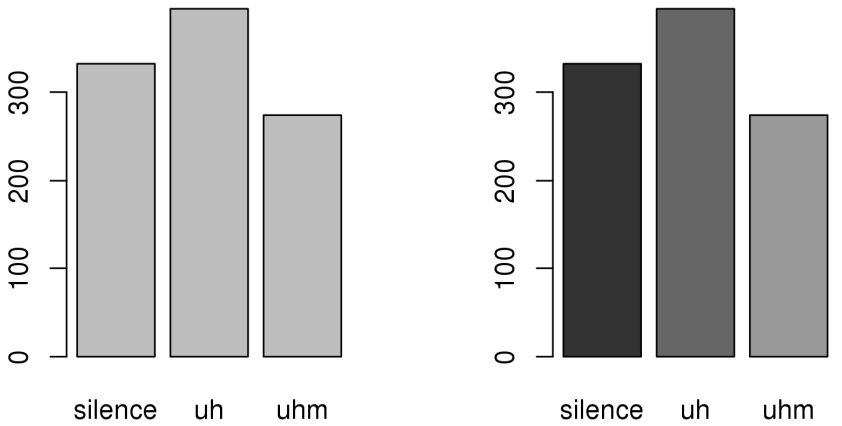

Figure 21. Bar plots with the frequencies of disfluencies

An interesting way to configure bar plots is to use `space=0` to have the bars be immediately next to each other. That is of course not exactly mind-blowing in itself, but it is one of two ways to make it easier to add further data/annotation to the plot. For example, you can then easily plot the observed frequencies into the middle of each bar using the function `text`. The first argument of `text` is a vector with the *x*-axis coordinates of the text to be printed (with `space=0`, 0.5 for the middle of the first bar, 1.5 for the middle of the second bar, and 2.5 for the middle of the third bar), the second argument is a vector with the *y*-axis coordinates of that text (half of each observed frequency so that the text ends up in the middle of the bars), and `labels=…` provides the text to be printed; cf. the left panel of Figure 22.

```
> barplot(table(FILLER), col=c("grey40", "grey60", "grey80"),
   names.arg=c("Silence", "Uh", "Uhm"), space=0)¶
> text(c(0.5, 1.5, 2.5), table(FILLER)/2, labels=
   table(FILLER))¶
```

The second way to create a similar graph – cf. the right panel of Figure 22 – involves some useful changes:

```
> mids<-barplot(table(FILLER), col=c("grey40", "grey60",
   "grey80"))¶
> text(mids, table(FILLER), labels=table(FILLER), pos=1)¶
```

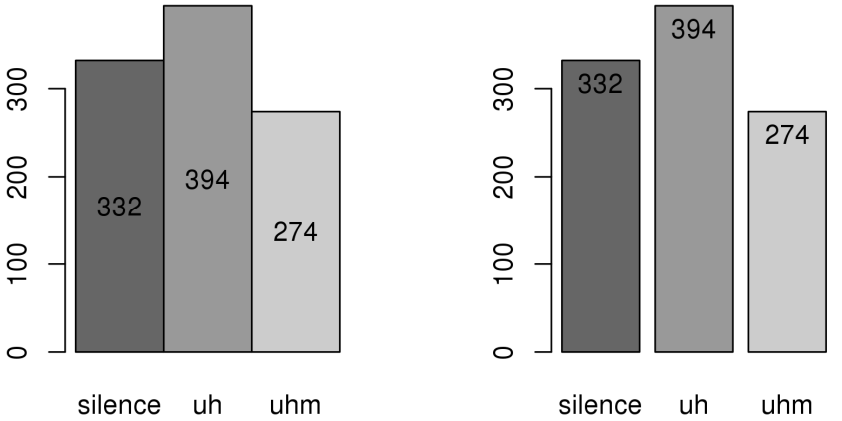

Figure 22. Bar plots with the frequencies of disfluencies

The first line now does not just plot the barplot, it also assigns what R returns to a data structure called `mids`, which contains the *x*-coordinates of the middles of the bars, which we can then use for `texting`. (Look at `mids`.) Second, the second line now uses `mids` for the *x*-coordinates of the text to be printed and it uses `pos=1` to make R print the text a bit below the specified coordinates; `pos=2`, `pos=3`, and `pos=4` would print the text a bit to the left, above, and to the right of the specified coordinates respectively.

The functions `plot` and `text` allow for another powerful graph: first, you generate a plot that contains nothing but the axes and their labels (with `type="n"`, cf. above), and then with `text` you plot words or numbers. Try this for an illustration of a kind of plot you will more often see below:

```
> tab<-table(FILLER)¶
> plot(tab, type="n", xlab="Disfluencies", ylab="Observed
   frequencies", xlim=c(0, 4), ylim=c(0, 500)); grid()¶
> text(seq(tab), tab, labels=tab)¶
```

Recommendation(s) for further study
the function `dotchart` for dot plot and the parameter settings `cex`, `srt`, `col`, `pch`, and `font` to tweak plots: `?par¶`.

1.1.4. Pareto-charts

A related way to represent the frequencies of the disfluencies is a pareto-chart. In pareto-charts, the frequencies of the observed categories are repre-

sented as in a bar plot, but they are first sorted in descending order of frequency and then overlaid by a line plot of cumulative percentages that indicates what percent of all data one category *and* all other categories to the left of that category account for. The function `pareto.chart` comes with the library `qcc` that you must (install and/or) load first; cf. Figure 23.

```
> library(qcc)¶
> pareto.chart(table(FILLER), main="")¶
Pareto chart analysis for table(FILLER)
           Frequency Cum.Freq. Percentage Cum.Percent.
  uh           394.0     394.0       39.4         39.4
  silence      332.0     726.0       33.2         72.6
  uhm          274.0    1000.0       27.4        100.0
```

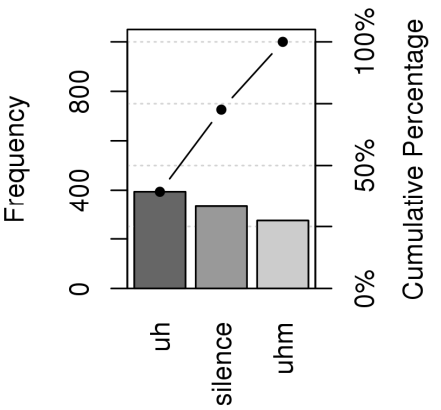

Figure 23. Pareto-chart with the frequencies of disfluencies

1.1.5. Histograms

While bar plots are probably the most frequent forms of representing the frequencies of nominal/categorical variables, histograms are most widespread for the frequencies of interval/ratio variables. In R, you can use `hist`, which just requires the relevant vector as its argument.

```
> hist(LENGTH)¶
```

For some ways to make the graph nicer, cf. Figure 24, whose left panel contains a histogram of the variable `LENGTH` with axis labels and grey bars.

```
> hist(LENGTH, main="", xlab="Length in ms", ylab=
   "Frequency", xlim=c(0, 2000), ylim=c(0, 100),
   col="grey80")¶
```

The right panel of Figure 24 contains a histogram of the probability densities (generated by `freq=FALSE`) with a curve (generated by `lines`).

```
> hist(LENGTH, main="", xlab="Length in ms", ylab="Density",
   freq=FALSE, xlim=c(0, 2000), col="grey50")¶
> lines(density(LENGTH))¶
```

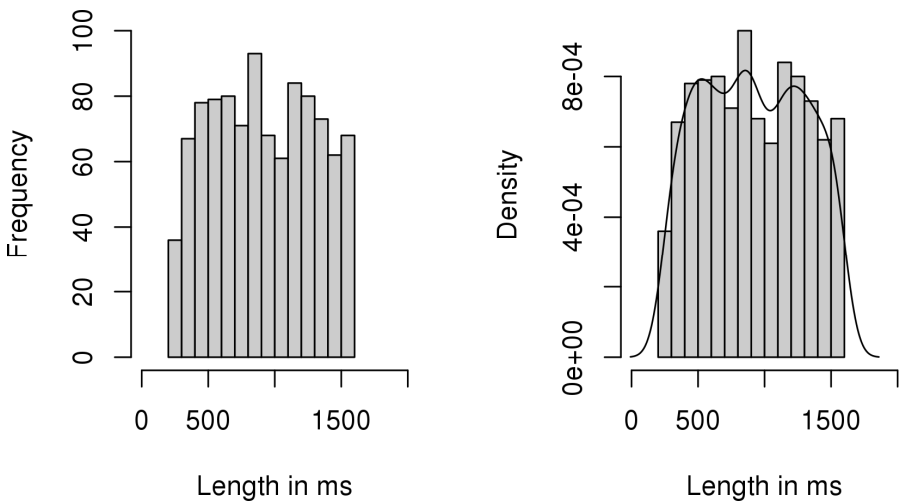

Figure 24. Histograms for the frequencies of lengths of disfluencies

With the argument `breaks=…` to `hist`, you can instruct R to try to use a particular number of bins (or bars). You either provide one integer – then R tries to create a histogram with as many bins – or you provide a vector with the boundaries of the bins. The latter raises the question of how many bins should or may be chosen? In general, you should not have more than 20 bins, and as one rule of thumb for the number of bins to choose you can use the formula in (14) (cf. Keen 2010:143–160 for discussion). The most important aspect is that the bins you choose do not misrepresent the data.

(14) Number of bins for a histogram of n data points = $1+3.32 \cdot \log_{10} n$

1.1.6. Empirical cumulatuive distributions

A very useful visualization of numerical data is the empirical cumulative distribution (function, abbreviated ecdf) plot, an example of which you have already seen as part of the pareto chart in Section 3.1.1.4. On the *x*-axis of an ecdf plot, you find the range of the variable that is visualized, on the *y*-axis you find a percentage scale from 0 to 1 (=100%), and the points in the coordinate system show how much in percent of all data one variable value and all other smaller values to the left of that value account for. Figure 25 shows such a plot for LENGTH and you can see that approximately 18% of all lengths are smaller than 500 ms.

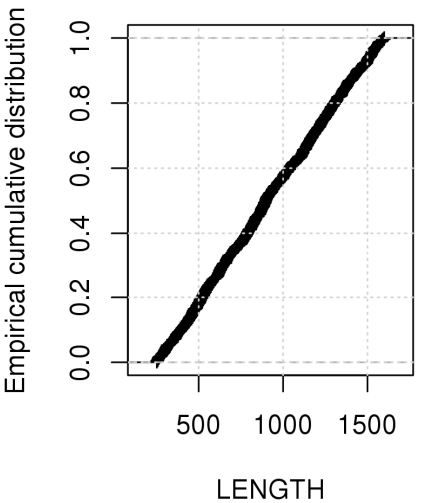

Figure 25. Ecdf plot of lengths of disfluencies

This plot is very useful because it does not lose information by binning data points: every data point is represented in the plot, which is why ecdf plots can be very revealing even for data that most other graphs cannot illustrate well. Let's see whether you've understood this plot: what do ecdf plots of normally-distributed and uniformly-distributed data look like?

THINK
BREAK

You will find the answer in the code file (with graphs); make sure you understand why so you can use this very useful type of graph.

Recommendation(s) for further study

- the functions `dotchart` and `stripchart` (with `method="jitter"`) to represent the distribution of individual data points in very efficient ways
- the function `scatterplot` (from the library `car`) for more sophisticated scatterplots
- the functions `plot3d` and `scatterplot3d` (from the library `rgl` and the library `scatterplot3d`) for different three-dimensional scatterplots

1.2. Measures of central tendency

Measures of central tendency are probably the most frequently used statistics. They provide a value that attempts to summarize the behavior of a variable. Put differently, they answer the question, if I wanted to summarize this variable and were allowed to use only one number to do that, which number would that be? Crucially, the choice of a particular measure of central tendency depends on the variable's level of measurement. For nominal/categorical variables, you should use the mode (if you do not simply list frequencies of all values/bins anyway, which is often better), for ordinal variables you should use the median, for interval/ratio variables you can often use the arithmetic mean.

1.2.1. The mode

The mode of a variable or distribution is the value that is most often observed. As far as I know, there is no function for the mode in R, but you can find it very easily. For example, the mode of `FILLER` is *uh*:

```
> which.max(table(FILLER))¶
uh
 2
> max(table(FILLER))¶
[1] 394
```

Careful when there is more than one level that exhibits the maximum number of observations – tabulating is usually safer.

1.2.2. The median

The measure of central tendency for ordinal data is the median, the value you obtain when you sort all values of a distribution according to their size and then pick the middle one (e.g., the median of the numbers from 1 to 5 is 3). If you have an even number of values, the median is the average of the two middle values.

```
> median(LENGTH)¶
[1] 897
```

1.2.3. The arithmetic mean

The best-known measure of central tendency is the arithmetic mean for interval/ratio variables. You compute it by adding up all values of a distribution or a vector and dividing that sum by the number of values, but of course there is also a function for this:

```
> sum(LENGTH)/length(LENGTH)¶
[1] 915.043
> mean(LENGTH)¶
[1] 915.043
```

One weakness of the arithmetic mean is its sensitivity to outliers:

```
> a<-1:10; a¶
[1]  1  2  3  4  5  6  7  8  9 10
> b<-c(1:9, 1000); b¶
[1]    1    2    3    4    5    6    7    8    9 1000
> mean(a)¶
[1] 5.5
> mean(b)¶
[1] 104.5
```

Although the vectors `a` and `b` differ with regard to only a single value, the mean of `b` is much larger than that of `a` because of that one outlier, in fact so much larger that `b`'s mean of 104.5 neither summarizes the values from 1 to 9 nor the value 1000 very well. There are two ways of handling such problems. First, you can add the argument `trim=…`, the percentage of elements from the top and the bottom of the distribution that are discarded before the mean is computed. The following lines compute the means of `a` and `b` after the highest and the lowest value have been discarded:

```
> mean(a, trim=0.1)¶
[1] 5.5
> mean(b, trim=0.1)¶
[1] 5.5
```

Second, you can just use the median, which is also a good idea if the data whose central tendency you want to report are not normally distributed.

```
> median(a); median(b)¶
[1] 5.5
[1] 5.5
```

Warning/advice
Just because R or your spreadsheet software can return many decimals does not mean you have to report them all. Use a number of decimals that makes sense given the statistic that you report.

1.2.4. The geometric mean

The geometric mean is used to compute averages of factors or ratios (whereas the arithmetic mean is computed to get the average of sums). Let's assume you have six recordings of a child at the ages 2;1 (two years and one month), 2;2, 2;3, 2;4, 2;5, and 2;6. Let us also assume you had a vector `lexicon` that contains the cumulative numbers of different words (types!) that the child produced at each age:

```
> lexicon<-c(132, 158, 169, 188, 221, 240)¶
> names(lexicon)<-c("2;1", "2;2", "2;3", "2;4", "2;5",
   "2;6")¶
```

You now want to know the average rate at which the lexicon increased. First, you compute the successive increases:

```
> increases<-lexicon[2:6]/lexicon[1:5]; increases¶
     2;2      2;3      2;4      2;5      2;6
1.196970 1.069620 1.112426 1.175532 1.085973
```

That is, by age 2;2, the child produced 19.697% more types than by age 2;1, by age 2;3, the child produced 6.962% more types than by age 2;2, etc. Now, you must *not* think that the average rate of increase of the lexicon is the arithmetic mean of these increases:

```
> mean(increases) # wrong!¶
[1] 1.128104
```

You can easily test that this is not the correct result. If this number was the true average rate of increase, then the product of 132 (the first lexicon size) and this rate of 1.128104 to the power of 5 (the number of times the supposed 'average rate' applies) should be the final value of 240. This is not the case:

```
> 132*mean(increases)^5¶
[1] 241.1681
```

Instead, you must compute the geometric mean. The geometric mean of a vector `x` with n elements is computed according to formula (15), and if you use this as the average rate of increase, you get the right result:

(15) $$mean_{geom} = (x_1 \cdot x_2 \cdot \ldots \cdot x_{n-1} \cdot x_n)^{1/n}$$

```
> rate.increase<-prod(increases)^(1/length(increases));
   rate.increase¶
[1] 1.127009
> 132*rate.increase^5¶
[1] 240
```

True, the difference between 240 – the correct value – and 241.1681 – the incorrect value – may seem negligible, but 241.1681 is still wrong and the difference is not always that small, as an example from Wikipedia (s.v. *geometric mean*) illustrates: If you do an experiment and get an increase rate of 10.000 and then you do a second experiment and get an increase rate of 0.0001 (i.e., a decrease), then the average rate of increase is not approximately 5.000 – the arithmetic mean of the two rates – but 1 – their geometric mean.[13]

Finally, let me again point out how useful it can be to plot words or numbers instead of points, triangles, … Try to generate Figure 26, in which the position of each word on the *y*-axis corresponds to the average length of the disfluency (e.g., 928.4 for women, 901.6 for men, etc.). (The horizontal line is the overall average length – you may not know yet how to plot that one.) Many tendencies are immediately obvious: men are below the average, women are above, silent disfluencies are of about average length, etc.

13. Alternatively, you can compute the geometric mean of `increases` as follows: `exp(mean(log(increases)))¶`.

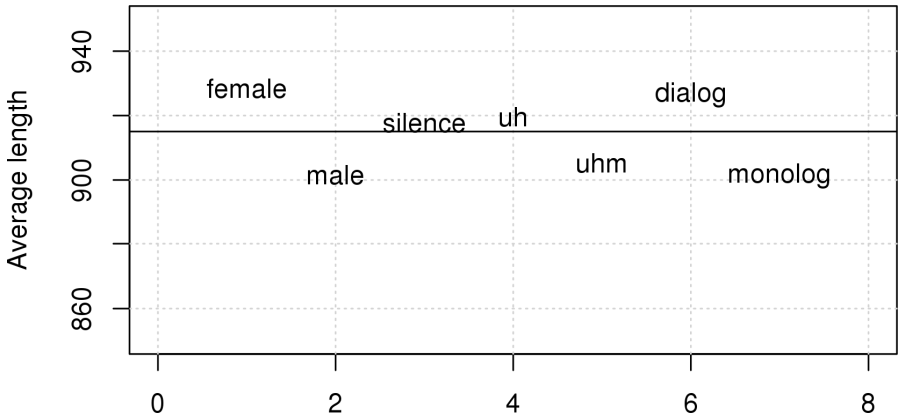

Figure 26. Mean lengths of disfluencies

1.3. Measures of dispersion

Most people know what measures of central tendencies are. What many people do not know is that they should never – NEVER! – report a measure of central tendency without some corresponding measure of dispersion. The reason for this rule is that without such a measure of dispersion you never know how good the measure of central tendency actually is at summarizing the data. Let us look at a non-linguistic example, the monthly temperatures of two towns and their averages:

```
> town1<-c(-5, -12, 5, 12, 15, 18, 22, 23, 20, 16, 8, 1)¶
> town2<-c(6, 7, 8, 9, 10, 12, 16, 15, 11, 9, 8, 7)¶
> mean(town1); mean(town2)¶
[1] 10.25
[1] 9.833333
```

On the basis of the means alone, the towns seem to have a very similar climate, but even a quick glance at Figure 27 shows that that is not true – in spite of the similar means, I know where I would want to be in February. Obviously, the mean of Town 2 summarizes the central tendency of Town 2 much better than the mean of Town 1 does for Town 1: the values of Town 1 vary much more widely around their mean. Thus, always provide a measure of dispersion for your measure of central tendency: relative entropy for the mode, the interquartile range or quantiles for the median and interval/ratio-scaled data that are non-normal or exhibit outliers, and the standard deviation or the variance for normal interval/ratio-scaled data.

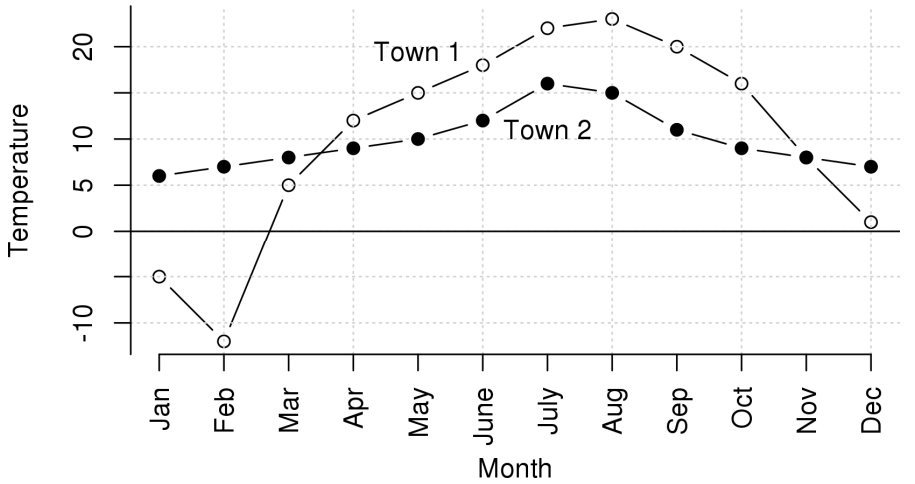

Figure 27. Temperature curves of two towns

1.3.1. Relative entropy

A simple dispersion measure for categorical data is relative entropy H_{rel}. H_{rel} is 1 when the levels of the relevant categorical variable are all equally frequent, and it is 0 when all data points have only one and the same variable level. For categorical variables with n levels, H_{rel} is computed as shown in formula (16), in which p_i corresponds to the frequency in percent of the i-th level of the variable:

$$H_{\text{rel}} = -\frac{\sum_{i=1}^{n}\left(p_i \cdot \ln p_i\right)}{\ln n} \tag{16}$$

Thus, if you count the articles of 300 noun phrases and find 164 cases with no determiner, 33 indefinite articles, and 103 definite articles, this is how you compute H_{rel}:

```
> article<-c(164, 33, 103)¶
> perc<-article/sum(article)¶
> hrel<--sum(perc*log(perc))/log(length(perc)); hrel¶
[1] 0.8556091
```

It is worth pointing out that the above formula does not produce the de-

sired result of 0 when only no-determiner cases are observed because `log(0)` is not defined:

```
> article<-c(300, 0, 0)¶
> perc<-article/sum(article)¶
> hrel<--sum(perc*log(perc))/log(length(perc)); hrel¶
[1] NaN
```

Usually, this is taken care of by simply setting the result of `log(0)` to zero (or sometimes also by incrementing all values by 1 before logging). This is a case where writing a function to compute logarithms that *can* handle 0s can be useful. For example, this is how you could define your own logarithm function `logw0` and then use that function instead of `log` to get the desired result:

```
> logw0<-function(x) {¶
+    ifelse (x==0, 0, log(x))¶
+ }¶
> hrel<--sum(perc*logw0(perc))/logw0(length(perc)); hrel¶
[1] 0
```

Distributions of categorical variables will be dealt with in much more detail below in Section 4.1.1.2.

1.3.2. The range

The simplest measure of dispersion for interval/ratio data is the range, the difference of the largest and the smallest value. You can either just use the function `range`, which requires the vector in question as its only argument, and then compute the difference from the two values with `diff`, or you just compute the range from the minimum and maximum yourself:

```
> range(LENGTH)¶
[1]  251 1600
> diff(range(LENGTH))¶
[1] 1349
> max(LENGTH)-min(LENGTH)¶
[1] 1349
```

This measure is extremely simple to compute but obviously also very sensitive: one outlier is enough to yield results that are not particularly meaningful anymore. For this reason, the range is not used very often.

1.3.3. Quantiles and quartiles

Another simple but useful and flexible measure of dispersion involves the quantiles of a distribution. We have met quantiles before in the context of probability distributions in Section 1.3.4. Theoretically, you compute quantiles by sorting the values in ascending order and then counting which values delimit the lowest *x*%, *y*%, etc. of the data; when these percentages are 25%, 50%, and 75%, then they are called quartiles. In R you can use the function `quantile`, (see below on `type=1`):

```
> a<-1:100¶
> quantile(a, type=1)¶
  0%  25%  50%  75% 100%
   1   25   50   75  100
```

If you write the integers from 1 to 100 next to each other, then 25 is the value that cuts off the lower 25%, etc. The value for 50% corresponds to the median, and the values for 0% and 100% are the minimum and the maximum. Let me briefly mention two arguments of this function. First, the argument `probs` allows you to specify other percentages. Second, the argument `type=…` allows you to choose other ways in which quantiles are computed. For discrete distributions, `type=1` is probably best, for continuous variables the default setting `type=7` is best.

```
> quantile(a, probs=c(0.05, 0.1, 0.5, 0.9, 0.95), type=1)¶
 5% 10% 50% 90% 95%
  5  10  50  90  95
```

The bottom line of using quantiles as a measure of dispersion of course is that the more the 25% quartile and the 75% quartile differ from each other, the more heterogeneous the data are, which is confirmed by looking at the data for the two towns: the so-called interquartile range – the difference between the 75% quartile and the 25% quartile – is much larger for Town 1 than for Town 2.

```
> quantile(town1)¶
   0%   25%   50%   75%  100%
-12.0   4.0  13.5  18.5  23.0
> IQR(town1)¶
[1] 14.5
> quantile(town2)¶
   0%   25%   50%   75%  100%
 6.00  7.75  9.00 11.25 16.00
> IQR(town2)¶
```

```
[1] 3.5
```

You can now apply this function to the lengths of the disfluencies:

```
> quantile(LENGTH, probs=c(0.2, 0.4, 0.5, 0.6, 0.8, 1),
   type=1)¶
 20%  40%  50%  60%  80% 100%
 519  788  897 1039 1307 1600
```

That is, the central 20% of all the lengths of disfluencies are greater than 788 and range up to 1039 (as you can verify with `sort(LENGTH)[401:600]¶`), 20% of the lengths are smaller than or equal to 519, 20% of the values are 1307 or larger, etc.

An interesting application of `quantile` is to use it to split vectors of continuous variables up into groups. For example, if you wanted to split the vector `LENGTH` into five groups of nearly equal ranges of values, you can use the function `cut` from Section 2.4.1 again, which splits up vectors into groups, and the function `quantile`, which tells `cut` what the groups should look like. That is, there are 200 values of `LENGTH` between and including 251 and 521 etc.

```
> LENGTH.GRP<-cut(LENGTH, breaks=quantile(LENGTH, probs=
   c(0, 0.2, 0.4, 0.6, 0.8, 1)), include.lowest=TRUE)¶
> table(LENGTH.GRP)¶
LENGTH.GRP
          [251,521]           (521,789]      (789,1.04e+03]
                200                 200                 200
(1.04e+03,1.31e+03]  (1.31e+03,1.6e+03]
                203                 197
```

1.3.4. The average deviation

Another way to characterize the dispersion of a distribution is the average deviation. You compute the absolute difference of every data point from the mean of the distribution (cf. `abs`), and then you compute the mean of these absolute differences. For Town 1, the average deviation is 9.04:

```
> town1¶
 [1]  -5 -12   5  12  15  18  22  23  20  16   8   1
> town1-mean(town1)¶
 [1] -15.25 -22.25  -5.25   1.75   4.75   7.75  11.75
   12.75  9.75   5.75  -2.25  -9.25
> abs(town1-mean(town1))¶
 [1] 15.25 22.25  5.25  1.75  4.75  7.75 11.75 12.75
```

```
   9.75  5.75  2.25  9.25
> mean(abs(town1-mean(town1)))¶
[1] 9.041667
> mean(abs(town2-mean(town2)))¶
[1] 2.472222
```

For the lengths of the disfluencies, we obtain:

```
> mean(abs(LENGTH-mean(LENGTH)))¶
[1] 329.2946
```

Although this is a quite intuitive measure, it is unfortunately hardly used anymore. For better or for worse (cf. Gorard 2004), you will more often find the dispersion measure discussed next, the standard deviation.

1.3.5. The standard deviation/variance

The standard deviation *sd* of a distribution *x* with *n* elements is defined in (17). This may look difficult at first, but the standard deviation is conceptually similar to the average deviation. For the average deviation, you compute the difference of each data point to the mean and take its absolute value – for the standard deviation you compute the difference of each data point to the mean, square these differences, sum them up, and after dividing the sum by *n*-1, you take the square root (to 'undo' the previous squaring).

$$(17) \qquad sd = \left(\frac{\sum_{i=1}^{n} \left(x_i - \bar{x} \right)^2}{n-1} \right)^{\frac{1}{2}}$$

Once we 'translate' this into R, it probably becomes clearer:

```
> town1¶
 [1]  -5 -12   5  12  15  18  22  23  20  16   8   1
> town1-mean(town1)¶
 [1] -15.25 -22.25  -5.25   1.75   4.75   7.75  11.75
  12.75   9.75   5.75  -2.25  -9.25
> (town1-mean(town1))^2¶
 [1] 232.5625 495.0625  27.5625   3.0625  22.5625  60.0625
  138.0625 162.5625  95.0625  33.0625   5.0625  85.5625
> sum((town1-mean(town1))^2)¶
[1] 1360.25
> sum((town1-mean(town1))^2)/(length(town1)-1)¶
```

```
[1] 123.6591
> sqrt(sum((town1-mean(town1))^2)/(length(town1)-1))¶
[1] 11.12021
```

There is of course an easier way …

```
> sd(town1); sd(town2)¶
[1] 11.12021
[1] 3.157483
```

Note in passing: the standard deviation is the square root of another measure, the *variance*, which you can also compute with the function `var`.

Recommendation(s) for further study
the function `mad` to compute another very robust measure of dispersion, the median absolute deviation

1.3.6. The variation coefficient

Even though the standard deviation is probably the most widespread measure of dispersion, it has a potential weakness: its size is dependent on the mean of the distribution, as you can see in the following example:

```
> sd(town1)¶
[1] 11.12021
> sd(town1*10)¶
[1] 111.2021
```

When the values, and hence the mean, is increased by one order of magnitude, then so is the standard deviation. You can therefore not compare standard deviations from distributions with different means if you do not first normalize them. If you divide the standard deviation of a distribution by its mean, you get the variation coefficient. You see that the variation coefficient is not affected by the multiplication with 10, and Town 1 still has a larger degree of dispersion.

```
> sd(town1)/mean(town1)¶
[1] 1.084899
> sd(town1*10)/mean(town1*10)¶
[1] 1.084899
> sd(town2)/mean(town2)¶
[1] 0.3210999
```

1.3.7. Summary functions

If you want to obtain several summarizing statistics for a vector (or a factor), you can use `summary`, whose output is self-explanatory.

```
> summary(town1)¶
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -12.00    4.00   13.50   10.25   18.50   23.00
```

An immensely useful graph is the so-called boxplot. In its simplest form, the function `boxplot` just requires one vector as an argument, but we also add `notch=TRUE`, which I will explain shortly, as well as a line that adds little plus signs for the arithmetic means. Note that I am assigning the output of `boxplot` to a data structure called `boxsum` for later inspection.

```
> boxsum<-boxplot(town1, town2, notch=TRUE,
   names=c("Town 1", "Town 2"))¶
> text(1:2, c(mean(town1), mean(town2)), c("+", "+"))¶
```

This plot, see Figure 28, contains a lot of valuable information:

- the bold-typed horizontal lines represent the medians of the two vectors;
- the regular horizontal lines that make up the upper and lower boundary of the boxes represent the hinges (approximately the 75%- and the 25% quartiles);

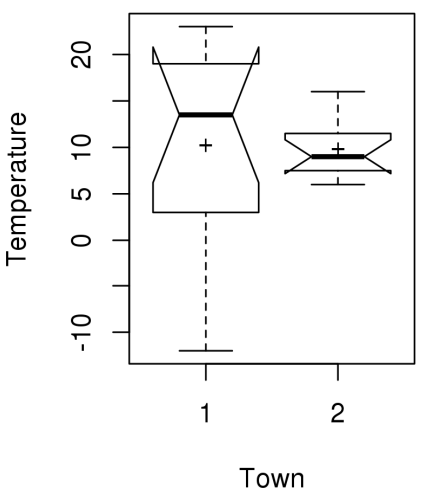

Figure 28. Boxplot of the temperatures of the two towns

– the whiskers – the dashed vertical lines extending from the box until the upper and lower limit – represent the largest and smallest values that are not more than 1.5 interquartile ranges away from the box;
– each data point that would be outside of the range of the whiskers would be represented as an outlier with an individual small circle;
– the notches on the left and right sides of the boxes extend across the range `±1.58*IQR/sqrt(n)`: if the notches of two boxplots do not overlap, then their medians will most likely be significantly different.

Figure 28 shows that the average temperatures of the two towns are very similar and probably not significantly different from each other. Also, the dispersion of Town 1 is much larger than that of Town 2. Sometimes, a good boxplot nearly obviates the need for further analysis; boxplots are extremely useful and will often be used in the chapters to follow. However, there are situations where the ecdf plot introduced above is better and the following example is modeled after what happened in a real dataset of a student I supervised. Run the code in the code file and consider Figure 29.

As you could see in the code file, I created a vector `x1` that actually contains data from two very different distributions whereas the vector `x2` contains data from only one but wider distribution.

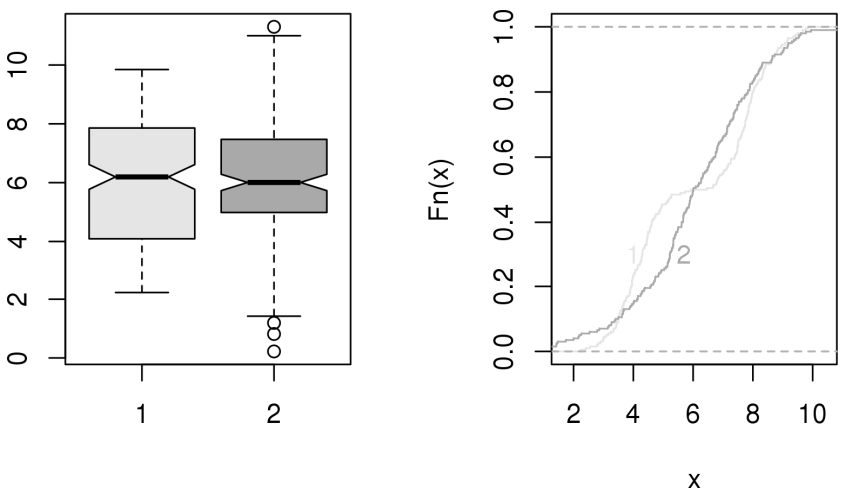

Figure 29. Boxplots (left panel) and ecdf plots (right panel) of two vectors

Crucially, the boxplots do not reveal that at all. Yes, the second darker boxplot is wider and has some outliers but the fact that the first lighter box-

plot represents a vector containing data from two different distributions is completely absent from the graph. The ecdf plots in the right panel show that very clearly, however: the darker line for the second vector increases steadily in a way that suggests one normal distribution whereas the lighter line for the first vector shows that it contains two normal distributions, given the two *s*-shaped curve segments. Thus, while the ecdf plot is not as intuitively understandable as a boxplot, it can be much more informative.

Recommendation(s) for further study
the functions `hdr.boxplot` (from the library `hdrcde`), `vioplot` (from the library `vioplot`), and `bpplot` (from the library `Hmisc`) for interesting alternatives to, or extensions of, boxplots

1.3.8. The standard error

The standard error of an arithmetic mean is defined as the standard deviation of the means of equally large samples drawn randomly from a population with replacement. Imagine you took a sample from a population and computed the arithmetic mean of some variable. Unless your sample is *perfectly* representative of the population, this mean will not correspond exactly to the arithmetic mean of that variable in the population, and it will also not correspond exactly to the arithmetic mean you would get from another equally large sample from the same population. If you take many (e.g., 10,000) random and equally large samples from the population with replacement and computed the arithmetic mean of each of them, then the standard deviation of all these means is the standard error.

```
> means<-vector(length=10000)¶
> for (i in 1:10000) {¶
+    means[i]<-mean(sample(LENGTH, size=1000, replace=TRUE))¶
+ }¶
> sd(means)¶
[1] 12.10577
```

The standard error of an arithmetic mean is computed according to the formula in (18), and from (18) you can already see that the larger the standard error of a mean, the smaller the likelihood that that mean is a good estimate of the population mean, and that the larger sample size *n*, the smaller the standard error becomes:

(18) $$se_{\text{mean}} = \sqrt{\frac{var}{n}} = \frac{sd}{\sqrt{n}}$$

Thus, the standard error of the mean length of disfluencies here is this, which is very close to our resampled result from above.

```
> mean(LENGTH)¶
[1] 915.043
> sqrt(var(LENGTH)/length(LENGTH))¶
[1] 12.08127
```

You can also compute standard errors for statistics other than arithmetic means but the only other example we look at here is the standard error of a relative frequency *p*, which is computed according to the formula in (19):

(19) $$se_{\text{percentage}} = \sqrt{\frac{p \cdot (1-p)}{n}}$$

Thus, the standard error of the percentage of all silent disfluencies out of all disfluencies (33.2% of 1000 disfluencies) is:

```
> prop.table(table(FILLER))¶
FILLER
silence      uh     uhm
  0.332   0.394   0.274
> sqrt(0.332*(1-0.332)/1000)¶
[1] 0.01489215
```

Standard errors will be much more important in Section 3.1.5 because they are used to compute so-called confidence intervals. Note that when you compare means of two roughly equally large samples and their intervals means±standard errors overlap, then you know the sample means are not significantly different. However, if these intervals do not overlap, this does not show that the means are significantly different (cf. Crawley 2005: 169f.). In Chapter 5, you will also get to see standard errors of differences of means, which are computed according to the formula in (20).

(20) $$se_{\text{difference between means}} = \sqrt{SE_{mean\_group1}^{\ 2} + SE_{mean\_group2}^{\ 2}}$$

Warning/advice
Standard errors are only really useful if the data to which they are applied are distributed pretty normally or when the sample size $n \geq 30$.

1.4. Centering and standardization (*z*-scores)

Very often it is useful or even necessary to compare values coming from different scales. An example (from Bortz 2005): if a student X scored 80% in a course and a student Y scored 60% in another course, can you then say that student X was better than student Y? On the one hand, sure you can: 80% is better than 60%. On the other hand, the test in which student Y participated could have been much more difficult than the one in which student X participated. It can therefore be useful to relativize/normalize the individual grades of the two students on the basis of the overall performance of students in their courses. (You encountered a similar situation above in Section 3.1.3.6 when you learned that it is not always appropriate to compare different standard deviations directly.) Let us assume the grades obtained in the two courses look as follows:

```
> grades.course.X<-rep((seq(0, 100, 20)), 1:6);
   grades.course.X¶
[1]   0  20  20  40  40  40  60  60  60  60  80  80  80  80
   80 100 100 100 100 100 100
> grades.course.Y<-rep((seq(0, 100, 20)), 6:1);
   grades.course.Y¶
[1]   0   0   0   0   0   0  20  20  20  20  20  40  40  40
   40  60  60  60  80  80 100
```

One way to normalize the grades is called *centering* and simply involves subtracting from each individual value within one course the average of that course.

```
> a<-1:5¶
> centered.scores<-a-mean(a); centered.scores¶
[1] -2 -1  0  1  2
```

You can see how these scores relate to the original values in `a`: since the mean of `a` is obviously 3, the first two centered scores are negative (i.e., smaller than `a`'s mean), the third is 0 (it does not deviate from `a`'s mean), and the last two centered scores are positive (i.e., larger than `a`'s mean).

Another more sophisticated way involves *standardizing*, i.e. trans-

forming the values to be compared into so-called *z*-scores, which indicate how many standard deviations each value of the vector deviates from the mean of the vector. The *z*-score of a value from a vector is the difference of that value from the mean of the vector, divided by the vector's standard deviation. You can compute that manually as in this simple example:

```
> z.scores<-(a-mean(a))/sd(a); z.scores¶
[1] -1.2649111 -0.6324555  0.0000000  0.6324555  1.2649111
```

The relationship between the *z*-scores and `a`'s original values is very similar to that between the centered scores and `a`'s values: since the mean of `a` is obviously 3, the first two *z*-scores are negative (i.e., smaller than `a`'s mean), the third *z*-score is 0 (it does not deviate from `a`'s mean), and the last two *z*-scores are positive (i.e., larger than `a`'s mean). Note that such *z*-scores have a mean of 0 and a standard deviation of 1:

```
> mean(z.scores)¶
[1] 0
> sd(z.scores)¶
[1] 1
```

Both normalizations can be performed with the function `scale`, which takes three arguments: the vector to be normalized, `center=…` (the default is `TRUE`) and `scale=…` (the default is `TRUE`). If you do not provide any arguments other than the vector to be standardized, then `scale`'s default setting returns a matrix that contains the *z*-scores and whose attributes correspond to the mean and the standard deviation of the vector:

```
> scale(a)¶
            [,1]
[1,] -1.2649111
[2,] -0.6324555
[3,]  0.0000000
[4,]  0.6324555
[5,]  1.2649111
attr(,"scaled:center")
[1] 3
attr(,"scaled:scale")
[1] 1.581139
```

If you set `scale` to `FALSE`, then you get centered scores:

```
> scale(a, scale=FALSE)¶
     [,1]
[1,]   -2
```

```
[2,]   -1
[3,]    0
[4,]    1
[5,]    2
attr(,"scaled:center")
[1] 3
```

If we apply both versions to our example with the two courses, then you see that the 80% scored by student X is only 0.436 standard deviations (and 13.33 percent points) better than the mean of his course whereas the 60% scored by student Y is actually 0.873 standard deviations (and 26.67 percent points) above the mean of his course. Thus, X's score is higher than Y's, but if we take the overall results in the two courses into consideration, then Y's performance is better; standardizing data is often useful.

1.5. Confidence intervals

In most cases, you are not able to investigate the whole population you are actually interested in because that population is not accessible and/or too large so investigating it is impossible, too time-consuming, or too expensive. However, even though you know that different samples will yield different statistics, you of course hope that your sample would yield a reliable estimate that tells you much about the population you are interested in:

- if you find in your sample of 1000 disfluencies that their average length is approximately 915 ms, then you hope that you can generalize from that to the population and future investigations;
- if you find in your sample of 1000 disfluencies that 33.2% of these are silences, then you hope that you can generalize from that to the population and future investigations.

So far, we have only discussed how you can compute percentages and means for samples – the question of how valid these are for populations is the topic of this section. In Section 3.1.5.1, I explain how you can compute confidence intervals for arithmetic means, and Section 3.1.5.2 explains how to compute confidence intervals for percentages. The relevance of such confidence intervals must not be underestimated: without a confidence interval it is unclear how well you can generalize from a sample to a population; apart from the statistics we discuss here, one can also compute confidence intervals for many others.

1.5.1. Confidence intervals of arithmetic means

If you compute a mean on the basis of a sample, you of course hope that it represents that of the population well. As you know, the average length of disfluencies in our example data is 915.043 ms (standard deviation: 382.04). But as we said above, other samples' means will be different so you would ideally want to quantify your confidence in this estimate. The so-called confidence interval, which is useful to provide with your mean, is the interval of values around the sample mean around which we will assume there is no significant difference with the sample mean. From the expression "significant difference", it follows that a confidence interval is typically defined as 1-significance level, i.e., typically as 1-0.05 = 0.95.

In a first step, you again compute the standard error of the arithmetic mean according to the formula in (18).

```
> se<-sqrt(var(LENGTH)/length(LENGTH)); se¶
[1] 12.08127
```

This standard error is used in (21) to compute the confidence interval. The parameter *t* in formula (21) refers to the distribution mentioned in Section 1.3.4.3, and its computation requires the number of degrees of freedom. In this case, the number of degrees of freedom *df* is the length of the vector-1, i.e. 999. Since you want to compute a *t*-value on the basis of a *p*-value, you need the function `qt`, and since you want a two-tailed interval – 95% of the values around the observed mean, i.e. values larger and smaller than the mean – you must compute the *t*-value for 2.5% (because 2.5% on both sides result in the desired 5%):

(21) $CI = \bar{x} \pm t \cdot SE$

```
> t.value<-qt(0.025, df=999, lower.tail=FALSE); t.value¶
[1] 1.962341
```

Now you can compute the confidence interval:

```
> mean(LENGTH)-(se*t.value); mean(LENGTH)+(se*t.value)¶
[1] 891.3354
[1] 938.7506
```

To do this more simply, you can use the function `t.test` with the relevant vector and use `conf.level=…` to define the relevant percentage. R then

computes a significance test the details of which are not relevant yet, which is why we only look at the confidence interval (with `$conf.int`):

```
> t.test(LENGTH, conf.level=0.95)$conf.int¶
[1] 891.3354 938.7506
attr(,"conf.level")
[1] 0.95
```

This confidence interval

> identifies a range of values a researcher can be 95% confident contains the true value of a population parameter (e.g., a population mean). Stated in probabilistic terms, the researcher can state there is a probability/likelihood of .95 that the confidence interval contains the true value of the population parameter. (Sheskin 2011:75; see also Field, Miles, and Field 2012:45)[14]

Note that when you compare means of two roughly equally large samples and their 95%-confidence intervals do not overlap, then you know the sample means are significantly different and, therefore, you would assume that there is a real difference between the population means, too. However, if these intervals do overlap, this does not show that the means are not significantly different from each other (cf. Crawley 2005: 169f.).

1.5.2. Confidence intervals of percentages

The above logic with regard to means also applies to percentages. Given a particular percentage from a sample, you want to know what the corresponding percentage in the population is. As you already know, the percentage of silent disfluencies in our sample is 33.2%. Again, you would like to quantify your confidence in that sample percentage. As above, you compute the standard error for percentages according to the formula in (19), and then this standard error is inserted into the formula in (22).

14 A different way of explaining confidence intervals is this: "A common error is to misinterpret the confidence interval as a statement about the unknown parameter [here, the percentage in the population, STG]. It is not true that the probability that a parameter is included in a 95% confidence interval is 95%. What is true is that if we derive a large number of 95% confidence intervals, we can expect the true value of the parameter to be included in the computed intervals 95% of the time" (Good and Hardin 2012:156)

```
> se<-sqrt(0.332*(1-0.332)/1000); se¶
[1] 0.01489215
```

(22) $CI = a \pm z \cdot SE$

The parameter z in (22) corresponds to the z-score mentioned above in Section 1.3.4.3, which defines 5% of the area under a standard normal distribution – 2.5% from the upper part and 2.5% from the lower part:

```
> z.score<-qnorm(0.025, lower.tail=FALSE); z.score¶
[1] 1.959964
```

For a 95% confidence interval for the percentage of silences, you enter:

```
> z.score<-qnorm(0.025, lower.tail=FALSE)¶
> 0.332-z.score*se; 0.332+z.score*se¶
[1] 0.3028119
[1] 0.3611881
```

The simpler way requires the function `prop.test`, which tests whether a percentage obtained in a sample is significantly different from an expected percentage. Again, the functionality of that significance test is not relevant yet, but this function also returns the confidence interval for the observed percentage. R needs the observed frequency (332), the sample size (1000), and the probability for the confidence interval. R uses a formula different from ours but returns nearly the same result.

```
> prop.test(332, 1000, conf.level=0.95)$conf.int¶
[1] 0.3030166 0.3622912
attr(,"conf.level")
[1] 0.95
```

Recommendation(s) for further study
Dalgaard (2002: Ch. 7.1 and 4.1), Crawley (2005: 167ff.)

Warning/advice
Since confidence intervals are based on standard errors, the warning from above applies here, too: if data are not normally distributed or the samples too small, then you should probably use other methods to estimate confidence intervals (e.g., bootstrapping).

2. Bivariate statistics

We have so far dealt with statistics and graphs that describe one variable or vector/factor. In this section, we now turn to methods to characterize two variables and their relation. We will again begin with frequencies, then we will discuss means, and finally talk about correlations. You will see that we can use many functions from the previous sections.

2.1. Frequencies and crosstabulation

We begin with the case of two nominal/categorical variables. Usually, one wants to know which combinations of variable levels occur how often. The simplest way to do this is cross-tabulation. Let's return to the disfluencies:

```
> UHM<-read.delim(file.choose())¶
> attach(UHM)¶
```

Let's assume you wanted to see whether men and women differ with regard to the kind of disfluencies they produce. First two questions: are there dependent and independent variables in this design and, if so, which?

THINK
BREAK

In this case, SEX is the independent variable and FILLER is the dependent variable. Computing the frequencies of variable level combinations in R is easy because you can use the same function that you use to compute frequencies of an individual variable's levels: `table`. You just give `table` a second vector or factor as an argument and R lists the levels of the first vector in the rows and the levels of the second in the columns:

```
> freqs<-table(FILLER, SEX); freqs¶
         SEX
FILLER    female male
  silence    171  161
  uh         161  233
  uhm        170  104
```

In fact you can provide even more vectors to `table`, just try it out, and

we will return to this below. Again, you can create tables of percentages with `prop.table`, but with two-dimensional tables there are different ways to compute percentages and you can specify one with `margin=`…. The default is `margin=NULL`, which computes the percentages on the basis of all elements in the table. In other words, all percentages in the table add up to 1. Another possibility is to compute row percentages: set `margin=1` and you get percentages that add up to 1 in every row. Finally, you can choose column percentages by setting `margin=2`: the percentages in each column add up to 1. This is probably the best way here since then the percentages adding up to 1 are those of the dependent variable.

```
> percents<-prop.table(table(FILLER, SEX), margin=2)¶
> percents¶
         SEX
FILLER       female      male
  silence 0.3406375 0.3232932
  uh      0.3207171 0.4678715
  uhm     0.3386454 0.2088353
```

You can immediately see that men appear to prefer *uh* and disprefer *uhm* while women appear to have no real preference for any disfluency. However, we of course do not know yet whether this is a significant result.

The function `addmargins` outputs row and column totals (or other user-defined margins, such as means):

```
> addmargins(freqs) # cf. also colSums and rowSums¶
         SEX
FILLER    female male  Sum
  silence    171  161  332
  uh         161  233  394
  uhm        170  104  274
  Sum        502  498 1000
```

Recommendation(s) for further study
the functions `xtabs` and especially `ftable` to generate more complex tables

2.1.1. Bar plots and mosaic plots

Of course you can also represent such tables graphically. The simplest way involves providing a formula as the main argument to `plot`. Such formulae consist of a dependent variable (here: FILLER: `FILLER`), a tilde ("~" meaning 'as a function of'), and an independent variable (here: GENRE: `GENRE`).

```
> plot(FILLER~GENRE)¶
```

The widths and heights of rows, columns, and the six boxes represent the observed frequencies. For example, the column for dialogs is a little wider than that for monologs because there are more dialogs in the data; the row for *uh* is widest because *uh* is the most frequent disfluency, etc.

Other similar graphs can be generated with the following lines:

```
> plot(GENRE, FILLER)¶
> plot(table(GENRE, FILLER))¶
> mosaicplot(table(GENRE, FILLER))¶
```

These graphs are called stacked bar plots or mosaic plots and are – together with association plots to be introduced below – often effective ways of representing crosstabulated data. In the code file for this chapter you will find R code for another kind of useful graph.

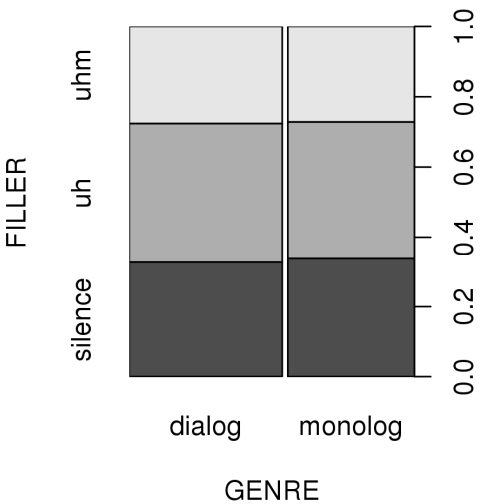

Figure 30. Stacked bar plot / mosaic plot for FILLER~GENRE

2.1.2. Spineplots

Sometimes, the dependent variable is nominal/categorical and the independent variable is interval/ratio-scaled. Let us assume that FILLER is the dependent variable, which is influenced by the independent variable LENGTH. (This does not make much sense here, we just do this for exposi-

tory purposes.) You can use the function `spineplot` with a formula:

```
> spineplot(FILLER~LENGTH)¶
```

The *y*-axis represents the dependent variable and its three levels. The *x*-axis represents the independent ratio-scaled variable, which is split up into the value ranges that would also result from `hist` (which also means you can change the ranges with `breaks=…`; cf. Section 3.1.1.5 above).

2.1.3. Line plots

Apart from these plots, you can also generate line plots that summarize frequencies. If you generate a table of relative frequencies, then you can create a primitive line plot by entering the code shown below.

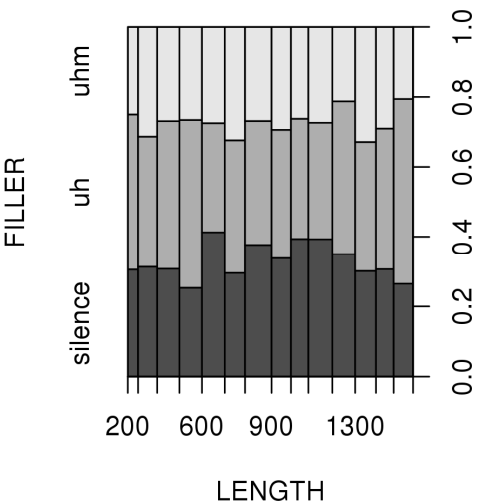

Figure 31. Spineplot for FILLER~LENGTH

```
> fill.table<-prop.table(table(FILLER, SEX), 2); fill.table¶
          SEX
FILLER       female      male
  silence 0.3406375 0.3232932
  uh      0.3207171 0.4678715
  uhm     0.3386454 0.2088353
> plot(fil.table[,1], ylim=c(0, 0.5), xlab="Disfluency",
   ylab="Relative frequency", type="b")¶
> points(fil.table[,2], type="b")¶
```

However, somewhat more advanced code in the companion file shows you how you can generate the graph in Figure 32. (Again, you may not understand the code immediately, but it will not take you long.)

Warning/advice
Sometimes, it is recommended to not represent such frequency data with a line plot like this because the lines 'suggest' that there are frequency values between the levels of the categorical variable, which is of course not the case. Again, you should definitely explore the function `dotchart` for this.

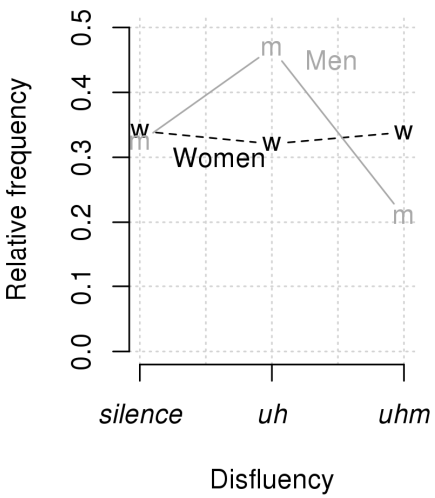

Figure 32. Line plot with the percentages of the interaction of SEX and FILLER

Recommendation(s) for further study
the function `plotmeans` (from the library `gplots`) to plot line plots with means and confidence intervals

2.2. Means

If the dependent variable is interval/ratio-scaled or ordinal and the independent variable is nominal/categorical, then one is often not interested in the frequencies of particular values of the dependent variable, but its central tendencies at each level of the independent variable. For example, you might want to determine whether men and women differ with regard to the average disfluency lengths. One way to get these means is the following:

```
> mean(LENGTH[SEX=="female"])¶
[1] 928.3984
> mean(LENGTH[SEX=="male"])¶
[1] 901.5803
```

This approach is too primitive for three reasons:

- you must define the values of LENGTH that you want to include manually, which requires a lot of typing (especially when the independent variable has more than two levels or, even worse, when you have more than one independent variable);
- you must know all relevant levels of the independent variables – otherwise you couldn't use them for subsetting in the first place;
- you only get the means of the variable levels you have explicitly asked for. However, if, for example, you made a coding mistake in one row – such as entering "malle" instead of "male" – this approach will not show you that.

Thus, we use an extremely useful function called `tapply`, which mostly takes three arguments. The first is a vector or factor to which you want to apply a function – here, this is LENGTH, to which we want to apply `mean`. The second argument is a vector or factor that has as many elements as the first one and that specifies the groups of values from the first vector/factor to which the function is to be applied. The last argument is the relevant function, here `mean`. We get:

```
> tapply(LENGTH, SEX, mean)¶
  female     male
928.3984 901.5803
```

Of course the result is the same as above, but you obtained it in a better way. You can of course use functions other than `mean`: `median`, `IQR`, `sd`, `var`, …, even functions you wrote yourself. For example, what do you get when you use `length`? The numbers of lengths observed for each sex.

2.2.1. Boxplots

In Section 3.1.3.7 above, we looked at boxplots, but restricted our attention to cases where we have one or more dependent variables (such as `town1` and `town2`). However, you can also use boxplots for cases where you have

one or more independent variables and a dependent variable. Again, the easiest way is to use a formula with the tilde meaning 'as a function of':

```
> boxplot(LENGTH~GENRE, notch=TRUE, ylim=c(0, 1600))¶
```

(If you only want to plot a boxplot and not provide any further arguments, it is actually enough to just enter `plot(LENGTH~GENRE)¶`: R 'infers' you want a boxplot because `LENGTH` is a numerical vector and `GENRE` is a factor.) Again, you can infer a lot from that plot: both medians are close to 900 ms and do most likely not differ significantly from each other (since the notches overlap). Both genres appear to have about the same amount of dispersion since the notches, the boxes, and the whiskers are nearly equally large, and both genres have no outliers.

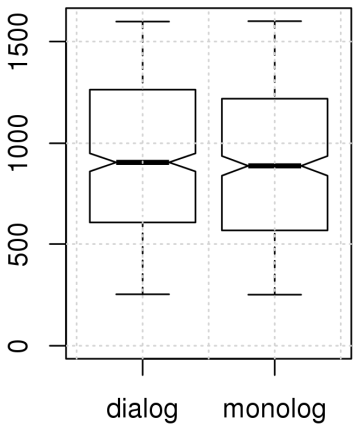

Figure 33. Boxplot for LENGTH~GENRE

Quick question: can you infer what this line does?

```
> text(seq(levels(GENRE)), tapply(LENGTH, GENRE, mean), "+")¶
```

THINK BREAK

It adds plusses into the boxplot representing the means of `LENGTH` for each `GENRE`: `seq(levels(GENRE))` returns 1:2, which is used as the *x*-

coordinates; the `tapply` code returns the means of `LENGTH` for each `GENRE`, and the "+" is what is plotted.

2.2.2. Interaction plots

So far we have looked at graphs representing one variable or one variable depending on another variable. However, there are also cases where you want to characterize the distribution of one interval/ratio-scaled variable depending on two, say, nominal/categorical variables. You can again obtain the means of the variable level combinations of the independent variables with `tapply`. You must specify the two independent variables in the form of a list, and the following two examples show you how you get the same means in two different ways (so that you see which variable goes into the rows and which into the columns):

```
> tapply(LENGTH, list(SEX, FILLER), mean)¶
          silence       uh      uhm
female 942.3333 940.5652 902.8588
male   891.6894 904.9785 909.2788
> tapply(LENGTH, list(FILLER, SEX), mean)¶
            female     male
silence 942.3333 891.6894
uh      940.5652 904.9785
uhm     902.8588 909.2788
```

Such results are best shown in tabular form such that you don't just provide the above means of the interactions as they were represented in Figure 32 above, but also the means of the individual variables. Consider Table 17 and the formula in its caption exemplifying the relevant R syntax.

Table 17. Means for LENGTH ~ FILLER * SEX

	SEX: *FEMALE*	SEX: *MALE*	Total
FILLER: *SILENCE*	942.33	891.69	917.77
FILLER: *UH*	940.57	904.98	919.52
FILLER: *UHM*	902.86	909.28	905.3
TOTAL	928.4	901.58	915.04

A plus sign between variables refers to just adding *main effects* of variables (i.e., effects of variables *in isolation*, e.g. when you only inspect the two means for SEX in the bottom row of totals or the three means for FILLER in the rightmost column of totals). A colon between variables refers

to only the interaction of the variables (i.e., effects of combinations of variables as when you inspect the six means in the main body of the table where SEX and FILLER are *combined*). Finally, an asterisk between variables denotes both the main effects *and* the interaction (here, all 12 means). With two variables `A` and `B`, `A*B` is the same as `A + B + A:B`.

Now to the results. These are often easier to understand when they are represented graphically. You can create and configure an interaction plot manually, but for a quick and dirty glance at the data, you can also use the function `interaction.plot`. As you might expect, this function takes at least three arguments:

- `x.factor`: a vector/factor whose values/levels are represented on the *x*-axis;
- `trace.factor`: the second argument is a vector/factor whose values/levels are represented with different lines;
- `response`: the third argument is a vector whose means for all variable level combinations will be represented on the *y*-axis by the lines.

That means, you can choose one of two formats, depending on which independent variable is shown on the *x*-axis and which is shown with different lines. While the represented means will of course be identical, I advise you to *always* generate and inspect both graphs anyway because one of the two graphs is usually easier to interpret. In Figure 34, you find both graphs for the above values and I prefer the lower panel.

```
> interaction.plot(FILLER, SEX, LENGTH); grid()¶
> interaction.plot(SEX, FILLER, LENGTH); grid()¶
```

Obviously, *uhm* behaves differently from *uh* and silences: the average lengths of women's *uh* and silence are larger than those of men, but the average length of women's *uhm* is smaller than that of men. But now an important question: why should you now not just report the means you computed with `tapply` and the graphs in Figure 34 in your study?

THINK
BREAK

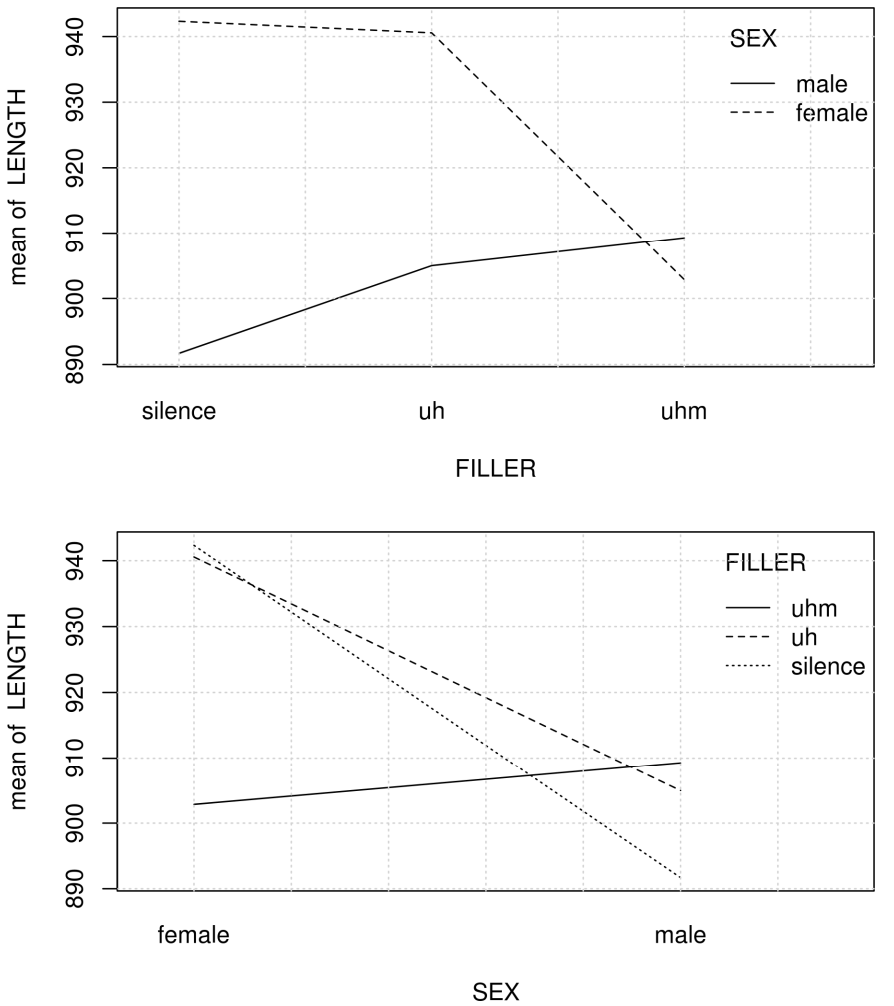

Figure 34. Interaction plot for LENGTH ~ FILLER : SEX

First, you should not just report the means like this because I told you to never ever report means without a measure of dispersion. Thus, when you want to provide the means, you must also add, say, standard deviations, standard errors, confidence intervals:

```
> tapply(LENGTH, list(SEX, FILLER), sd)¶
        silence       uh      uhm
female 361.9081 397.4948 378.8790
male   370.6995 397.1380 382.3137
```

How do you get the standard errors and the confidence intervals?

THINK
BREAK

```
> se<-tapply(LENGTH, list(SEX, FILLER), sd)/
   sqrt(tapply(LENGTH, list(SEX, FILLER), length)); se¶
         silence       uh      uhm
female 27.67581 31.32698 29.05869
male   29.21522 26.01738 37.48895

> t.value<-qt(0.025, df=999, lower.tail=FALSE); t.value¶
[1] 1.962341
> tapply(LENGTH, list(SEX, FILLER), mean)-(t.value*se)¶
         silence       uh      uhm
female 888.0240 879.0910 845.8357
male   834.3592 853.9236 835.7127
> tapply(LENGTH, list(SEX, FILLER), mean)+(t.value*se)¶
         silence        uh     uhm
female 996.6427 1002.0394 959.882
male   949.0197  956.0335 982.845
```

And this output immediately shows again why measures of dispersion are important: the standard deviations are large and the means plus/minus one standard error overlap (as do the confidence intervals), which shows that the differences are not significant. You can see this with `boxplot`, which allows formulae with more than one independent variable (`boxplot(LENGTH~SEX*FILLER, notch=TRUE)¶`, with an asterisk for the interaction).

Second, the graphs should not be used as they are (at least not uncritically) because R has chosen the range of the *y*-axis such that it is as small as possible but still covers all necessary data points. However, this small range on the *y*-axis has visually inflated the differences in Figure 34 – a more realistic representation would have either included the value $y = 0$ (as in the first pair of the following four lines) or chosen the range of the *y*-axis such that the complete range of LENGTH is included (as in the second pair of the following four lines):

```
> interaction.plot(SEX, FILLER, LENGTH, ylim=c(0, 1000))¶
> interaction.plot(FILLER, SEX, LENGTH, ylim=c(0, 1000))¶
> interaction.plot(SEX, FILLER, LENGTH, ylim=range(LENGTH))¶
> interaction.plot(FILLER, SEX, LENGTH, ylim=range(LENGTH))¶
```

2.3. Coefficients of correlation and linear regression

The last section in this chapter is devoted to cases where both the dependent and the independent variable are ratio-scaled. For this scenario we turn to a new data set. First, we clear our memory of all data structures we have used so far:

```
> rm(list=ls(all=TRUE))¶
```

We look at data to determine whether there is a correlation between the reaction times in ms of second language learners in a lexical decision task and the length of the stimulus words. We have

- a dependent ratio-scaled variable: the reaction time in ms MS_LEARNER, whose correlation with the following independent variable we are interested in;
- an independent ratio-scaled variable: the length of the stimulus words LENGTH (in letters).

Such correlations are typically quantified using a so-called coefficient of correlation *r*. This coefficient, and many others, are defined to fall in the range between -1 and +1. Table 18 explains what the values mean: the sign of a correlation coefficient reflects the *direction* of the correlation, and the absolute size reflects the *strength* of the correlation. When the correlation coefficient is 0, then there is no correlation between the two variables in question, which is why H_0 says $r = 0$ – the two-tailed H_1 says $r \neq 0$.

Table 18. Correlation coefficients and their interpretation

Correlation coefficient	Labeling the correlation	Kind of correlation
$0.7 < r \leq 1$	very high	positive correlation:
$0.5 < r \leq 0.7$	high	the more/higher …, the more/higher …
$0.2 < r \leq 0.5$	intermediate	the less/lower …, the less/lower …
$0 < r \leq 0.2$	low	
$r \approx 0$	no statistical correlation (H_0)	
$0 > r \geq -0.2$	low	negative correlation:
$-0.2 > r \geq -0.5$	intermediate	the more/higher …, the less/lower …
$-0.5 > r \geq -0.7$	high	the less/lower …, the more/higher …
$-0.7 > r \geq -1$	very high	

Let us load and plot the data, using by now familiar lines of code:

```
> ReactTime<-read.delim(file.choose())¶
> str(ReactTime); attach(ReactTime)¶
'data.frame':   20 obs. of  3 variables:
 $ CASE      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ LENGTH    : int  14 12 11 12 5 9 8 11 9 11 ...
 $ MS_LEARNER: int  233 213 221 206 123 176 195 207 172 ...
> plot(MS_LEARNER~LENGTH, xlim=c(0, 15), ylim=c(0, 300),
   xlab="Word length in letters", ylab="Reaction time of
   learners in ms"); grid()¶
```

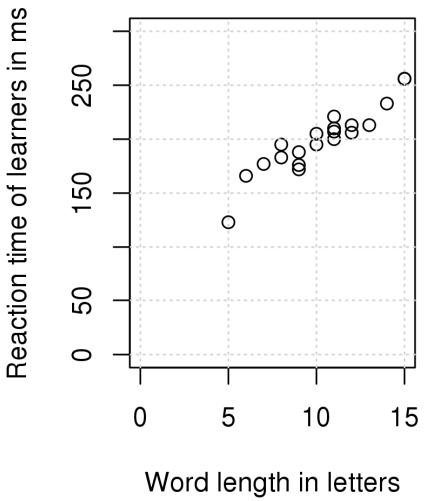

Figure 35. Scatterplot[15] for MS_LEARNER~LENGTH

What kind of correlation is that, a positive or a negative one?

THINK
BREAK

This is a positive correlation, because we can describe it with a "the more …, the more …" statement: the longer the word, the longer the reaction time: when you move from the left (short words) to the right (long words), the reaction times get higher. But we also want to quantify the correlation and compute the Pearson product-moment correlation *r*.

15 Check the code file for how to handle overlapping points.

First, we do this manually: We begin by computing the *covariance* of the two variables according to the formula in (23).

$$(23) \qquad Covariance_{x,\,y} = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\cdot\left(y_i - \bar{y}\right)}{n-1}$$

As you can see, the covariance involves computing the differences of each variable's value from the variable's mean. For example, when the *i*-th value of both the vector *x* and the vector *y* are above the averages of *x* and *y*, then this pair of *i*-th values will contribute a positive value to the covariance. In R, we can compute the covariance manually or with the function `cov`, which requires the two relevant vectors:

```
> covariance<-sum((LENGTH-mean(LENGTH))*(MS_LEARNER-
   mean(MS_LEARNER)))/(length(MS_LEARNER)-1)¶
> covariance<-cov(LENGTH, MS_LEARNER); covariance¶
[1] 79.28947
```

The sign of the covariance already indicates whether two variables are positively or negatively correlated; here it is positive. However, we cannot use the covariance to quantify the correlation between two vectors because its size depends on the scale of the two vectors: if you multiply both vectors with 10, the covariance becomes 100 times as large as before although the correlation as such has of course not changed:

```
> cov(MS_LEARNER*10, LENGTH*10)¶
[1] 7928.947
```

Therefore, we divide the covariance by the product of the standard deviations of the two vectors and obtain *r*. This is a very high positive correlation, *r* is close to the theoretical maximum of 1. In R, we can do all this more efficiently with the function `cor`. Its first two arguments are the two vectors in question, and the third specifies the desired kind of correlation:

```
> covariance/(sd(LENGTH)*sd(MS_LEARNER))¶
[1] 0.9337171
> cor(MS_LEARNER, LENGTH, method="pearson")¶
[1] 0.9337171
```

The correlation can be investigated more closely, though. We can try to

predict values of the dependent variable on the basis of the independent one. This method is called *linear regression*. In its simplest form, it involves trying to draw a straight line in such a way that it represents the scattercloud best. Here, *best* is defined as 'minimizing the sums of the squared vertical distances of the observed *y*-values (here: reaction times) and the predicted *y*-values reflected by the regression line.' That is, the regression line is drawn fairly directly through the scattercloud because then these deviations are smallest. It is defined by a regression equation with two parameters, an intercept *a* and a slope *b*. Without discussing the relevant formulae here, I immediately explain how to get these values with R. Using the formula notation you already know, you define and inspect a so-called linear model using the function `lm`:

```
> model<-lm(MS_LEARNER~LENGTH); model¶
Call:
lm(formula = MS_LEARNER ~ LENGTH)
Coefficients:
(Intercept)       LENGTH
      93.61         10.30
```

That is, the intercept – the *y*-value of the regression line at $x = 0$ – is 93.61, and the slope of the regression line is 10.3, which means that for every letter of a word the estimated reaction time increases by 10.3 ms. For example, our data do not contain a word with 16 letters, but since the correlation between the variables is so strong, we can come up with a good prediction for the reaction time such words might result in:

predicted reaction time	=	intercept	+	$b \cdot$ LENGTH
258.41	≈	93.61	+	10.3 · 16

```
> 93.61+10.3*16¶
[1] 258.41
```

(This prediction of the reaction time is of course overly simplistic as it neglects the large number of other factors that influence reaction times but within the current linear model this is how it would be computed.) Alternatively, you can use the function `predict`, whose first argument is the (linear) model and whose second argument can be a data frame called `newdata` that contains a column with values for each independent variable for which you want to make a prediction. With the exception of differences resulting from me only using two decimals, you get the same result:

```
> predict(model, newdata=expand.grid(LENGTH=16))¶
[1] 258.4850
```

The use of `expand.grid` is overkill here for a data frame with a single length but I am using it here because it anticipates our uses of `predict` and `expand.grid` below where we can actually get predictions for a large number of values in one go (as in the following; the output is not shown here):

```
> predict(model, newdata=expand.grid(LENGTH=1:16))¶
```

If you only use the model as an argument to `predict`, you get the values the model predicts for every observed word length in your data in the order of the data points (same with `fitted`).

```
> round(predict(model), 2)¶
     1      2      3      4      5      6      7      8
237.88 217.27 206.96 217.27 145.14 186.35 176.05 206.96
     9     10     11     12     13     14     15     16
186.35 206.96 196.66 165.75 248.18 227.57 248.18 186.35
    17     18     19     20
196.66 155.44 176.05 206.96
```

The first value of LENGTH is 14, so the first of the above values is the reaction time we expect for a word with 14 letters, etc. Since you now have the needed parameters, you can also draw the regression line. You do this with the function `abline`, which either takes a linear model object as an argument or the intercept and the slope; cf. Figure 36:

```
> plot(MS_LEARNER~LENGTH, xlim=c(0, 15), ylim=c(0, 300),
   xlab="Word length in letters", ylab="Reaction time of
   learners in ms"); grid()¶
> abline(model) # abline(93.61, 10.3)¶
```

It is obvious why the correlation coefficient is so high: the regression line is an excellent summary of the data points since all points are fairly close to it. (Below, we will see two ways of making this graph more informative.) We can even easily check how far away every predicted value is from its observed value.

This difference – the vertical distance between an observed *y*-value / reaction time and the *y*-value on the regression line for the corresponding *x*-value – is called a *residual*, and the function `residuals` requires just the linear model object as its argument.

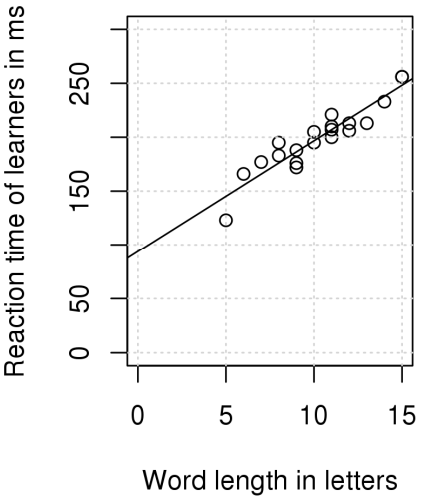

Figure 36. Scatterplot with regressions line for MS_LEARNER~LENGTH

```
> round(residuals(model), 2)¶
     1      2      3      4      5      6      7      8
 -4.88  -4.27  14.04 -11.27 -22.14 -10.35  18.95   0.04
     9     10     11     12     13     14     15     16
-14.35  -6.96   8.34  11.25   7.82 -14.57   7.82   1.65
    17     18     19     20
 -1.66 10.56   6.95   3.04
```

You can easily test manually that these are in fact the residuals:

```
> round(MS_LEARNER-(predict(model)+residuals(model)), 2)¶
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Note two important points though: First, regression equations and lines are most useful for the range of values covered by the observed values. Here, the regression equation was computed on the basis of lengths between 5 and 15 letters, which means that it will probably be much less reliable for lengths of 50+ letters. Second, in this case the regression equation also makes some rather non-sensical predictions because theoretically/ mathematically it predicts reactions times of around 0 ms for word lengths of -9. Such considerations will become important later on.

The correlation coefficient *r* also allows you to specify how much of the variance of one variable can be accounted for by the other variable. What does that mean? In our example, the values of both variables –

MS_LEARNER and LENGTH – are not all identical: they vary around their means and this variation was called dispersion and quantified with the standard deviation or the variance. If you square *r* and multiply the result by 100, then you obtain the amount of variance of one variable that the other variable accounts for. In our example, $r = 0.933$, which means that 87.18% of the variance of the reaction times can be accounted for – in a statistical sense, not necessarily a cause-effect sense – on the basis of the word lengths. This value, r^2, is referred to as *coefficient of determination*.

Incidentally, I sometimes heard students or colleagues compare two *r*-values such that they say something like, "Oh, here $r = 0.6$, nice, that's twice as much as in this other data set, where $r = 0.3$." Even numerically speaking, this is at least misleading, if nothing worse. Yes, 0.6 is twice as high as 0.3, but one should not compare *r*-values directly like this – one has to apply the so-called Fisher's *Z*-transformation first, which is exemplified in the following two lines:

```
> r<-0.3; 0.5*log((1+r)/(1-r))¶
[1] 0.3095196
> r<-0.6; 0.5*log((1+r)/(1-r))¶
[1] 0.6931472
> 0.6931472/0.3095196
[1] 2.239429
```

Thus, an *r*-value of 0.6 is twice as high as one of 0.3, but it reflects a correlation that is in fact nearly $2^1/_4$ times as strong. How about writing a function `fisher.z` that would compute *Z* from *r* for you …

The product-moment correlation *r* is probably the most frequently used correlation. However, there are a few occasions on which it should not be used. First, when the relevant variables are not interval/ratio-scaled but ordinal or when they are not both normally distributed (cf. below Section 4.4), then it is better to use another correlation coefficient, for example Kendall's tau τ. This correlation coefficient is based only on the ranks of the variable values and thus more suited for ordinal data. Second, when there are marked outliers in the variables, then you should also use Kendall's τ, because as a measure that is based on ordinal information only it is, just like the median, less sensitive to outliers. Cf. Figure 37, which shows a scatterplot with one noteworthy outlier in the top right corner. If you cannot justify excluding this data point, then it can influence *r* very strongly, but not τ. Pearson's *r* and Kendall's τ for all data points but the outlier are 0.11 and 0.1 respectively, and the regression line with the small slope shows that there is clearly no correlation between the two variables. However, if we

include the outlier, then Pearson's *r* suddenly becomes 0.75 (and the regression line's slope is changed markedly) while Kendall's τ remains appropriately small: 0.14.

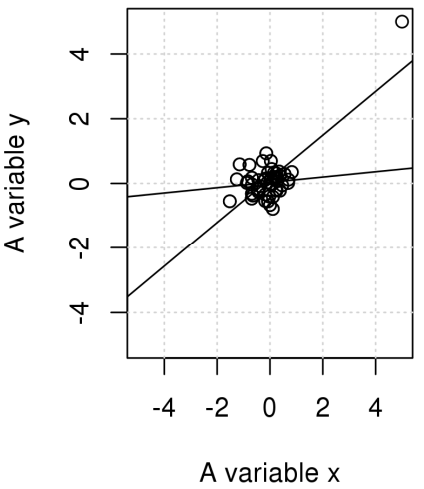

Figure 37. The effect of outliers on *r*

But how do you compute Kendall's τ? The computation of Kendall's τ is rather complex (especially with larger samples and ties), which is why I only explain how to compute it with R. The function is actually the same as for Pearson's *r* – `cor` – but the argument `method=…` is changed. For our experimental data we again get a high correlation, which turns out to be a little bit smaller than *r*. (Note that correlations are bidirectional – the order of the vectors does not matter – but linear regressions are not because you have a dependent and an independent variable and it matters what goes before the tilde – that which is predicted – and what goes after it.)

```
> cor(LENGTH, MS_LEARNER, method="kendall")¶
[1] 0.8189904
```

The previous explanations were all based on the assumption that there is in fact a linear correlation between the two variables or one that is best characterized with a straight line. This need not be the case, though, and a third scenario in which neither *r* nor τ are particularly useful involves cases where these assumptions do not hold. Often, this can be seen by just looking at the data. Figure 38 represents a well-known example from Anscombe (1973) (from <_inputfiles/03-2-3_anscombe.csv>), which has

the intriguing characteristics that

– the means and variances of the *x*-variable;
– the means and variances of the *y*-variable;
– the correlations and the linear regression lines of *x* and *y*;

are all identical although the distributions are obviously very different.

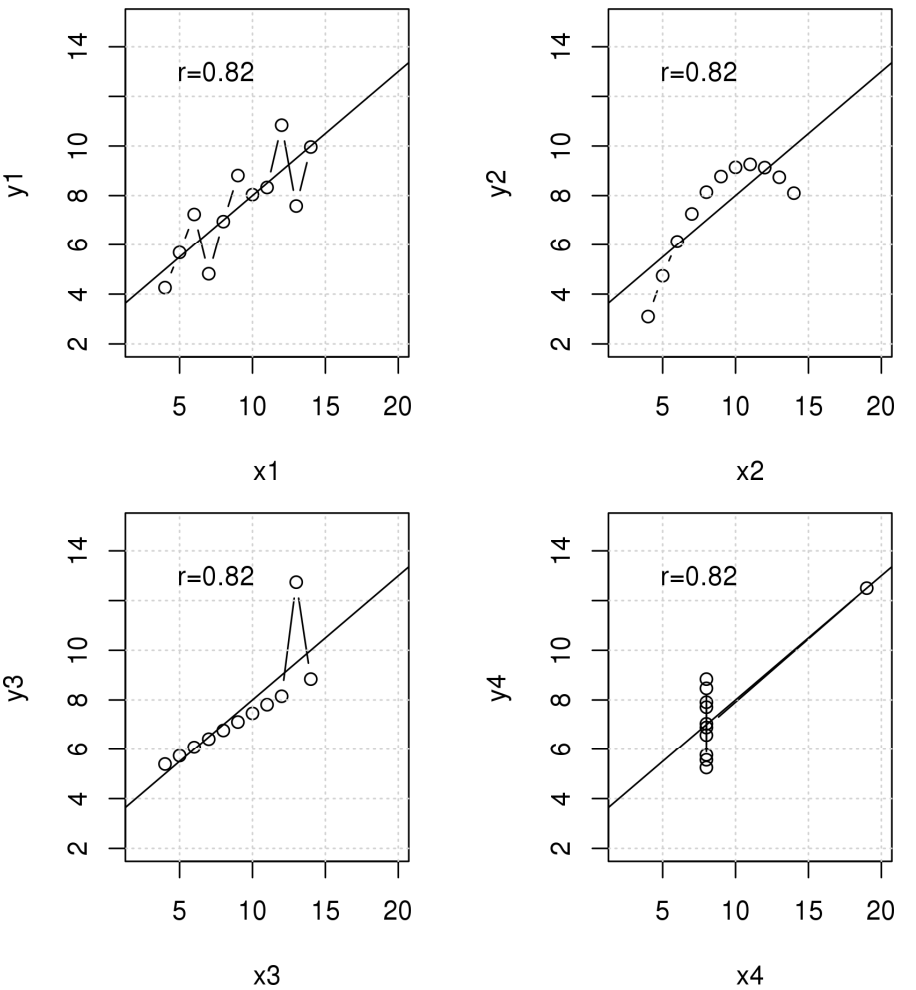

Figure 38. The sensitivity of linear correlations: the Anscombe data

In the top left of Figure 38, there is a case where r and τ are unproblematic. In the top right we have a situation where x and y are related in a curvilinear fashion – using a linear correlation here does not make much sense.[16] In the two lower panels, you see distributions in which individual outliers have a huge influence on r and the regression line. Since all the summary statistics are identical, this example illustrates most beautifully how important, in fact *indispensable*, a visual inspection of your data is, which is why in the following chapters visual exploration nearly always precedes statistical computation.

Now you should do the exercise(s) for Chapter 3 …

Warning/advice

Do not let the multitude of graphical functions and settings of R and/or your spreadsheet software tempt you to produce visual overkill. Just because you *can* use 6 different fonts, 10 colors, and cute little smiley symbols does not mean you *should*: Visualization should help you and/or the reader understand something otherwise difficult to grasp, which also means you should make sure your graphs are fairly self-sufficient, i.e. contain all the information required to understand them (e.g., meaningful graph and axis labels, legends, etc.) – a graph may need an explanation, but if the explanation is three quarters of a page, chances are your graph is not helpful (cf. Keen 2010: Chapter 1).

Recommendation(s) for further study

- the function `s.hist` (from the library `ade4`) and `scatterplot` (from the library `car`) to produce more refined scatterplots with histograms or boxplots
- Good and Hardin (2012: Ch. 8), Crawley (2007: Ch. 5, 27), Braun and Murdoch (2008: Section 3.2), and Keen (2010) for much advice to create good graphs; cf. also <http://cran.r-project.org/src/contrib/Views/Graphics.html>

16. I do not discuss nonlinear regressions; cf. Crawley (2007: Ch. 18, 20) for overviews.

Chapter 4
Analytical statistics

The most important questions of life are,
for the most part, really only questions of probability.
Pierre-Simon Laplace
(from <http://www-rohan.sdsu.edu/%7Emalouf/>)

In my description of the phases of an empirical study in Chapter 1, I skipped over one essential step: how to decide which significance test to use (Section 1.3.4). In this chapter, I will now discuss this step in some detail as well as then discuss how to conduct a variety of significance tests you may want to perform on your data. More specifically, in this chapter I will explain how descriptive statistics from Chapter 3 are used in the domain of hypothesis-testing. For example, in Section 3.1 I explained how you compute a measure of central tendency (such as a mean) or a measure of dispersion (such as a standard deviation) for a particular sample. In this chapter, you will see how you test whether such a mean or such a standard deviation differs significantly from a known mean or standard deviation or the mean or standard deviation of a second sample.

However, before we begin with actual tests: how do you decide which of the many tests out there is required for your hypotheses and data? One way to try to narrow down the truly bewildering array of tests is to ask yourself the six questions I will list in (24) to (29) and discuss presently, and the answers to these questions usually point you to only one or two tests that you can apply to your data. (A bit later, I will also provide a visual aid for this process.).

Ok, here goes. The first question is shown in (24).

(24) What kind of study are you conducting?

Typically, there are only two possible answers to that question: "hypothesis-generating" and "hypothesis-testing." The former means that you are approaching a (typically large) data set with the intentions of detecting structure(s) and developing hypotheses for future studies; your approach to the data is therefore data-driven, or bottom-up; an example for this will be discussed in Section 5.6. The latter is what most of the examples in this

book are about and means your approach to the data involves specific hypotheses you want to test and requires the types of tests in this chapter and most of the following one.

(25) What kinds of variables are involved in your hypotheses, and how many?

There are essentially two types of answers. One pertains to the information value of the variables and we have discussed this in detail in Section 1.3.2.2 above. The other allows for four different possible answers. First, you may only have one dependent variable, in which case, you normally want to compute a so-called goodness-of-fit test to test whether the results from your data correspond to other results (from a previous study) or correspond to a known distribution (such as a normal distribution). Examples include

- is the ratio of *no*-negations (e.g., *He is no stranger*) and *not*-negations (e.g., *He is not a stranger*) in your data 1 (i.e., the two negation types are equally likely)?
- does the average acceptability judgment you receive for a sentence correspond to that of a previous study?

Second, you may have one dependent and one independent variable or you may just have two sets of measurements (i.e. two dependent variables). In both cases you typically want to compute a monofactorial test for independence to determine whether the values of one/the independent variable are correlated with those of the other/dependent variable. For example,

- does the animacy of the referent of the direct object (a categorical independent variable) correlate with the choice of one of two postverbal constituent orders (a categorical dependent variable)?
- does the average acceptability judgment (a mean of a ratio/interval dependent variable) vary as a function of whether the subjects doing the rating are native speakers or not (a categorical independent variable)?

Third, you may have one dependent and two or more independent variables, in which case you want to compute a multifactorial analysis (such as a multiple regression) to determine whether the individual independent variables and their interactions correlate with, or predict, the dependent variable. For example,

- does the frequency of a negation type (a categorical dependent variable with the levels *NO* vs. *NOT*; cf. above) depend on the mode of communication (a binary independent variable with the levels *SPOKEN* vs. *WRITTEN*), the type of verb that is negated (a categorical independent variable with the levels *COPULA*, *HAVE*, or *LEXICAL*), and/or the interaction of these independent variables?
- does the reaction time to a word *w* in a lexical decision task (a ratio-scaled dependent variable) depend on the word class of *w* (a categorical independent variable), the frequency of *w* in a reference corpus (a ratio/interval independent variable), whether the subject has seen a word semantically related to *w* on the previous trial or not (a binary independent variable), whether the subject has seen a word phonologically similar to *w* on the previous trial or not (a binary independent variable), and/or the interactions of these independent variables?

Fourth, you have two or more dependent variables, in which case you may want to perform a multivariate analysis, which can be exploratory (such as hierarchical cluster analysis, principal components analysis, factor analysis, multi-dimensional scaling, etc.) or hypothesis-testing in nature (MANOVA). For example, if you retrieved from corpus data ten words and the frequencies of all content words occurring close to them, you can perform a cluster analysis to see which of the words behave more (or less) similarly to each other, which often is correlated with semantic similarity.

(26) Are data points in your data related such that you can associate them to each other meaningfully and in a principled way?

This question is concerned with whether you have what are called independent or dependent samples (and brings us back to the notion of independence discussed in Section 1.3.4.1). For example, your two samples – e.g., the numbers of mistakes made by ten male and ten female non-native speakers in a grammar test – are independent of each other if you cannot connect each male subject's value to that of one female subject on a meaningful and principled basis. You would not be able to do so if you randomly sampled ten men and ten women and let them take the same test.

There are two ways in which samples can be dependent. One is if you test subjects more than once, e.g., before and after a treatment. In that case, you could meaningfully connect each value in the before-treatment sample to a value in the after-treatment sample, namely connect each subject's two values. The samples are dependent because, for instance, if subject #1 is

very intelligent and good at the language tested, then these characteristics will make his results better than average in both tests, esp. compared to a subject who is less intelligent and proficient in the language and who will perform worse in both tests. Recognizing that the samples are dependent this way will make the test of before-vs.-after treatments more precise.

The second way in which samples may be dependent can be explained using the above example of ten men and ten women. If the ten men were the husbands of the ten women, then one would want to consider the samples dependent. Why? Because spouses are on average more similar to each other than randomly chosen people: they often have similar IQs, similar professions, they spend more time with each other than with randomly-selected people, etc. Thus, one should associate each husband with his wife, making this two dependent samples.

Independence of data points is often a very important criterion: many tests assume that data points are independent, and for many tests you must choose your test depending on what kind of samples you have.

(27) What is the statistic of the dependent variable in the statistical hypotheses?

There are essentially five different answers to this question, which were already mentioned in Section 1.3.2.3 above, too. Your dependent variable may involve frequencies/counts, central tendencies, dispersions, correlations, or distributions.

(28) What does the distribution of the data or your test statistic look like? Normal, some other way that can ultimately be described by a probability function (or a way that can be transformed to look like a probability function), or some other way?

(29) How big are the samples you collected? $n < 30$ or $n \geq 30$?

These questions relate back to Section 1.3.4, where I explained two things: First, if your data / test statistics follow a particular probability distribution, you can often use a computationally simpler parametric test, and if your data / test statistics don't, you must often use a non-parametric test. Second, given sufficient sample sizes, even data from a decidedly non-normal distribution can begin to look normal and, thus, allow you to apply parametric tests. It is safer, however, to be very careful and, maybe be conservative and run both types of tests.

Let us now use a graph (<sflwr_navigator.png>) that visualizes this pro-

cess, which you should have downloaded as part of all the files from the companion website. Let's exemplify the use of this graph using the above example scenario: you hypothesize that the average acceptability judgment (a mean of an ordinal dependent variable) varies as a function of whether the subjects providing the ratings are native or non-native speakers (a binary/categorical independent variable).

You start at the rounded red box with *approach* in it. Then, the above scenario is a hypothesis-testing scenario so you go down to *statistic*. Then, the above scenario involves averages so you go down to the rounded blue box with *mean* in it. Then, the hypothesis involves both a dependent and an independent variable so you go down to the right, via *1 DV 1 IV* to the transparent box with (tests for) *independence/difference* in it. You got to that box via the blue box with *mean* so you continue to the next blue box containing *information value*. Now you make two decisions: first, the dependent variable is ordinal in nature. Second, the samples are independent. Thus, you take the arrow down to the bottom left, which leads to a blue box with *U*-test in it. Thus, the typical test for the above question would be the *U*-test (to be discussed below), and the R function for that test is already provided there, too: `wilcox.test`.

Now, what does the dashed arrow mean that leads towards that box? It means that you would also do a *U*-test if your dependent variable was interval/ratio-scaled but violated other assumptions of the *t*-test. That is, dashed arrows provide alternative tests for the first-choice test from which they originate.

Obviously, this graph is a simplification and does not contain everything one would want to know, but I think it can help beginners to make first choices for tests so I recommend that, as you continue with the book, you always determine for each section which test to use and how to identify this on the basis of the graph.

Before we get started, let me remind you once again that in your own data your nominal/categorical variables should ideally always be coded with meaningful character strings so that R recognizes them as factors when reading in the data from a file. Also, I will assume that you have downloaded the data files from the companion website.

Recommendation(s) for further study
Good and Hardin (2012: Ch. 6) on choosing test statistics

1. Distributions and frequencies

In this section, I will illustrate how to test whether distributions and frequencies from one sample differ significantly from a known distribution (cf. Section 4.1.1) or from another sample (cf. Section 4.1.2). In both sections, we begin with variables from the interval/ratio level of measurement and then proceed to lower levels of measurement.

1.1. Distribution fitting

1.1.1. One dep. variable (ratio-scaled)

In this section, I will discuss how you compare whether the distribution of one dependent interval-/ratio-scaled variable is significantly different from a known distribution. I will restrict my attention to one of the most frequent cases, the situation where you test whether a variable is normally distributed (because as mentioned above in Section 1.3.4, many statistical techniques require a normal distribution so you must some know test like this).

We will deal with an example from the first language acquisition of tense and aspect in Russian. Simplifying a bit here, one can often observe a relatively robust correlation between past tense and perfective aspect as well as non-past tenses and imperfective aspect. Such a correlation can be quantified with Cramer's *V* values (cf. Stoll and Gries, 2009, and Section 4.2.1 below). Let us assume you studied how this association – the Cramer's *V* values – changes for one child over time. Let us further assume you had 117 recordings for this child, computed a Cramer's *V* value for each one, and now you want to see whether these are normally distributed. This scenario involves

- a dependent interval/ratio-scaled variable called TENSEASPECT, consisting of the Cramer's *V* values;
- no independent variable because you are not testing whether the distribution of the variable TENSEASPECT is influenced by, or correlated with, something else.

You can test for normality in several ways. The test we will use is the Shapiro-Wilk test (remember: check <sflwr_navigator.png> to see how we get to this test!), which does not really have any assumptions other than ratio-scaled data and involves the following procedure:

Procedure
- Formulating the hypotheses
- Visualizing the data
- Computing the test statistic W and p

As always, we begin with the hypotheses:

H_0: The data points do not differ from a normal distribution; $W = 1$.
H_1: The data points differ from a normal distribution; $W \neq 1$.

First, you load the data from <_inputfiles/04-1-1-1_tense-aspect.csv> and create a graph; the code for the left panel is shown below but you can also generate the right panel using the code from the code file.

```
> RussianTensAsp<-read.delim(file.choose())¶
> attach(RussianTensAsp)¶
> hist(TENSE_ASPECT, xlim=c(0, 1), main="", xlab="Tense-Apect
   correlation", ylab="Frequency") # left panel¶
```

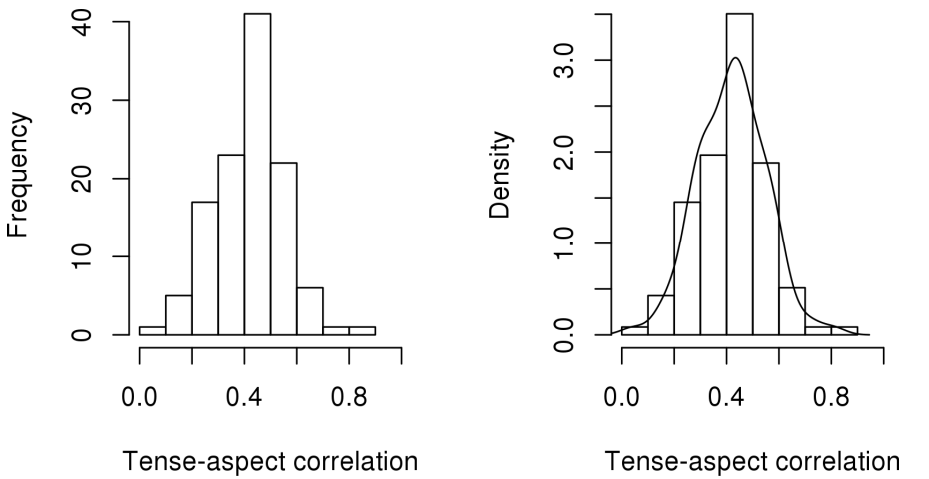

Figure 39. Histogram of the Cramer's V values reflecting the strengths of the tense-aspect correlations

At first glance, this looks very much like a normal distribution, but of course you must do a real test. The Shapiro-Wilk test is rather cumbersome to compute semi-manually, which is why its manual computation will not be discussed here (unlike nearly all other monofactorial tests). In R, how-

ever, the computation could not be easier. The relevant function is called `shapiro.test` and it only requires one argument, the vector to be tested:

```
> shapiro.test(TENSE_ASPECT)¶
        Shapiro-Wilk normality test
data:  TENSE_ASPECT
W = 0.9942, p-value = 0.9132
```

What does this mean? This simple output teaches an important lesson: Usually, you want to obtain a significant result, i.e., a *p*-value that is smaller than 0.05 because this allows you to accept H_1. Here, however, you may actually welcome an insignificant result because normally-distributed variables are often easier to handle. The reason for this is again the logic underlying the falsification paradigm. When $p < 0.05$, you reject H_0 and accept H_1. But here you 'want' H_0 to be true because H_0 states that the data are normally distributed. You obtained a *p*-value of 0.9132, which means you cannot reject H_0 and, thus, consider the data to be normally distributed. You would therefore summarize this result in the results section of your paper as follows: "According to a Shapiro-Wilk test, the distribution of this child's Cramer's *V* values measuring the tense-aspect correlation does not deviate significantly from normality: $W = 0.9942$; $p = 0.9132$." (In parentheses or after a colon you usually mention all statistics that helped you decide whether or not to accept H_1.)

As an alternative to the Shapiro-Wilk test, you can also use a Kolmogorov-Smirnov test for goodness of fit. This test requires the function `ks.test` and is more flexible than the Shapiro-Wilk-Test, since it can test for more than just normality and can also be applied to vectors with more than 5000 data points. To test the Cramer's *V* value for normality, you provide them as the first argument, then you name the distribution you want to test against (for normality, `"pnorm"`), and then, to define the parameters of the normal distribution, you provide the mean and the standard deviation of the Cramer's *V* values:

```
> ks.test(TENSE_ASPECT, "pnorm", mean=mean(TENSE_ASPECT),
   sd=sd(TENSE_ASPECT))¶
 One-sample Kolmogorov-Smirnov test
data:  TENSE_ASPECT
D = 0.078, p-value = 0.4752
alternative hypothesis: two-sided
```

The result is the same as above: the data do not differ significantly from normality. You also get a warning because `ks.test` assumes that no two

values in the input are the same, but here some values (e.g., 0.27, 0.41, and others) are attested more than once; below you will see a quick and dirty fix for this problem.

Recommendation(s) for further study
- as alternatives to the above functions, the functions `jarqueberaTest` and `dagoTest` (both from the library `fBasics`)
- the function `mshapiro.test` (from the library `mvnormtest`) to test for multivariate normality
- the function `qqnorm` and its documentation (for quantile-quantile plots)
- Crawley (2005: 100f.), Crawley (2007: 316f.), Sheskin (2011: Test 7)

1.1.2. One dep. variable (nominal/categorical)

In this section, we are going to return to an example from Section 1.3, the constructional alternation of particle placement in English, which is again represented in (30).

(30) a. He picked up the book. (verb - particle - direct object)
b. He picked the book up. (verb - direct object - particle)

As you already know, often both constructions are acceptable and native speakers can often not explain their preference for one of the two. One may therefore expect that both constructions are equally frequent, and this is what you are going to test. This scenario involves

- a dependent nominal/categorical variable CONSTRUCTION: *VERB-PARTICLE-OBJECT* vs. CONSTRUCTION: *VERB-OBJECT-PARTICLE*;
- no independent variable, because you do not investigate whether the distribution of CONSTRUCTION is dependent on anything else.

Such questions are generally investigated with tests from the family of chi-squared tests, which is one of the most important and widespread tests. Since there is no independent variable, you test the degree of fit between your observed and an expected distribution, which should remind you of Section 3.1.5.2. This test is referred to as the chi-squared goodness-of-fit test and involves the following steps:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Computing the frequencies you would expect given H_0
- Testing the assumption(s) of the test:
  - all observations are independent of each other
  - 80% of the expected frequencies are ≥ 5[17]
  - all expected frequencies are > 1
- Computing the contributions to chi-squared for all observed frequencies
- Computing the test statistic χ^2, *df*, and *p*

The first step is very easy here. As you know, H_0 typically postulates that the data are distributed randomly/evenly, and that means that both constructions occur equally often, i.e., 50% of the time (just as tossing a fair coin many times will result in a largely equal distribution). Thus:

H_0: The frequencies of the two variable levels of CONSTRUCTION are identical – if you find a difference in your sample, this difference is just random variation; $n_{\text{V Part DO}} = n_{\text{V DO Part}}$.

H_1: The frequencies of the two variable levels of CONSTRUCTION are not identical; $n_{\text{V Part DO}} \neq n_{\text{V DO Part}}$.

Note that this is a two-tailed H_1; no direction of the difference is provided. Next, you would collect some data and count the occurrences of both constructions, but we will abbreviate this step and use frequencies reported in Peters (2001). She conducted an experiment in which subjects described pictures and obtained the construction frequencies represented in Table 19.

Table 19. Observed construction frequencies of Peters (2001)

Verb - Particle - Direct Object	Verb - Direct Object - Particle
247	150

17. This threshold value of 5 is the one most commonly mentioned. There are a few studies that show that the chi-squared test is fairly robust even if this assumption is violated – especially when, as is here the case, H_0 postulates that the expected frequencies are equally high (cf. Zar 1999: 470). However, to keep things simple, I stick to the most common conservative threshold value of 5 and refer you to the literature quoted in Zar. If your data violate this assumption, then you must compute a binomial test (if, as here, you have two groups) or a multinomial test (for three or more groups); cf. the recommendations for further study.

Obviously, there is a strong preference for the construction in which the particle follows the verb directly. At first glance, it seems very unlikely that H_0 could be correct, given these data.

One very important side remark here: beginners often look at something like Table 19 and say, oh, ok, we have interval/ratio data: 247 and 150. Why is this wrong?

THINK
BREAK

It's wrong because Table 19 does not show you the raw data – what it shows you is already a numerical summary. You don't have interval/ratio data – you have an interval/ratio summary of categorical data, because the numbers 247 and 150 summarize the frequencies of the two levels of the categorical variable CONSTRUCTION (which you probably obtained from applying `table` to a vector/factor). One strategy to not mix this up is to always conceptually envisage what the raw data table would look like in the case-by-variable format discussed in Section 1.3.3. In this case, it would look like this:

Table 20. The case-by-variable version of the data in Table 19

CASE	CONSTRUCTION
1	vpo
2	vpo
247	vpo
248	vop
	vop
397	vop

From this format, it is quite obvious that the variable CONSTRUCTION is categorical. So, don't mix up interval/ratio summaries of categorical data with interval/ratio data.

As the first step of our evaluation, you should now have a look at a graphical representation of the data. A first possibility would be to generate, say, a dot chart. Thus, you first enter the two frequencies – first the frequency data, then the names of the frequency data (for the plotting) – and then you create a dot chart or a bar plot as follows:

```
> VPCs<-c(247, 150) # VPCs="verb-particle constructions"¶
> names(VPCs)<-c("V-Part-DO", "V-DO-Part")¶
> dotchart(VPCs, xlim=c(0, 250))¶
> barplot(VPCs)¶
```

The question now of course is whether this preference is statistically significant or whether it could just as well have arisen by chance. According to the above procedure, you must now compute the frequencies that follow from H_0. In this case, this is easy: since there are altogether 247+150 = 397 constructions, which should be made up of two equally large groups, you divide 397 by 2:

```
> VPCs.exp<-rep(sum(VPCs)/length(VPCs), length(VPCs))¶
> VPCs.exp¶
[1] 198.5 198.5
```

You must now check whether you can actually do a chi-squared test here, but the observed frequencies are obviously larger than 5 and we assume that Peters's data points are in fact independent (because we will assume that each construction has been provided by a different speaker). We can therefore proceed with the chi-squared test, the computation of which is fairly straightforward and summarized in (31).

(31) $$\text{Pearson chi-squared} = \chi^2 = \sum_{i=1}^{n} \frac{\left(observed - expected\right)^2}{expected}$$

That is to say, for every value of your frequency table you compute a so-called contribution to chi-squared by (i) computing the difference between the observed and the expected frequency, (ii) squaring this difference, and (iii) dividing that by the expected frequency again. The sum of these contributions to chi-squared is the test statistic chi-squared. Here, it is approximately 23.7.

(32) $$\text{Pearson } \chi^2 = \frac{\left(247 - 198.5\right)^2}{198.5} + \frac{\left(150 - 198.5\right)^2}{198.5} \approx 23.7$$

```
> sum(((VPCs-VPCs.exp)^2)/VPCs.exp)¶
[1] 23.70025
```

Obviously, this value increases as the differences between observed and

expected frequencies increase (because then the numerators become larger). That also means that chi-squared becomes 0 when all observed frequencies correspond to all expected frequencies: then the numerators become 0. Thus, we can simplify our statistical hypotheses to the following:

H_0: $\chi^2 = 0$.
H_1: $\chi^2 > 0$.

But the chi-squared value alone does not show you whether the differences are large enough to be statistically significant. So, what do you do with this value? Before computers became more widespread, a chi-squared value was used to look up whether the result is significant or not in a chi-squared table. Such tables typically have the three standard significance levels in the columns and different numbers of degrees of freedom (*df*) in the rows. *Df* here is the number of categories minus 1, i.e., *df* = 2-1 = 1, because when we have two categories, then one category frequency can vary freely but the other is fixed (so that we can get the observed number of elements, here 397). Table 21 is one such chi-squared table for the three significance levels and *df* = 1 to 3.

Table 21. Critical χ^2-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $1 \leq df \leq 3$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 1$	3.841	6.635	10.828
$df = 2$	5.991	9.21	13.816
$df = 3$	7.815	11.345	16.266

You can actually generate those values yourself with the function `qchisq`. That function requires three arguments:

- `p`: the *p*-value(s) for which you need the critical chi-squared values (for some *df*);
- `df`: the *df*-value(s) for the *p*-value for which you need the critical chi-squared value;
- `lower.tail=FALSE`: the argument to instruct R to only use the area under the chi-squared distribution curve that is to the right of / larger than the observed chi-squared value.

```
> qchisq(c(0.05, 0.01, 0.001), 1, lower.tail=FALSE)¶
[1]  3.841459  6.634897 10.827566
```

More advanced users find code to generate all of Table 21 in the code file. Once you have such a table, you can test your observed chi-squared value for significance by determining whether it is larger than the chi-squared value(s) tabulated at the observed number of degrees of freedom. You begin with the smallest tabulated chi-squared value and compare your observed chi-squared value with it and continue to do so as long as your observed value is larger than the tabulated ones. Here, you first check whether the observed chi-squared is significant at the level of 5%, which is obviously the case: 23.7 > 3.841. Thus, you can check whether it is also significant at the level of 1%, which again is the case: 23.7 > 6.635. Thus, you can finally even check if the observed chi-squared value is maybe even highly significant, and again this is so: 23.7 > 10.827. You can therefore reject H_0 and the usual way this is reported in your results section is this: "According to a chi-squared goodness-of-fit test, the frequency distribution of the two verb-particle constructions deviates highly significantly from the expected one ($\chi^2 = 23.7$; $df = 1$; $p_{\text{two-tailed}} < 0.001$): the construction where the particle follows the verb directly was observed 247 times although it was only expected 199 times, and the construction where the particle follows the direct objet was observed only 150 times although it was expected 199 times."

With larger and more complex amounts of data, this semi-manual way of computation becomes more cumbersome (and error-prone), which is why we will simplify all this a bit. First, you can of course compute the *p*-value directly from the chi-squared value using the mirror function of `qchisq`, viz. `pchisq`, which requires the above three arguments:

```
> pchisq(23.7, 1, lower.tail=FALSE)¶
[1] 1.125825e-06
```

As you can see, the level of significance we obtained from our stepwise comparison using Table 21 is confirmed: *p* is indeed much smaller than 0.001, namely 0.00000125825. However, there is another even easier way: why not just do the whole test with one function? The function is called `chisq.test`, and in the present case it requires maximally three arguments:

- `x`: a vector with the observed frequencies;
- `p`: a vector with the expected percentages (not the frequencies!);
- `correct=TRUE` or `correct=FALSE`: when the sample size *n* is small ($15 \leq n \leq 60$), it is sometimes recommended to apply a so-called continuity

correction (after Yates); `correct=TRUE` is the default setting.[18]

In this case, this is easy: you already have a vector with the observed frequencies, the sample size n is much larger than 60, and the expected probabilities result from H_0. Since H_0 says the constructions are equally frequent and since there are just two constructions, the vector of the expected probabilities contains two times $^1/_2 = 0.5$. Thus:

```
> chisq.test(VPCs, p=c(0.5, 0.5))¶
 Chi-squared test for given probabilities
data:  VPCs
X-squared = 23.7003, df = 1, p-value = 1.126e-06
```

You get the same result as from the manual computation but this time you immediately also get a p-value. What you do not also get are the expected frequencies, but these can be obtained very easily, too. The function `chisq.test` computes more than it returns. It returns a data structure (a so-called list) so you can assign a name to this list and then inspect it for its contents (output not shown):

```
> test<-chisq.test(VPCs, p=c(0.5, 0.5))¶
> str(test)¶
```

Thus, if you require the expected frequencies, you just retrieve them with a $ and the name of the list component you want, and of course you get the result you already know.

```
> test$expected¶
[1] 198.5 198.5
```

Let me finally mention that the above method computes a p-value for a two-tailed test. There are many tests in R where you can define whether you want a one-tailed or a two-tailed test. However, this does not work with the chi-squared test. If you require the critical chi-squared value for $p_{\text{one-tailed}} = 0.05$ for $df = 1$, then you must compute the critical chi-squared value for $p_{\text{two-tailed}} = 0.1$ for $df = 1$ (with `qchisq(0.1, 1, lower.tail=FALSE)¶`), since your prior knowledge is rewarded such that a less extreme result in the predicted direction will be sufficient (cf. Section 1.3.4). Also, this means that when you need the $p_{\text{one-tailed}}$-value for a chi-square value, just take half of the $p_{\text{two-tailed}}$-value of the same chi-square value. In this

18. For further options, cf. `?chisq.test¶`, `formals(chisq.test)¶` or `args(chisq.test)¶`.

case, if your H_1 had been directional, this would have been your *p*-value. But again: this works only with $df = 1$.

```
> pchisq(23.7, 1, lower.tail=FALSE)/2¶
```

Warning/advice

Above I warned you to never change your hypotheses *after* you have obtained your results and then sell your study as successful support of the 'new' H_1. The same logic does not allow you to change your hypothesis from a two-tailed one to a one-tailed one because your $p_{\text{two-tailed}} = 0.08$ (i.e., non-significant) so that the corresponding $p_{\text{one-tailed}} = 0.04$ (i.e., significant). Your choice of a one-tailed hypothesis must be motivated *conceptually*.

Another hugely important warning: never ever compute a chi-square test like the above on percentages – always on 'real' observed frequencies!

Recommendation(s) for further study

- the functions `binom.test` or `dbinom` to compute binomial tests
- the function `prop.test` (cf. Section 3.1.5.2) to test relative frequencies / percentages for deviations from expected frequencies / percentages
- the function `dmultinom` to help compute multinomial tests
- Baayen (2008: Section 4.1.1), Sheskin (2011: Test 8, 9)

1.2. Tests for differences/independence

In Section 4.1.1, we looked at goodness-of-fit tests for distributions and frequencies – now we turn to tests for differences/independence.

1.2.1. One dep. variable (ordinal/interval/ratio scaled) and one indep. variable (nominal) (indep. samples)

Let us now look at an example in which two independent samples are compared with regard to their overall distributions. You will test whether men and women differ with regard to the frequencies of hedges they use in discourse (i.e., expressions such as *kind of* or *sort of*). Again, note that we are here only concerned with the overall distributions – not just means or just variances. We could of course do that, too, but it is of course possible that the means are very similar while the variances are not and a test for differ-

ent means might not uncover the overall distributional difference.

Let us assume you have recorded 60 two-minute conversations between a confederate of an experimenter, each with one of 30 men and 30 women, and then counted the numbers of hedges that the male and female subjects produced. You now want to test whether the distributions of hedge frequencies differs between men and women. This question involves

- an independent nominal/categorical variable, SEX: *MALE* and SEX: *FEMALE*;
- a dependent interval/ratio-scaled: the number of hedges produced: HEDGES.

The question of whether the two sexes differ in terms of the distributions of hedge frequencies is investigated with the two-sample Kolmogorov-Smirnov test (again, check <sflwr_navigator.png>):

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test: the data are continuous
- Computing the cumulative frequency distributions for both samples, the maximal absolute difference D of both distributions, and p

First the hypotheses: the text form is straightforward and the statistical version is based on a test statistic called D to be explained below

H_0: The distribution of the dependent variable HEDGES does not differ depending on the levels of the independent variable SEX; $D = 0$.

H_1: The distribution of the dependent variable HEDGES differs depending on the levels of the independent variable SEX; $D > 0$.

Before we do the actual test, let us again inspect the data graphically. You first load the data from <_inputfiles/04-1-2-1_hedges.csv>, check the data structure (I will usually not show that output here in the book), and make the variable names available.

```
> Hedges<-read.delim(file.choose())¶
> str(Hedges)¶
> attach(Hedges)¶
```

You are interested in the general distribution, so one plot you can create is a stripchart. In this kind of plot, the frequencies of hedges are plotted separately for each sex, but to avoid that identical frequencies are plotted directly onto each other (and can therefore not be distinguished anymore), you also use the argument `method="jitter"` to add a tiny value to each data point, which decreases the chance of overplotted data points (also try `method="stack"`). Then, you include the meaningful point of $x = 0$ on the x-axis. Finally, with the function `rug` you add little bars to the x-axis (`side=1`) which also get jittered. The result is shown in Figure 40.

```
> stripchart(HEDGES~SEX, method="jitter", xlim=c(0, 25),
   xlab="Number of hedges", ylab="Sex")¶
> rug(jitter(HEDGES), side=1)¶
```

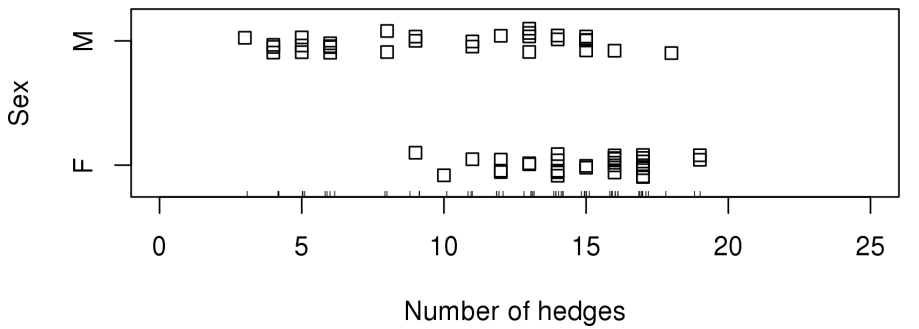

Figure 40. Stripchart for HEDGES~SEX

It is immediately obvious that the data are distributed quite differently: the values for women appear to be a little higher on average and more homogeneous than those of the men. The data for the men also appear to fall into two groups, a suspicion that also receives some *prima facie* support from the following two histograms in Figure 41. (Note that all axis limits are again defined identically to make the graphs easier to compare.)

```
> par(mfrow=c(1, 2))¶
> hist(HEDGES[SEX=="M"], xlim=c(0, 25), ylim=c(0, 10), ylab=
   "Frequency", main="")¶
> hist(HEDGES[SEX=="F"], xlim=c(0, 25), ylim=c(0, 10), ylab=
   "Frequency", main="")¶
> par(mfrow=c(1, 1))¶
```

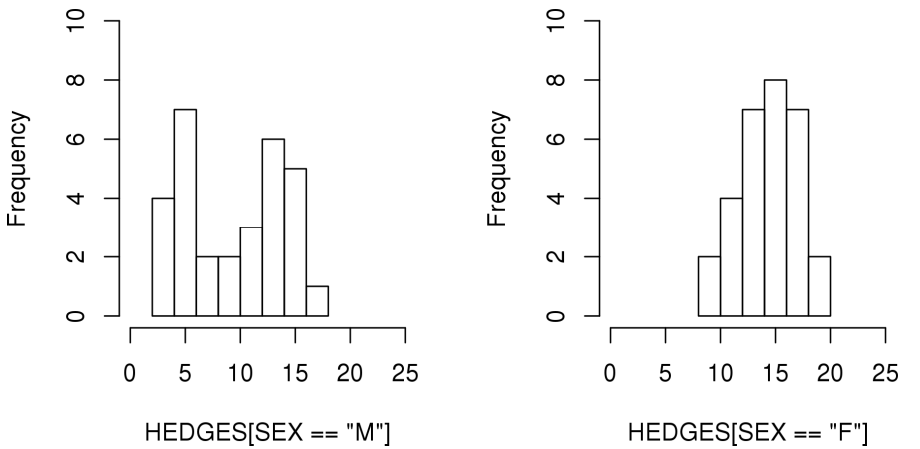

Figure 41. Histograms of the number of hedges by men and women

The assumption of continuous data points is not exactly met because frequencies are discrete – there are no frequencies 3.3, 3.4, etc. – but HEDGES spans quite a range of values and we could in fact jitter the values to avoid ties. To test these distributional differences with the Kolmogorov-Smirnov test, which involves the empirical cumulative distribution of the data, you first rank-order the data: You sort the values of SEX in the order in which you need to sort HEDGES, and then do the same to HEDGES itself:

```
> SEX<-SEX[order(HEDGES)]¶
> HEDGES<-HEDGES[order(HEDGES)]¶
```

The next step is a little more complex. You must now compute the maximum of all differences of the two cumulative distributions of the hedges. You can do this in three steps: First, you generate a frequency table with the numbers of hedges in the rows and the sexes in the columns. This table in turn serves as input to `prop.table`, which generates a table of column percentages (hence `margin=2`; cf. Section 3.2.1, output not shown):

```
> dists<-prop.table(table(HEDGES, SEX), margin=2); dists¶
```

This table shows that, say, 10% of all numbers of hedges of men are 4, but these are of course not cumulative percentages yet. The second step is therefore to convert these percentages into cumulative percentages. You can use `cumsum` to generate the cumulative percentages for both columns and can even compute the differences in the same line:

```
> differences<-cumsum(dists[,1])-cumsum(dists[,2])¶
```

That is, you subtract from every cumulative percentage of the first column (the values of the women) the corresponding value of the second column (the values of the men). The third and final step is then to determine the maximal absolute difference, which is the test statistic *D*:

```
> max(abs(differences))¶
[1] 0.4666667
```

You can then look up this value in a table for Kolmogorov-Smirnov tests; for a significant result, the computed value must be larger than the tabulated one. For cases in which both samples are equally large, Table 22 shows the critical *D*-values for two-tailed Kolmogorov-Smirnov tests (computed from Sheskin 2011: Table A23).

Table 22. Critical *D*-values for two-sample Kolmogorov-Smirnov tests

	$p = 0.05$	$p = 0.01$
$n_1 = n_2 = 29$	0.3571535	0.428059
$n_1 = n_2 = 30$	0.3511505	0.4208642
$n_1 = n_2 = 31$	0.3454403	0.4140204

Our value of $D = 0.4667$ is not only significant ($D > 0.3511505$), but even very significant ($D > 0.4208642$). You can therefore reject H_0 and summarize the results: "According to a two-sample Kolmogorov-Smirnov test, there is a significant difference between the distributions of hedge frequencies of men and women: women seem to use more hedges and behave more homogeneously than the men, who use fewer hedges and whose data appear to fall into two groups ($D = 0.4667$, $p_{\text{two-tailed}} < 0.01$)."

The logic of this test is not always immediately clear but worth exploring. To that end, we look at a graphical representation. The following lines plot the two empirical cumulative distribution functions (`ecdf`) of men (in black) and women (in grey) as well as a vertical line at position $x = 9$, where the largest difference ($D = 0.4667$) was found. This graph in Figure 42 below shows what the Kolmogorov-Smirnov test reacts to: different empirical cumulative distributions.

```
> plot(ecdf(HEDGES[SEX=="M"]), do.points=TRUE, verticals=
   TRUE, main="Hedges: men (black) vs. women (grey)",
   xlab="Numbers of hedges")¶
> lines(ecdf(HEDGES[SEX=="F"]), do.points=TRUE, verticals=
```

```
   TRUE, col="darkgrey")¶
> abline(v=9, lty=2)¶
```

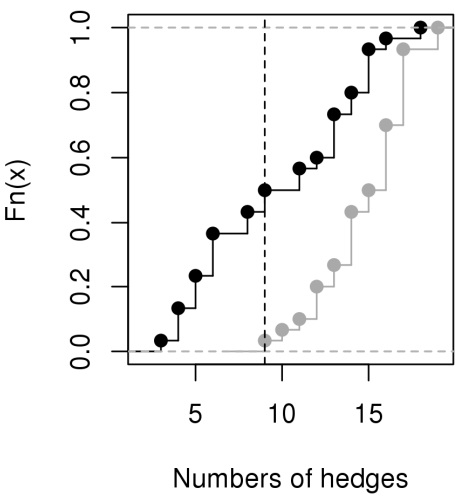

Figure 42. Empirical cumulative distribution functions of the numbers of hedges of men (black) and women (grey)

For example, the fact that the values of the women are higher and more homogeneous is indicated especially in the left part of the graph where the low hedge frequencies are located and where the values of the men already rise but those of the women do not. More than 40% of the values of the men are located in a range where no hedge frequencies for women were obtained at all. As a result, the largest difference at position $x = 9$ arises where the curve for the men has already risen considerably while the curve for the women has only just begun to take off. This graph also explains why H_0 postulates $D = 0$. If the curves are completely identical, there is no difference between them and D becomes 0.

The above explanation simplified things a bit. First, you do not always have two-tailed tests and identical sample sizes. Second, identical values – so-called *ties* – can complicate the computation of this test (and others). Fortunately, you do not really have to worry about any of this because the R function `ks.test` does everything for you in just one line. You just need the following arguments:[19]

– `x` and `y`: the two vectors whose distributions you want to compare;

19. Unfortunately, the function `ks.test` does not take a formula as input.

– `alternative="two-sided"` for two-tailed tests (the default) or `alternative="greater"` or `alternative="less"` for one-sided tests depending on which H_1 you want to test: the argument `alternative="…"` refers to the first-named vector so that `alternative="greater"` means that the cumulative distribution function of the first vector is above that of the second.

When you test a two-tailed H_1 as we do here, then the line to enter into R reduces to the following, and you get the same *D*-value and the *p*-value. (I omitted the warning about ties here but, again, you can use `jitter` to get rid of it; cf. the code file.)

```
> ks.test(HEDGES[SEX=="M"], HEDGES[SEX=="F"])¶
 Two-sample Kolmogorov-Smirnov test
data:  HEDGES[SEX == "M"] and HEDGES[SEX == "F"]
D = 0.4667, p-value = 0.002908
alternative hypothesis: two-sided
```

Recommendation(s) for further study

– apart from the function mentioned in the text (`plot(ecdf(…))`), you can create such graphs also with `plot.stepfun`
– Crawley (2005: 100f.), Crawley (2007: 316f.), Baayen (2008: Section 4.2.1), Sheskin (2011: Test 13)

1.2.2. One dep. variable (nominal/categorical) and one indep. variable (nominal/categorical) (indep. samples)

In Section 4.1.1.2 above, we discussed how you test whether the distribution of a dependent nominal/categorical variable is significantly different from another known distribution. A probably more frequent situation is that you test whether the distribution of one nominal/categorical variable is dependent on another nominal/categorical variable.

Above, we looked at the frequencies of the two verb-particle constructions. We found that their distribution was not compatible with H_0. However, we also saw earlier that there are many variables that are correlated with the constructional choice. One of these is whether the referent of the direct object is given information, i.e., known from the previous discourse, or not. Specifically, previous studies found that objects referring to given referents prefer the position before the particle whereas objects referring to new referents prefer the position after the particle. We will look at this hypothesis

(for the sake of simplicity as a two-tailed hypothesis). It involves

- a dependent nominal/categorical variable, namely CONSTRUCTION: *VERB-PARTICLE-OBJECT* vs. CONSTRUCTION: *VERB-OBJECT-PARTICLE*;
- an independent variable nominal/categorical variable, namely the givenness of the referent of the direct object: GIVENNESS: *GIVEN* vs. GIVENNESS: *NEW*;
- independent samples because we will assume that, in the data below, the fact any particular constructional choice is unrelated to any other one (this is often far from obvious, but too complex to be discussed here in more detail).

As before, such questions are investigated with chi-squared tests: you test whether the levels of the independent variable result in different frequencies of the levels of the dependent variable. The overall procedure for a chi-squared test for independence is very similar to that of a chi-squared test for goodness of fit, but you will see below that the computation of the expected frequencies is (only superficially) a bit different from above.

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Computing the frequencies you would expect given H_0
- Testing the assumption(s) of the test:
  - all observations are independent of each other
  - 80% of the expected frequencies are ≥ 5 (cf. n. 17)
  - all expected frequencies are > 1
- Computing the contributions to chi-squared for all observed frequencies
- Computing the test statistic χ^2, *df*, and *p*

The hypotheses are simple, especially since we apply what we learned from the chi-squared test for goodness of fit from above:

H_0: The frequencies of the levels of the dependent variable CONSTRUCTION do not vary as a function of the levels of the independent variable GIVENNESS; $\chi^2 = 0$.

H_1: The frequencies of the levels of the dependent variable CONSTRUCTION vary as a function of the levels of the independent variable GIVENNESS; $\chi^2 > 0$.

In order to discuss this version of the chi-squared test, we return to the data from Peters (2001). As a matter of fact, the above discussion did not utilize all of Peters's data because I omitted an independent variable, namely GIVENNESS. Peters (2001) did not just study the frequency of the two constructions – she studied what we are going to look at here, namely whether GIVENNESS is correlated with CONSTRUCTION. In the picture-description experiment described above, she manipulated the variable GIVENNESS and obtained the already familiar 397 verb-particle constructions, which patterned as represented in Table 23. (By the way, the cells of such 2-by-2 tables are often referred to with the letters *a* to *d*, *a* being the top left cell (85), *b* being the top right cell (65), etc.)

Table 23. Observed construction frequencies of Peters (2001)

	GIVENNESS: *GIVEN*	GIVENNESS: *NEW*	Row totals
CONSTRUCTION: *V DO PART*	85	65	150
CONSTRUCTION: *V PART DO*	100	147	247
Column totals	185	212	397

First, we explore the data graphically. You load the data from <_inputfiles/04-1-2-2_vpcs.csv>, create a table of the two factors, and get a first visual impression of the distribution of the data (cf. Figure 43).

```
> VPCs<-read.delim(file.choose())¶
> str(VPCs); attach(VPCs)¶
> Peters.2001<-table(CONSTRUCTION, GIVENNESS)¶
> plot(CONSTRUCTION~GIVENNESS)¶
```

Obviously, the differently-colored areas are differently big between rows/columns. To test these differences for significance, we need the frequencies expected from H_0. But how do we compute the frequencies predicted by H_0? Since this is a central question, we will discuss this in detail.

Let us assume Peters had obtained the totals in Table 24. What would the distribution following from H_0 look like? Above in Section 4.1.1.2, we said that H_0 typically postulates equal frequencies. Thus, you might assume – correctly – that the expected frequencies are those represented in Table 24. All marginal totals are 100 and every variable has two equally frequent levels so we have 50 in each cell.

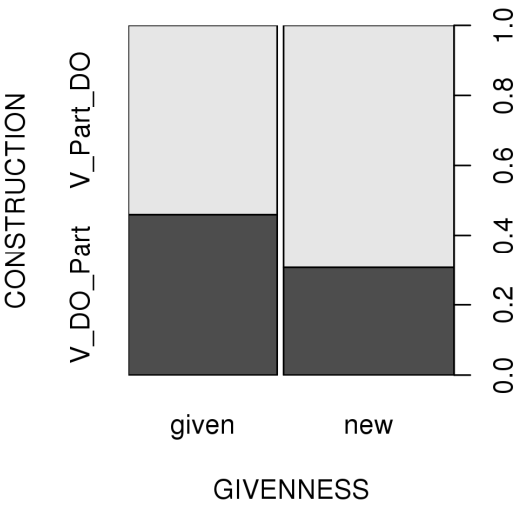

Figure 43. Mosaic plot for CONSTRUCTION~GIVENNESS

Table 24. Fictitious observed construction frequencies of Peters (2001)

	GIVENNESS: *GIVEN*	GIVENNESS: *NEW*	Row totals
CONSTRUCTION: *V DO PART*			100
CONSTRUCTION: *V PART DO*			100
Column totals	100	100	200

Table 25. Fictitious expected construction frequencies of Peters (2001)

	GIVENNESS: *GIVEN*	GIVENNESS: *NEW*	Row totals
CONSTRUCTION: *V DO PART*	50	50	100
CONSTRUCTION: *V PART DO*	50	50	100
Column totals	100	100	200

The statistical hypotheses that go beyond just stating whether or not $\chi^2 = 0$ would then be:

H_0: $n_{\text{V DO Part \& Ref DO = given}} = n_{\text{V DO Part \& Ref DO} \neq \text{given}} = n_{\text{V Part DO \& Ref DO = given}} = n_{\text{V Part DO \& Ref DO} \neq \text{given}}$

H_1: as H_0, but there is at least one "≠" instead of an "=".

However, life is usually not that simple, for example when (a) as in Peters (2001) not all subjects answer all questions or (b) naturally-observed data are counted that are not as nicely balanced. Thus, in Peters's real data, it does not make sense to simply assume equal frequencies. Put differently, H_0 cannot look like Table 24 because the row totals of Table 23 show that the different levels of GIVENNESS are not equally frequent. If GIVENNESS had no influence on CONSTRUCTION, you would expect that the frequencies of the two constructions for each level of GIVENNESS would exactly reflect the frequencies of the two constructions in the whole sample. That means (i) all marginal totals (row/column totals) must remain constant (as they reflect the numbers of the investigated elements), and (ii) the proportions of the marginal totals determine the cell frequencies in each row and column. From this, a rather complex set of hypotheses follows:

H_0: $n_{\text{V DO Part \& Ref DO = given}} : n_{\text{V DO Part \& Ref DO} \neq \text{given}} \propto$
$n_{\text{V Part DO \& Ref DO = given}} : n_{\text{V Part DO \& Ref DO} \neq \text{given}} \propto$
$n_{\text{Ref DO = given}} : n_{\text{Ref DO} \neq \text{given}}$ and
$n_{\text{V DO Part \& Ref DO = given}} : n_{\text{V Part DO \& Ref DO = given}} \propto$
$n_{\text{V DO Part \& Ref DO} \neq \text{given}} : n_{\text{V Part DO \& Ref DO} \neq \text{given}} \propto$
$n_{\text{V DO Part}} : n_{\text{V Part DO}}$

H_1: as H_0, but there is at least one "≠" instead of an "=".

In other words, you cannot simply say, "there are 2·2 = 4 cells and I assume each expected frequency is 397 divided by 4, i.e., approximately 100." If you did that, the upper row total would amount to nearly 200 – but that can't be right since there are only 150 cases of CONSTRUCTION: *VERB-OBJECT-PARTICLE*. Thus, you must include this information, that there are only 150 cases of CONSTRUCTION: *VERB-OBJECT-PARTICLE*, into the computation of the expected frequencies. The easiest way to do this is using percentages: there are $^{150}/_{397}$ cases of CONSTRUCTION: *VERB-OBJECT-PARTICLE* (i.e. 0.3778 = 37.78%). Then, there are $^{185}/_{397}$ cases of GIVENNESS: *GIVEN* (i.e., 0.466 = 46.6%). If the two variables are independent of each other, then the probability of their joint occurrence is 0.3778·0.466 = 0.1761. Since there are altogether 397 cases to which this probability applies, the expected frequency for this combination of variable levels is 397·0.1761 = 69.91. This logic can be reduced to (33).

(33) $$n_{\text{expected cell frequency}} = \frac{row\ sum \cdot column\ sum}{n}$$

If you apply this logic to every cell, you get Table 26.

Table 26. Expected construction frequencies of Peters (2001)

	GIVENNESS: *GIVEN*	GIVENNESS: *NEW*	Row totals
CONSTRUCTION: *V DO PART*	69.9	80.1	150
CONSTRUCTION: *V PART DO*	115.1	131.9	247
Column totals	185	212	397

You can immediately see that this table corresponds to the above H_0: the ratios of the values in each row and column are exactly those of the row totals and column totals respectively. For example, the ratio of 69.9 to 80.1 to 150 is the same as that of 115.1 to 131.9 to 247 and as that of 185 to 212 to 397, and the same is true in the other dimension. Thus, H_0 is not "all cell frequencies are identical" – it is "the ratios of the cell frequencies are equal (to each other and the respective marginal totals)."

This method to compute expected frequencies can be extended to arbitrarily complex frequency tables (see Gries 2009b: Section 5.1). But how do we test whether these deviate strongly enough from the observed frequencies? Thankfully, we do not need such complicated hypotheses but can use the simpler versions of $\chi^2 = 0$ and $\chi^2 > 0$ used above, and the chi-squared test for independence is identical to the chi-squared goodness-of-fit test you already know: for each cell, you compute a contribution to chi-squared and sum those up to get the chi-squared test statistic.

As before, the chi-squared test can only be used when its assumptions are met. The expected frequencies are large enough and for simplicity's sake we assume here that every subject only gave just one sentence so that the observations are independent of each other: for example, the fact that some subject produced a particular sentence on one occasion does then not affect any other subject's formulation. We can therefore proceed as above and compute (the sum of) the contributions to chi-squared on the basis of the same formula, here repeated as (34):

$$\text{(34)} \qquad \text{Pearson } \chi^2 = \sum_{i=1}^{n} \frac{\left(observed - expected\right)^2}{expected}$$

The results are shown in Table 27 and the sum of all contributions to chi-squared, chi-squared itself, is 9.82. However, we again need the num-

ber of degrees of freedom. For two-dimensional tables and when the expected frequencies are computed on the basis of the observed frequencies as here, the number of degrees of freedom is computed as shown in (35).[20]

Table 27. Contributions to chi-squared for the data of Peters (2001)

	GIVENNESS: *GIVEN*	GIVENNESS: *NEW*	Row totals
CONSTRUCTION: *V DO PART*	3.26	2.85	
CONSTRUCTION: *V PART DO*	1.98	1.73	
Column totals			9.82

(35) df = (no. of rows-1) · (no. of columns-1) = (2-1)·(2-1) = 1

With both the chi-squared and the *df-val*ue, you can look up the result in a chi-squared table (e.g., Table 28 below, which is the same as Table 21). As above, if the observed chi-squared value is larger than the one tabulated for p = 0.05 at the required *df-val*ue, then you can reject H_0. Here, chi-squared is not only larger than the critical value for p = 0.05 and df = 1, but also larger than the critical value for p = 0.01 and df = 1. But, since the chi-squared value is not also larger than 10.827, the actual p-value is somewhere between 0.01 and 0.001: the result is very, but not highly significant.

Table 28. Critical χ^2-values for $p_{\text{two-tailed}}$ = 0.05, 0.01, and 0.001 for $1 \leq df \leq 3$

	p = 0.05	p = 0.01	p = 0.001
df = 1	3.841	6.635	10.828
df = 2	5.991	9.21	13.816
df = 3	7.815	11.345	16.266

Fortunately, all this is much easier when you use R's built-in function. Either you compute just the p-value as before,

```
> pchisq(9.82, 1, lower.tail=FALSE)¶
[1] 0.001726243
```

20. In our example, the expected frequencies were computed from the observed frequencies in the marginal totals. If you compute the expected frequencies not from your observed data but from some other distribution, the computation of *df* changes to: *df* = (number of rows · number of columns)-1.

or you use the function `chisq.test` and do everything in a single step. The most important arguments for our purposes are:

- `x`: the two-dimensional table for which you do a chi-squared test;
- `correct=TRUE` or `correct=FALSE`; cf. above for the correction.[21]

```
> test.Peters<-chisq.test(Peters.2001, correct=FALSE)¶
> test.Peters¶
 Pearson's Chi-squared test
data:  Peters.2001
X-squared = 9.8191, df = 1, p-value = 0.001727
```

This is how you obtain expected frequencies or the chi-squared value:

```
> test.Peters$expected¶
            GIVENNESS
CONSTRUCTION     given       new
   V_DO_Part  69.89924  80.10076
   V_Part_DO 115.10076 131.89924
> test.Peters$statistic¶
X-squared
 9.819132
```

You now know that GIVENNESS is correlated with CONSTRUCTION, but you neither know yet how strong that effect is nor which variable level combinations are responsible for this result. As for the effect size, even though you might be tempted to use the size of the chi-squared value or the *p*-value to quantify the effect, you must not do that. This is because the chi-squared value is dependent on the sample size, as we can easily see:

```
> chisq.test(Peters.2001*10, correct=FALSE)¶
 Pearson's Chi-squared test
data:  Peters.2001 * 10
X-squared = 98.1913, df = 1, p-value < 2.2e-16
```

For effect sizes, this is of course a disadvantage since just because the sample size is larger, this does not mean that the relation of the values to each other has changed, too. You can easily verify this by noticing that the ratios of percentages, for example, have stayed the same. For that reason, the effect size is often quantified with a coefficient of correlation (called ϕ in the case of k×2/m×2 tables or Cramer's V for k×m tables with k or m >

21. For further options, cf. again `?chisq.test¶`. Note also what happens when you enter `summary(Peters.2001)¶`.

2), which falls into the range between 0 and 1 (0 = no correlation; 1 = perfect correlation) and is unaffected by the sample size. ϕ / Cramer's V is computed according to the formula in (36):

(36) ϕ / Cramer's V / Cramer's index I =

$$\sqrt{\frac{\chi^2}{n \cdot (\min[n_{rows}, n_{columns}] - 1)}}$$

In R, you can of course do this in one line of code:

```
> sqrt(test.Peters$statistic/
   sum(Peters.2001)*(min(dim(Peters.2001))-1))¶
X-squared
0.1572683
```

Given the theoretical range of values, this is a rather small effect size.[22] The correlation is probably not random, but also not strong.

Another measure of effect size, which can however only be applied to 2×2 tables, is the so-called odds ratio. An *odds ratio* tells you how the likelihood of one variable level changes in response to a change of the other variable's level. The *odds* of an event E correspond to the fraction in (37).

(37) $odds = \frac{p_E}{1 - p_E}$ (you get probabilities from odds with $\frac{odds}{1 + odds}$)

The odds ratio for a 2×2 table such as Table 23 is the ratio of the two odds (or 1 divided by that ratio, depending on whether you look at the event E or the event $\neg E$ (not E)), as in (38):

(38) *odds ratio* for Table 23 = $\frac{85}{100} \Big/ \frac{65}{147} = 1.9223$

In words, the odds of CONSTRUCTION: *V DO PART* are $(^{85}/_{185}) / (1-^{85}/_{185}) = {}^{85}/_{100} = 0.85$ when the referent of the direct object is given and $(^{65}/_{212}) / (1-{}^{65}/_{212}) = {}^{65}/_{147} = 0.4422$ when the referent of the direct object is new. This in

22. The theoretical range from 0 to 1 is really only possible in particular situations, but still a good heuristic to interpret this value.

turn means that CONSTRUCTION: *V DO PART* is $^{0.85}/_{0.4422} \approx 1.9223$ times more likely when the referent of the direct object is given than when it is not. From this, it also follows that the odds ratio in the absence of an interaction is ≈ 1.[23]

Table 27 also shows which variable level combinations contribute most to the significant correlation: the larger the contribution to chi-squared of a cell, the more that cell contributes to the overall chi-squared value; in our example, these values are all rather small – none exceeds the chi-squared value for $p = 0.05$ and $df = 1$, i.e., 3.841. In R, you can get the contributions to chi-squared as follows:

```
> test.Peters$residuals^2¶
             GIVENNESS
CONSTRUCTION    given      new
   V_DO_Part 3.262307 2.846825
   V_Part_DO 1.981158 1.728841
```

That is, you square the Pearson residuals. The Pearson residuals, which you obtain as follows, reveal the direction of effect for each cell: negative and positive values mean that observed values are smaller and larger than the expected values respectively.

```
> test.Peterst$residuals¶
             GIVENNESS
CONSTRUCTION     given       new
   V_DO_Part  1.806186 -1.687254
   V_Part_DO -1.407536  1.314854
```

Thus, if, given the small contributions to chi-square, one wanted to draw any further conclusions at all, then one could only say that the variable level combination contributing most to the significant result is the combination of CONSTRUCTION: *V DO PART* and GIVENNESS: *GIVEN*, which is more often observed than expected, but the individual cells' effects here are really rather small.

An interesting and revealing graphical representation is available with the function `assocplot`, whose most relevant argument is the two-

23. Often, you may find the logarithm of the odds ratio (see especially Section 5.3). When the two variables are not correlated, this log of the *odds ratio* is log 1 = 0, and positive/negative correlations result in positive/negative log odds ratios, which is often a little easier to interpret. For example, if you have two odds ratios such as $odds\ ratio_1 = 0.5$ and $odds\ ratio_2 = 1.5$, then you cannot immediately and intuitively see, which effect is larger. The logs of the odds ratios – log $odds\ ratio_1 = -0.693$ and log $odds\ ratio_2 = 0.405$ – tell you immediately the former is larger because it is further away from 0.

dimensional table under investigation: In this plot (Figure 44), "the area of the box is proportional to the difference in observed and expected frequencies." The black rectangles above the dashed lines indicate observed frequencies exceeding expected frequencies; grey rectangles below the dashed lines indicate observed frequencies smaller than expected frequencies; the heights of the boxes are proportional to the above Pearson residuals and the widths are proportional to the square roots of the expected frequencies. Note I do not just plot the table, but the transposed table – that's what the `t()` does. This is so that the row/column organization of the plot corresponds to that of the original table:

```
> assocplot(t(Peters.2001), col=c("black", "darkgrey"))¶
```

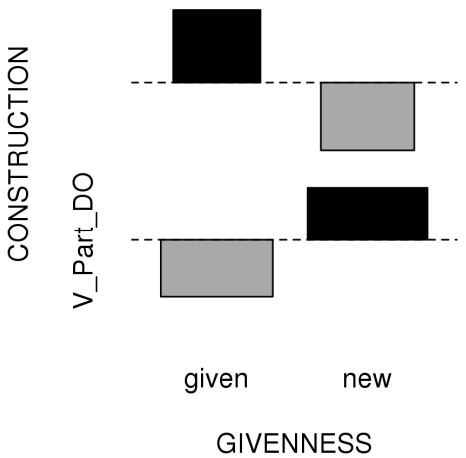

Figure 44. Association plot for CONSTRUCTION~GIVENNESS

Another interesting way to look at the data is a mixture between a plot and a table. The table/graph in Figure 45 has the same structure as Table 23, but (i) the sizes in which the numbers are plotted directly reflects the size of the residuals (i.e., bigger numbers deviate more from the expected frequencies than smaller numbers, where *bigger* and *smaller* are to be understood in terms of plotting size), and (ii) the coloring and the signs indicates how the observed frequencies deviate from the expected ones: black indicates positive residuals and grey indicates negative residuals. (For lack of a better term, I refer to this as a cross-tabulation plot.)

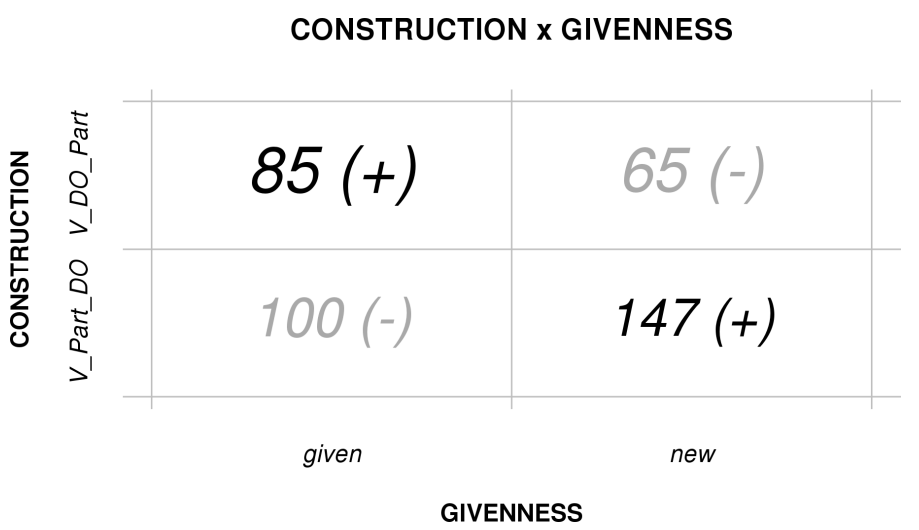

Figure 45. Cross-tabulation plot for CONSTRUCTION~GIVENNESS

This is how you would summarize all the results: "New objects are strongly preferred in the construction Verb-Particle-Direct Object and are dispreferred in Verb-Direct Object-Particle. The opposite kind of constructional preference is found for given objects. According to a chi-squared test for independence, this correlation is very significant ($\chi^2 = 9.82$; $df = 1$; $p_{\text{two-tailed}} < 0.002$), but the effect is not particularly strong ($\phi = 0.157$, odds ratio = 1.9223).

Let me finally emphasize that the above procedure is again the one providing you with a *p*-value for a two-tailed test. In the case of 2×2 tables, you can perform a one-tailed test as discussed in Section 4.1.1.2 above, but you cannot do one-tailed tests for tables with $df > 1$.

Recommendation(s) for further study

- the function `dotchart` as well as `mosaic` (from the library `vcd`) and `table.cont` (from the library `ade4`) for other kinds of plots
- the function `assocstats` (from the library `vcd`) for a different way to compute chi-square tests and effect sizes at the same time
- the function `CrossTable` (from the library `gmodels`) for more comprehensive tables
- the argument `simulate.p.value=TRUE` of the function `chisq.test` and the function `fisher.test`, which you can use when the expected frequencies are too small for a regular chi-squared test

- the Marascuilo procedure to test which observed row or column frequencies are different from each other in pairwise tests (cf. Gries to appear, who also discusses how to test a subtable out of a larger table)
- Crawley (2005: 85ff.), Crawley (2007: 301ff.), Sheskin (2011: Test 16)

Warning/advice
Again: never ever compute a chi-squared test on percentages – always on 'real' observed frequencies! (Trust me, there is a reason I repeat this …)

Let me mention one additional useful application of the chi-squared test (from Zar 1999: Section 23.4 and Sheskin 2011: 691ff.). Sometimes, you may have several isomorphic 2×2 tables on the same phenomenon, maybe because you found another source that discusses the same kind of data. You may then want to know whether or not the data are so similar that you can actually merge or amalgamate the data into one single data set. Here are the text hypotheses for that kind of question:

H_0: The trends in the different data sets do not differ from each other: $\chi^2_{\text{heterogeneity}} = 0$.
H_1: The trends in the different data sets differ from each other: $\chi^2_{\text{heterogeneity}} \neq 0$.

To explore this approach, let us compare Peters's data to those of Gries (2003a). You can enter the latter into R directly using the function `matrix`, which needs the vector of observed frequencies (columnwise), the number of columns, and the names of the dimensions (first rows, then columns):

```
> Gries.2003<-matrix(c(143, 53, 66, 141), ncol=2,
   dimnames=list(CONSTRUCTION=c("V_DO_Part",  "V_Part_DO"),
   GIVENNESS=c("given", "new")))¶
> Gries.2003¶
          given new
V_DO_Part   143  66
V_Part_DO    53 141
```

On the one hand, these data look very different from those of Peters (2001) because, here, when GIVENNESS is *GIVEN*, then CONSTRUCTION: *V_DO_PART* is nearly three times as frequent as CONSTRUCTION: *V_PART_DO* (and not in fact less frequent, as in Peters's data). On the other hand, the data are also similar because in both cases given direct objects increase the likelihood of CONSTRUCTION: *V_DO_PART*. A direct compari-

son of the association plots (not shown here, but you can use the following code to generate them) makes the data seem very much alike – how much more similar could two association plots be?

```
> par(mfrow=c(1, 2))¶
> assocplot(t(Peters.2001))¶
> assocplot(t(Gries.2003))¶
> par(mfrow=c(1, 1))¶
```

However, you should not really compare the sizes of the boxes in association plots – only the overall tendencies – so we turn to the heterogeneity chi-squared test. The heterogeneity chi-squared value is computed as the difference between the sum of chi-squared values of the original tables and the chi-squared value for the merged tables (that's why they have to be isomorphic), and it is evaluated with a number of degrees of freedom that is the difference between the sum of the degrees of freedom of all merged tables and the degrees of freedom of the merged table. Sounds pretty complex, but in fact it is not. The following code should make everything clear. First, you compute the chi-squared test for the data from Gries (2003a):

```
> test.Gries<-chisq.test(Gries.2003, correct=FALSE)¶
> test.Gries¶
  Pearson's Chi-squared test
data:  Gries.2003
X-squared = 68.0364, df = 1, p-value < 2.2e-16
```

Then you compute the sum of chi-squared values of the original tables:

```
> test.Peters$statistic+test.Gries$statistic¶
X-squared
[1] 77.85552
```

After that, you compute the chi-squared value of the combined table …

```
> chisq.test(Peters.2001+Gries.2003,
   correct=FALSE)$statistic¶
X-squared
[1] 65.87908
```

… and then the heterogeneity chi-squared and its degrees of freedom (you get the *df*-values with `$parameter`):

```
> het.chisq<-77.85552-65.87908 # 11.97644¶
> het.df<-1+1-1 # 1¶
```

How do you now get the *p*-value for these results?

THINK
BREAK

```
> pchisq(het.chisq, het.df, lower.tail=FALSE)¶
[1] 0.0005387742
```

The data from the two studies exhibit the same overall trend (given objects increase the likelihood of CONSTRUCTION: *V_DO_PART*) but they still differ highly significantly from each other ($\chi^2_{\text{heterogeneity}} = 11.98$; $df = 1$; $p_{\text{two-tailed}} < 0.001$). How can that be? Because of the different effect sizes: the odds ratio for Peters's data was 1.92, but in Gries's data it is nearly exactly three times as large, which is also what you would write in your results section; we will return to this example in Chapter 5.

```
> (143/66)/(53/141)¶
[1] 5.764151
```

1.2.3. One dep. variable (nominal/categorical) (dep. samples)

One central requirement of the chi-squared test for independence is that the tabulated data points are independent of each other. There are situations, however, where this is not the case, and in this section I discuss one method you can use on such occasions.

Let us assume you want to test whether metalinguistic knowledge can influence acceptability judgments. This is relevant because many acceptability judgments used in linguistic research were produced by the investigating linguists themselves, and one may well ask oneself whether it is really sensible to rely on judgments by linguists with all their metalinguistic knowledge instead of on judgments by linguistically naïve subjects. This is especially relevant since studies have shown that judgments by linguists, who after all think a lot about linguistic expressions, can deviate a lot from judgments by laymen, who usually don't (cf. Spencer 1973, Labov 1975, or Greenbaum 1976). In an admittedly oversimplistic case, you could ask 100 linguistically naïve native speakers to rate a sentence as 'acceptable' or 'unacceptable'. After the ratings have been made, you could tell the subjects which phenomenon the study investigated and which variable you

thought influenced the sentences' acceptability. Then, you would give the sentences back to the subjects to have them rate them once more. The question would be whether the subjects' newly acquired metalinguistic knowledge would make them change their ratings and, if so, how. This question involves

- a dependent nominal/categorical variable, namely BEFORE: *ACCEPTABLE* vs. BEFORE: *UNACCEPTABLE*;
- a dependent nominal/categorical variable, namely AFTER: *ACCEPTABLE* vs. AFTER: *UNACCEPTABLE*;
- dependent samples since every subject produced two judgments.

For such scenarios, you use the McNemar test (or Bowker test, cf. below). This test is related to the chi-squared tests discussed above in Sections 4.1.1.2 and 4.1.2.2 and involves the following procedure:

Procedure
- Formulating the hypotheses
- Computing the frequencies you would expect given H_0
- Testing the assumption(s) of the test:
  - the observed variable levels are related in a pairwise manner
  - the expected frequencies are ≥ 5
- Computing the test statistic χ^2, *df*, and *p*

First, the hypotheses:

H_0: The frequencies of the two possible ways in which subjects produce a judgment in the second rating task that differs from that in the first rating task are equal; $\chi^2 = 0$.

H_1: The frequencies of the two possible ways in which subjects produce a judgment in the second rating task that differs from that in the first rating task are not equal; $\chi^2 \neq 0$.

To get to know this test, we use the fictitious data summarized in Table 29, which you read in from the file <_inputfiles/04-1-2-3_accjudg.csv>. Table 29 suggests there has been a major change of judgments: Of the 100 rated sentences, only 31+17 = 48 sentences – not even half! – were judged identically in both ratings. But now you want to know whether the way in which the 52 judgments changed is significantly different from chance.

```
> AccBeforeAfter<-read.delim(file.choose())¶
> str(AccBeforeAfter); attach(AccBeforeAfter)¶
```

Table 29. Observed frequencies in a fictitious study on acceptability judgments

		AFTER		
		ACCEPTABLE	*INACCEPTABLE*	Row totals
BEFORE	*ACCEPTABLE*	31	39	70
	INACCEPTABLE	13	17	30
	Column totals	44	56	100

The McNemar test only involves those cases where the subjects changed their opinion, .i.e. cells *b* and *c* of the input table. If these are distributed equally, then the expected distribution of the 52 cases in which subjects change their opinion is that in Table 30.

Table 30. Expected frequencies in a fictitious study on acceptability judgments

		AFTER		
		ACCEPTABLE	*INACCEPTABLE*	Row totals
BEFORE	*ACCEPTABLE*		26	
	INACCEPTABLE	26		
	Column totals			

From this, you can see that both expected frequencies are larger than 5 so you can indeed do the McNemar test. As before, you compute a chi-squared value (using the by now familiar formula in (39)) and a *df-val*ue according to the formula in (40) (where *k* is the number of rows/columns):

$$(39) \qquad \chi^2 = \sum_{i=1}^{n} \frac{\left(observed - expected\right)^2}{expected} = 13$$

$$(40) \qquad df = \frac{k \cdot \left(k - 1\right)}{2} = 1$$

As before, you can look up this chi-squared value in the familiar kind of chi-square table and, again as before, if the computed chi-squared value is larger than the tabulated one for the relevant *df-val*ue for $p = 0.05$, you may reject H_0. As you can see, the chi-squared value is too large for H_0 and we accept H_1.

Table 31. Critical χ^2-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $1 \leq df \leq 3$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 1$	3.841	6.635	10.828
$df = 2$	5.991	9.21	13.816
$df = 3$	7.815	11.345	16.266

This is how you summarize this finding in the results section: "According to a McNemar test, the way 52 out of 100 subjects changed their judgments after they were informed of the purpose of the experiment is significantly different from chance: in the second rating task, the number of 'acceptable' judgments is much smaller ($\chi^2 = 13$; $df = 1$; $p_{\text{two-tailed}} < 0.001$)."

In R, this is again much easier. You need the function `mcnemar.test` and it typically requires two arguments:

- `x`: a two-dimensional table which you want to test;
- `correct=FALSE` or `correct=TRUE` (the default): when the number of changes < 30, then som recommend the continuity correction.

```
> mcnemar.test(table(BEFORE, AFTER), correct=FALSE)¶
 McNemar's Chi-squared test
data:  table(BEFORE, AFTER)
McNemar's chi-squared = 13, df = 1, p-value = 0.0003115
```

The summary and conclusions are of course the same. When you do this test for $k \times k$ tables (with $k > 2$), this test is sometimes called Bowker test.

Recommendation(s) for further study

- Sheskin (2011: Test 20) on the McNemar test, its exact alternative, which you can compute with `dbinom`
- Sheskin (2011: Test 26) for Cochran's extension of the McNemar test to test three or more measurements of a dichotomous variable, which takes only a few lines of code to compute in R – why don't you try to write such a function?
- the function `runs.test` (from the library `tseries`) to test the randomness of a binary sequence

2. Dispersions

Sometimes, it is necessary and/or interesting to not just look at the general

characteristics of a distribution but also at more narrowly defined distributional characteristics. The two most obvious characteristics are the dispersion and the central tendency of a distribution. This section is concerned with the dispersion – more specifically, the variance or standard deviation – of a variable; Section 4.3 discusses measures of central tendency.

For some research questions, it is useful to know, for example, whether two distributions have the same or a similar dispersion. Put differently, do two distributions spread around their means in a similar or in a different way? We touched upon this topic a little earlier in Section 3.1.3.6, but to illustrate the point once more, consider Figure 46.

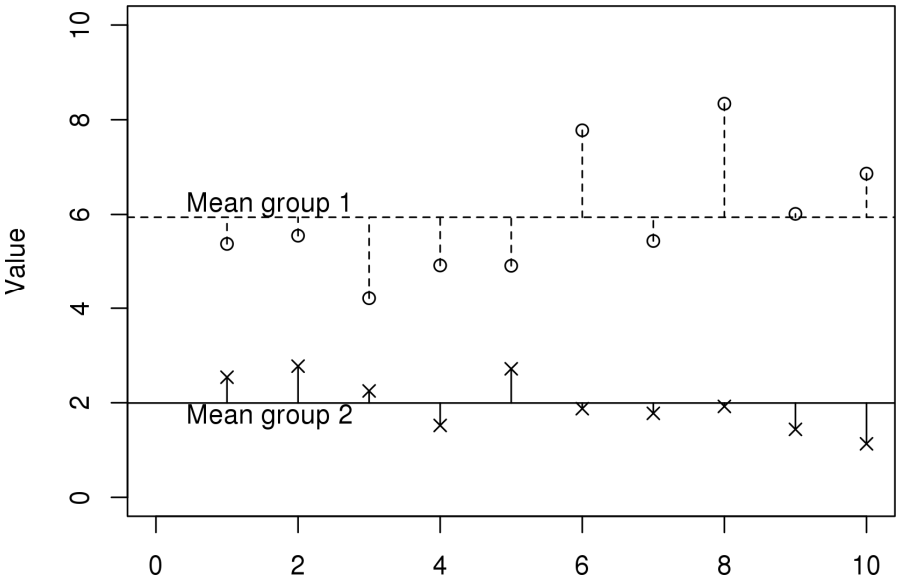

Figure 46. Two fictitious distributions

Figure 46 shows two distributions, one group of 10 values (represented by unfilled circles) and another group of 10 values (represented by crosses). The means of these groups are shown with the two horizontal lines (dashed for the first group), and the deviations of each point from its group mean are shown with the vertical lines. As you can easily see, the groups do not just differ in terms of their means ($mean_{\text{group 2}} = 1.99$; $mean_{\text{group 1}} = 5.94$), but also in terms of their dispersion: the deviations of the points of group 1 from their mean are much larger than their counterparts in group 2. While this difference is obvious in Figure 46, it can be much harder to discern in other cases, which is why we need a statistical test. In Section 4.2.1,

we discuss how you test whether the dispersion of one dependent interval/ratio-scaled variable is significantly different from a known dispersion value. In Section 4.2.2, we discuss how you test whether the dispersion of one dependent ratio-scaled variable differs significantly in two groups.

2.1. Goodness-of-fit test for one dep. variable (ratio-scaled)

As an example for this test, we return to the above data on first language acquisition of Russian tense-aspect patterning. In Section 4.1.1.1 above, we looked at how the correlation between the use of tense and aspect of one child developed over time. Let us assume, you now want to test whether the overall variability of the values for this child is significantly different from that of another child for whom you already have data. Let us also assume that for this other child you found a variance of 0.025.

This question involves the following variables and is investigated with a chi-squared test as described below:

- a dependent ratio-scaled variable, namely the variable TENSEASPECT, consisting of the Cramer's *V* values;
- no independent variable because you are not testing whether the distribution of the variable TENSEASPECT is influenced by, or correlated with, something else.

Procedure
- Formulating the hypotheses
- Computing descriptive statistics
- Testing the assumption(s) of the test: the population from which the sample whose variance is tested has been drawn or at least the sample itself from which the variance is computed is normally distributed
- Computing the test statistic χ^2, *df*, and *p*

As usual, you begin with the hypotheses:

H_0: The variance of the data for the newly investigated child does not differ from the variance of the child investigated earlier; sd^2 TENSEASPECT of the new child = sd^2 TENSEASPECT of the already investigated child, or sd^2 of the new child = 0.025, or the ratio of the two variances is 1.

H_1: The variance of the data for the newly investigated child differs

from the variance of the child investigated earlier; sd^2 TENSEASPECT of the new child ≠ sd^2 TENSEASPECT of the already investigated child, or sd^2 of the new child ≠ 0.025, or the ratio of the two variances is not 1.

You load the data from <_inputfiles/04-2-1_tense-aspect.csv>.

```
> RussianTensAsp<-read.delim(file.choose())¶
> str(RussianTensAsp); attach(RussianTensAsp)¶
```

As a next step, you must test whether the assumption of this chi-squared test is met and whether the data are in fact normally distributed. We have discussed this in detail above so we run the test here without further ado.

```
> shapiro.test(TENSE_ASPECT)¶
 Shapiro-Wilk normality test
data:   TENSE_ASPECT
W = 0.9942, p-value = 0.9132
```

Just like in Section 4.1.1.1 above, you get a *p*-value of 0.9132, which means you must not reject H_0, you can consider the data to be normally distributed, and you can compute this chi-squared test. You first compute the sample variance that you want to compare to the previous results:

```
> var(TENSE_ASPECT)¶
[1] 0.01687119
```

To test whether this value is significantly different from the known variance of 0.025, you compute a chi-squared statistic as in formula (41).

$$\chi^2 = \frac{(n-1)\cdot sample\ \ variance}{population\ \ variance} \tag{41}$$

This chi-squared value has *n*-1 = 116 degrees of freedom. In R:

```
> chi.squared<-((length(TENSE_ASPECT)-1)*var(TENSE_ASPECT))/
   0.025¶
> chi.squared¶
[1] 78.28232
```

As usual, you can create those critical values yourself or you look up this chi-squared value in the familiar kind of table.

```
> qchisq(c(0.05, 0.01, 0.001), 116, lower.tail=FALSE)¶
```

Table 32. Critical χ^2-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $115 \leq df \leq 117$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 115$	141.03	153.191	167.61
$df = 116$	142.138	154.344	168.813
$df = 117$	143.246	155.496	170.016

Since the obtained value of 78.28 is much smaller than the relevant critical value of 142.138, the difference between the two variances is not significant. You can compute the exact *p*-value as follows:

```
> pchisq(chi.squared, (length(TENSE_ASPECT)-1), lower.tail=
   FALSE)¶
[1] 0.9971612¶
```

This is how you would summarize the result: "According to a chi-squared test, the variance of the newly investigated child (0.017) does not differ significantly from the variance of the child investigated earlier (0.025): $\chi^2 = 78.28$; $df = 116$; $p_{\text{two-tailed}} > 0.05$."

2.2. One dep. variable (ratio-scaled) and one indep. variable (nominal)

The probably more frequent scenario in the domain 'testing dispersions' is the case where you test whether two samples or two variables exhibit the same dispersion (or at least two dispersions that do not differ significantly). Since the difference of dispersions or variances is probably not a concept you spent much time thinking about so far, let us look at one illustrative example from the domain of sociophonetics. Gaudio (1994) studied the pitch range of heterosexual and homosexual men. At issue was therefore not the average pitch, but its variability, a good example for how variability as such can be interesting. In that study, four heterosexual and four homosexual men were asked to read aloud two text passages and the resulting recordings were played to 14 subjects who were asked to guess which speakers were heterosexual and which were homosexual. Interestingly, the subjects were able to distinguish the sexual orientation nearly perfectly. The only (insignificant) correlation which suggested itself as a possible explanation was that the homosexual men exhibited a wider pitch range in

one of the text types, i.e., a result that has to do with variability/dispersion.

We will now look at an example from second language acquisition. Let us assume you want to study how native speakers of a language and very advanced learners of that language differed in a synonym-finding task in which both native speakers and learners are presented with words for which they are asked to name synonyms. You may now not be interested in the exact numbers of synonyms – maybe, the learners are so advanced that these are actually fairly similar in both groups – but in whether the learners exhibit more diversity in the amounts of time they needed to come up with all the synonyms they can name. This question involves

- a dependent ratio-scaled variable, namely SYNTIMES, the time subjects needed to name the synonyms;
- a nominal/categorical independent variable, namely SPEAKER: *LEARNER* and SPEAKER: *NATIVE*.

This kind of question is investigated with the so-called *F*-test for homogeneity of variances, which involves the following steps:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test:
  - the population from which the sample whose variance is tested has been drawn or at least the sample itself from which the variance is computed is normally distributed
  - the samples are independent of each other
- Computing the test statistic F, df_1 and df_2, and p

First, you formulate the hypotheses. Note that H_1 is non-directional / two-tailed.

H_0: The times the learners need to name the synonyms they can think of are not differently variable from the times the native speakers need to name the synonyms they can think of; the ratio of the variances $F = 1$.

H_1: The times the learners need to name the synonyms they can think of are differently variable from the times the native speakers need to name the synonyms they can think of; the ratio of the variances $F \neq 1$.

As an example, we use the (fictitious) data in <_inputfiles/04-2-2_synonymtimes.csv>:

```
> SynonymTimes<-read.delim(file.choose())¶
> str(SynonymTimes); attach(SynonymTimes)¶
```

You compute the variances for both subject groups and plot the data into Figure 47. The variability of the two groups seem very similar: the boxes have quite similar sizes, but the ranges of the whiskers differ a bit; cf. the code file for some additional exploration with more precise ecdf plots.

```
> tapply(SYNTIMES, SPEAKER, var)¶
  Learner   Native
10.31731 14.15385
> boxplot(SYNTIMES~SPEAKER, notch=TRUE)¶
> rug(jitter(SYNTIMES), side=2)¶
```

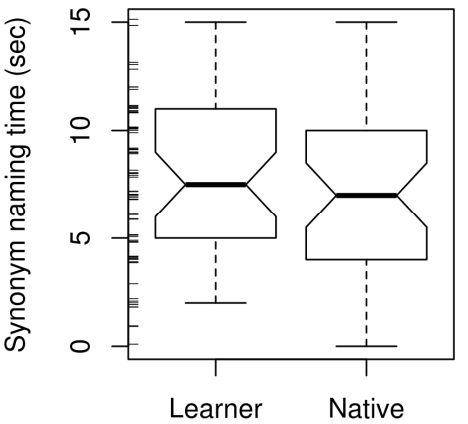

Figure 47. Boxplot for SYNTIMES~SPEAKER

The *F*-test requires a normal distribution of the population or at least the sample. We again use the Shapiro-Wilk test, this time with `tapply`. Nothing to worry about: both samples do not deviate significantly from normality and you can do an *F*-test. This test requires you to compute the quotient of the two variances (traditionally, but not necessarily – see below – the larger variance is used as the numerator). Now we compute the ratio of the two variances, which turns out to be not 1, but somewhat close to it.

```
>tapply(SYNTIMES, SPEAKER, shapiro.test)¶
$Learner
```

```
        Shapiro-Wilk normality test
data:  X[[1L]]
W = 0.9666, p-value = 0.2791
$Native
        Shapiro-Wilk normality test
data:  X[[2L]]
W = 0.9751, p-value = 0.5119

> F.value<-var(SYNTIMES[SPEAKER=="Native"])/
   var(SYNTIMES[SPEAKER=="Learner"]); F.value¶
[1] 1.371855
```

To see whether this value is significantly different from 1, you again need to consider degrees of freedom, this time even two: one for the numerator, one for the denominator. Both can be computed very easily by just subtracting 1 from the sample sizes (of the samples for the variances); cf. the formula in (42).

(42) $df_{\text{numerator}} = n_{\text{numerator sample}}\text{-}1$; $df_{\text{denominator}} = n_{\text{denominator sample}}\text{-}1$

You get 39 in both cases and can look up the result in an *F*-table.

Table 33. Critical *F*-values for $p_{\text{two-tailed}} = 0.05$ and $38 \leq df_{1,\,2} \leq 40$

	$df_2 = 38$	$df_2 = 39$	$df_2 = 40$
$df_1 = 38$	1.907	1.8963	1.8862
$df_1 = 39$	1.9014	1.8907	1.8806
$df_1 = 40$	1.8961	1.8854	1.8752

Obviously, the result is not significant: the computed *F*-value is smaller than the tabulated one for $p = 0.05$ (which is 1.8907). As usual, you can compute the critical *F*-values yourself, and you would have to use the function `qf` for that. We need four arguments:

- `p`: the *p*-value for which you want to determine the critical *F*-value (for some *df*-values);
- `df1` and `df2`: the two *df*-values for the *p*-value for which you want to determine the critical *F*-value;
- the argument `lower.tail=FALSE`, to instruct R to only consider the area under the curve above / to the right of the relevant *F*-value.

There is one last thing, though. When we discussed one- and two-tailed tests in Section 1.3.4 above, I mentioned that in the graphical representa-

tion of one-tailed tests (cf. Figure 6 and Figure 8) you add the probabilities of the events you see when you move away from the expectation of H_0 in *one* direction while in the graphical representation of two-tailed tests (cf. Figure 7 and Figure 9) you add the probabilities of the events you see when you move away from the expectation of H_0 in *both* directions. The consequence of that was that the prior knowledge that allowed you to formulate a directional H_1 was rewarded such that you needed a less extreme finding to get a significant result. This also means, however, that when you want to compute a two-tailed *p*-value using `lower.tail=FALSE`, then you need the *p*-value for $^{0.05}/_{2} = 0.025$. This value tells you which *F*-value cuts off 0.025 on only one side of the graph (say, the right one), but since a two-tailed test requires that you cut off the same area on the other/left side as well, this means that this is also the desired critical *F*-value for $p_{\text{two-tailed}} = 0.05$. Figure 48 illustrates this logic:

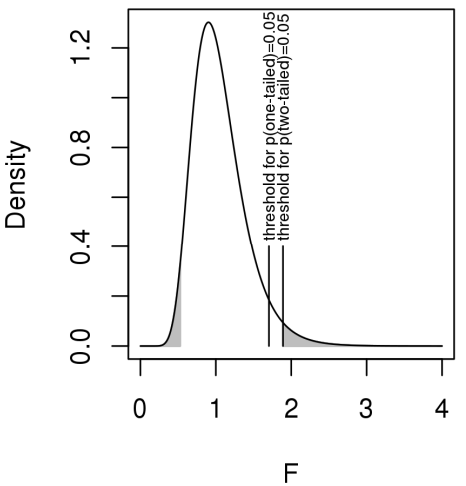

Figure 48. Density function for an *F*-distribution with $df_1 = df_2 = 39$, two-tailed test

As mentioned above, the expectation from H_0 is that $F = 1$. The right vertical line indicates the *F*-value you need to obtain for a significant two-tailed test with $df_{1,\,2} = 39$; this *F*-value is the one you already know from Table 33 – 1.8907 – which means you get a significant two-tailed result if either one of the variances is 1.8907 times larger than the other. The left vertical line indicates the *F*-value you need to obtain for a significant one-tailed test with $df_{1,\,2} = 39$; this *F*-value is 1.7045, which means you get a

significant one-tailed result if the variance you predict to be larger (!) is 1.7045 times larger than the one you predict to be smaller. To compute the *F*-values for the two-tailed tests yourself, as a beginner you may want to enter just these lines and proceed in a similar way for all other cells in Table 33, and the code file contains code to generate all of Table 33.

```
> qf(0.025, 39, 39, lower.tail=TRUE)¶
[1] 0.5288993
> qf(0.025, 39, 39, lower.tail=FALSE)¶
[1] 1.890719
```

The observed *F*-value is obviously too small for either a directional or a non-directional significant result: 1.53 < 1.89. It is more useful, however, to immediately compute the *p*-value for your *F*-value. Since you now use the reverse of `qf`, `pf`, you must now not divide but multiply by 2:

```
> 2*pf(F.value, 39, 39, lower.tail=FALSE)¶
[1] 0.3276319
```

As we've seen, with a *p*-value of $p = 0.3276$, the *F*-value of about 1.37 for $df_{1, 2} = 39$ is obviously not significant. The function for the *F*-test in R that easily takes care of all of the above is called `var.test` and it requires at least two arguments, the two samples. Just like many other functions, you can approach this in two ways: you can provide R with a formula,

```
> var.test(SYNTIMES~SPEAKER)¶
 F test to compare two variances
data:  SYNTIMES by SPEAKER
F = 0.7289, num df = 39, denom df = 39, p-value = 0.3276
alternative hypothesis: true ratio of variances is not
   equal to 1
95 percent confidence interval:
 0.385536 1.378221
sample estimates:
ratio of variances
         0.7289402
```

or you can use a vector-based alternative:

```
> var.test(SYNTIMES[SPEAKER=="Learner"],
   SYNTIMES[SPEAKER=="Native"])¶
```

Don't be confused if the *F*-value you get from R is not the same as the one you computed yourself. Barring mistakes, the value outputted by R is

then $^1/_F$-value – R does not automatically put the larger variance into the numerator, but the variance whose name comes first in the alphabet, which here is "Learner" (before "Native"). The *p*-value then shows you that R's result is the same as yours. You can now sum this up as follows: "The native's synonym-finding times exhibit a variance that is approximately 40% larger than that of the learners (14.15 vs. 10.32), but according to an *F*-test, this difference is not significant: $F = 0.73$; $df_{\text{learner}} = 39$; $df_{\text{native}} = 39$; $p_{\text{two-tailed}} = 0.3276$."

Recommendation(s) for further study

- Dalgaard (2002: 89), Crawley (2007: 289ff.), Baayen (2008: Section 4.2.3), Sheskin (2011: Tests 3, 11a)
- the function `fligner.test` to test the homogeneity of variance when the data violate the assumption of normality
- Good and Hardin (2012: 100ff.) for other (advanced!) possibilities to compare variances
- see the code file for a function `exact.f.test.indep` that I wrote to compute an exact version of this *F*-test, which you can use when your sample sizes are very small (maybe <15); careful, this test may take quite some time

3. Means

The probably most frequent use of simple significance tests apart from chi-squared tests are tests of differences between means. In Section 4.3.1, we will be concerned with goodness-of-fit tests, i.e., scenarios where you test whether an observed measure of central tendency is significantly different from another already known mean (recall this kind of question from Section 3.1.5.1); in Section 4.3.2, we then turn to tests where measures of central tendencies from two samples are compared to each other.

3.1. Goodness-of-fit tests

3.1.1. One dep. variable (ratio-scaled)

Let us assume you are again interested in the use of hedges. Early studies suggested that men and women exhibit different communicative styles with regard to the frequency of hedges (and otherwise). Let us also assume you

knew from the literature that female subjects in experiments used on average 12 hedges in a two-minute conversation with a female confederate of the experimenter. You also knew that the frequencies of hedges are normally distributed. You now did an experiment in which you recorded 30 two-minute conversations of female subjects with a male confederate and counted the same kinds of hedges as were counted in the previous studies (and of course we assume that with regard to all other parameters, your experiment was an exact replication of the earlier one). You now want to test whether the average number of hedges in your experiment is significantly different from the value of 12 reported in the literature. This question involves

- a dependent ratio-scaled variable, namely HEDGES, which will be compared to the value from the literature;
- no independent variable since you do not test whether HEDGES is influenced by something else.

For such cases, you use a one-sample *t*-test, which involves these steps:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics
- Testing the assumption(s) of the test: the population from which the sample whose mean is tested has been drawn or at least the sample itself from which the mean is computed is normally distributed
- Computing the test statistic *t*, *df*, and *p*

As always, you begin with the hypotheses:

H_0: The average of HEDGES in the conversations of the subjects with the male confederate does not differ significantly from the already known average; hedges in your experiment = 12, or hedges in your experiment-12 = 0, or $t = 0$;

H_1: The average ofHEDGES in the conversations of the subjects with the male confederate differs from the previously reported average; hedges in your experiment ≠ 12, or hedges in your experiment-12 ≠ 0, $t \neq 0$.

Then you load the data from <_inputfiles/04-3-1-1_hedges.csv>:

```
> Hedges<-read.delim(file.choose())¶
> str(Hedges); attach(Hedges)¶
```

Next, you compute the mean frequency of hedges you found in your experiment as well as a measure of dispersion (cf. the code file for a graph):

```
> mean(HEDGES); sd(HEDGES)¶
[1] 14.83333
[1] 2.506314
```

While the literature mentioned that the numbers of hedges are normally distributed, you test whether this holds for your data, too:

```
> shapiro.test(HEDGES)¶
 Shapiro-Wilk normality test
data:  HEDGES
W = 0.946, p-value = 0.1319
```

It does. You can therefore immediately proceed to the formula in (43).

$$t = \frac{\bar{x}_{sample} - \bar{x}_{population}}{sd_{sample} \Big/ \sqrt{n_{sample}}} \tag{43}$$

```
> (mean(HEDGES)-12) / (sd(HEDGES)/sqrt(length(HEDGES)))¶
[1] 6.191884
```

To see what this value means, we need degrees of freedom again. Again, this is easy here since $df = n\text{-}1$, i.e., $df = 29$. When you look up the t-value for $df = 29$ in the usual kind of table, the t-value you computed must again be larger than the one tabulated for your df at $p = 0.05$. To compute the critical p-value, you use `qt` with the p-value and the required df-value. Since you do a two-tailed test, you must cut off $^{0.05}/_{2} = 2.5\%$ on both sides of the distribution, which is illustrated in Figure 49.

Table 34. Critical t-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $28 \leq df \leq 30$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 28$	2.0484	2.7633	3.6739
$df = 29$	2.0452	2.7564	3.6594
$df = 30$	2.0423	2.75	3.646

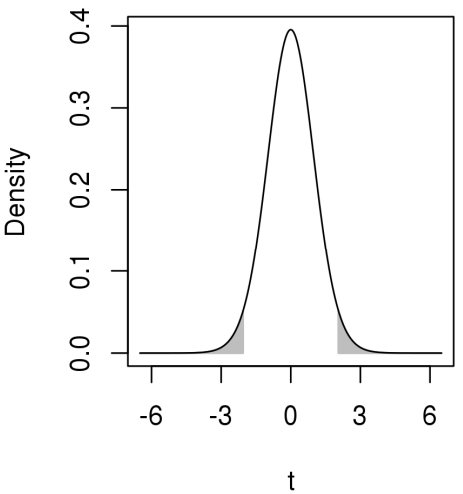

Figure 49. Density function for a *t*-distribution for *df* = 29, two-tailed test

The critical *t*-value for $p = 0.025$ and $df = 29$ is therefore:

```
> qt(c(0.025, 0.975), 29, lower.tail=FALSE)¶
[1]  2.045230 -2.045230
```

The exact *p*-value can be computed with `pt` and the obtained *t*-value is highly significant: 6.1919 is not just larger than 2.0452, but even larger than the *t*-value for $p = 0.001$ and $df = 29$. You could also have guessed that because the *t*-value of 6.19 is far in the right grey margin in Figure 49.

```
> 2*pt(6.191884, 29, lower.tail=FALSE)¶
[1] 9.42153e-07
```

To sum up: "On average, female subjects that spoke to a male confederate of the experimenter for two minutes used 14.83 hedges (standard deviation: 2.51). According to a one-sample *t*-test, this average is highly significantly larger than the value previously noted in the literature (for female subjects speaking to a female confederate of the experimenter): $t = 6.1919$; $df = 29$; $p_{\text{two-tailed}} < 0.001$."

With the right function in R, you need just one line. The relevant function is called `t.test` and requires the following arguments:

- `x`: a vector with the sample data;
- `mu=`..., the population mean to which the sample mean of `x` is compared;

– `alternative="two-sided"` for two-tailed tests (the default) or one of `alternative="greater"` or `alternative="less"`, depending on which H_1 you want to test: the value you assign to `alternative` states the relation of the sample mean to the population mean.

```
> t.test(HEDGES, mu=12)¶
 One Sample t-test
data:  HEDGES
t = 6.1919, df = 29, p-value = 9.422e-07
alternative hypothesis: true mean is not equal to 12
95 percent confidence interval:
 13.89746 15.76921
sample estimates:
mean of x
 14.83333
```

You get the already known mean of 14.83 as well as the *df*- and *t*-value we computed semi-manually. In addition, we get the exact *p*-value and the confidence interval of the mean which does not include the value of 12.

Recommendation(s) for further study
Baayen (2008: Section 4.1.2), Sheskin (2011: Test 2)

3.1.2. One dep. variable (ordinal)

In the previous section, we discussed a test that allows you to test whether the mean of a sample from a normally-distributed population is different from an already known population mean. This section deals with a test you can use when the data violate the assumption of normality or when they are not interval-/ratio-scaled to begin with. We will explore this test by looking at an interesting little morphological phenomenon, namely subtractive word-formation processes in which parts of usually two source words are merged into a new word. Two such processes are blends and complex clippings; some well-known examples of the former are shown in (44a), while (44b) provides a few examples of the latter; in all examples, the letters of the source words that enter into the new word are underlined.

(44) a. *brunch* (*br*eakfast × l*unch*), *motel* (*mot*or × h*otel*), *smog* (*smo*ke × *fog*), *foolosopher* (*fool* × *philosopher*)
b. *scifi* (*sci*ence × *fi*ction), *fedex* (*fede*ral × *ex*press), *sysadmin* (*system* × *admin*istrator)

One question that may arise upon looking at these coinages is to what degree the formation of such words is supported by some degree of similarity of the source words. There are many different ways to measure the similarity of words, and the one we are going to use here is the so-called Dice coefficient (cf. Brew and McKelvie 1996). You can compute a Dice coefficient for two words in two simple steps. First, you split the words up into letter (or phoneme or …) bigrams. For *motel* (*motor* × *hotel*) you get:

- *motor*: *mo*, *ot*, *to*, *or*;
- *hotel*: *ho*, *ot*, *te*, *el*.

Then you count how many of the bigrams of each word occur in the other word, too. In this case, these are two: the *ot* of *motor* also occurs in *hotel*, and thus the *ot* of *hotel* also occurs in *motor*.[24] This number, 2, is divided by the number of bigrams to yield the Dice coefficient:

(45) $Dice_{motor\ \&\ hotel} = {}^{2}/_{8} = 0.25$

In other words, the Dice coefficient is the percentage of shared bigrams out of all bigrams (and hence ratio-scaled). We will now investigate the question of whether source words that entered into subtractive word-formation processes are more similar to each other than words in general are similar to each other. Let us assume, you know that the average Dice coefficient of randomly chosen words is 0.225 (with a standard deviation of 0.0809; the median is 0.151 with an interquartile range of 0.125). These figures already suggest that the data may not be normally distributed.[25]

This study involves

- a dependent ratio-scaled variable, namely the SIMILARITY of the source words, which will be compared with the already known mean/median;
- no independent variable since you do not test whether SIMILARITY is influenced by something else.

The hypotheses should be straightforward:

24. In R, such computations can be easily automated and done for hundreds of thousands of words. For example, if the vector `a` contains a word, this line returns all its bigrams: `substr(rep(a, nchar(a)-1), 1:(nchar(a)-1), 2:(nchar(a)))¶`; for many such applications, cf. Gries (2009a).
25. For authentic data, cf. Gries (2006), where I computed Dice coefficients for all 499,500 possible pairs of 1,000 randomly chosen words.

H_0: The average of SIMILARITY for the source words that entered into subtractive word-formation processes is not significantly different from the known average of randomly chosen word pairs; Dice coefficients of source words = 0.225, or Dice coefficients of source words-0.225 = 0.

H_1: The average of SIMILARITY for the source words that entered into subtractive word-formation processes is different from the known average of randomly chosen word pairs; Dice coefficients of source words ≠ 0.225, or Dice coefficients of source words-0.225 ≠ 0.

The data to be investigated here are in <_inputfiles/04-3-1-2_dices.csv>; they are data of the kind studied in Gries (2006).

```
> Dices<-read.delim(file.choose())¶
> str(Dices); attach(Dices)¶
```

From the summary statistics, you could already infer that the similarities of randomly chosen words are not normally distributed. We can therefore assume that this is also true of the sample of source words, but of course you also test this assumption (cf. the code file for a plot):

```
> shapiro.test(DICE)¶
 Shapiro-Wilk normality test
data:  DICE
W = 0.9615, p-value = 0.005117
```

The Dice coefficients are not normally, but symmetrically distributed (as you can also clearly see in the ecdf plot). Thus, even though Dice coefficients are ratio-scaled and although the sample size is >30, you may want to be careful and not use the one-sample *t*-test but, for example, the so-called one-sample sign test for the median, which involves these steps:

Procedure
Formulating the hypotheses
Computing the frequencies of the signs of the differences between the observed values and the expected average
Computing the probability of error *p*

You first rephrase the hypotheses; I only provide new statistical ones:

H_0: $median_{\text{Dice coefficients of your source words}} = 0.151$.

H_1: $median_{\text{Dice coefficients of your source words}} \neq 0.151$.

Then, you compute descriptive statistics: the median and its interquartile range. Obviously, the observed median Dice coefficient is a bit higher than 0.151, the median Dice coefficient of the randomly chosen word pairs, but it is impossible to guess whether the difference is going to be significant.

```
> median(DICE); IQR(DICE)¶
[1] 0.1775
[1] 0.10875
```

For the one-sample sign test, you first determine how many observations are above and below the expected median, because if the expected median was a good characterization of the observed data, then 50% of the observed data should be above the expected median and 50% should be below it. (NB: you must realize that this means that the exact sizes of the deviations from the expected median are not considered here – you only look at whether the observed values are larger or smaller than the expected median, but not how much larger or smaller.)

```
> sum(DICE>0.151); sum(DICE<0.151)¶
[1] 63
[1] 37
```

63 of the 100 observed values are larger than the expected median (the rest is smaller than the expected median) – since you expected 50, it seems as if the Dice coefficients observed in your source words are significantly larger than those of randomly chosen words. As before, this issue can also be approached graphically, using the logic and the function `dbinom` from Section 1.3.4.1, Figure 7. Figure 50 shows the probabilities of all possible results you can get in 100 trials – because you look at the Dice coefficients of 100 subtractive formations. First, consider the left panel of Figure 50.

According to H_0, you would expect 50 Dice coefficients to be larger than the expected median, but you found 63. Thus, you add the probability of the observed result (the black bar for 63 out of 100) to the probabilities of all those that deviate from H_0 even more extremely, i.e., the chances to find 64, 65, …, 99, 100 Dice coefficients out of 100 that are larger than the expected median. These probabilities from the left panel sum up to approximately 0.006.

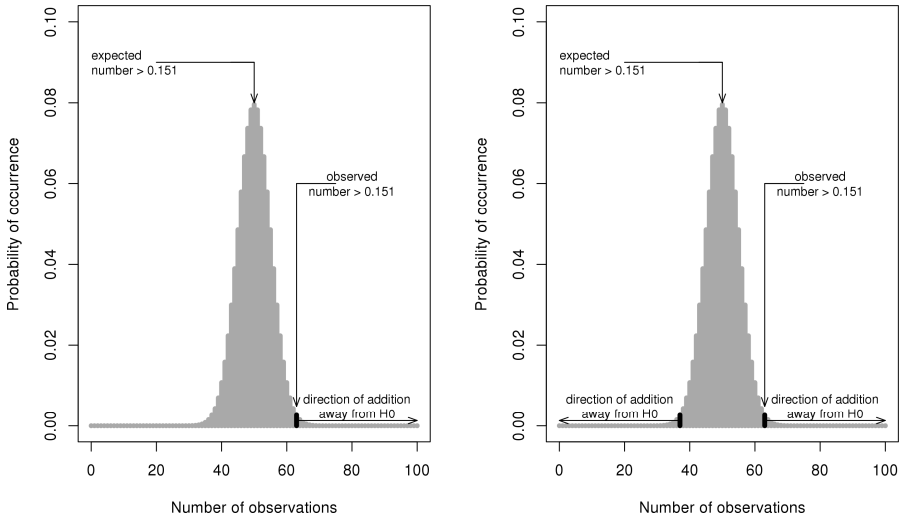

Figure 50. Probability distributions for 100 binomial trials test

```
> sum(dbinom(63:100, 100, 0.5))¶
[1] 0.006016488
```

But you are not finished yet … As you can see in the left panel of Figure 50, so far you only include the deviations from H_0 in one direction – but your H_1 is non-directional, i.e., two-tailed. You must therefore also include the probabilities of the events that deviate just as much and more from H_0 in the other direction: 37, 36, …, 1, 0 Dice coefficients out of 100 that are smaller than the expected median, as represented in the right panel of Figure 50. The probabilities sum up to the same value (because the distribution of binomial probabilities around $p = 0.5$ is symmetric).

```
> sum(dbinom(37:0, 100, 0.5))¶
[1] 0.006016488
```

Again: if you expect 50 out of 100, but observe 63 out of 100, and want to do a two-tailed test, you must add the summed probability of finding 63 to 100 larger Dice coefficients (the upper/right 38 probabilities) to the summed probability of finding 0 to 37 smaller Dice coefficients (the lower/left 38 probabilities). The $p_{\text{two-tailed}}$-value of 0.01203298 you then get is significant. You can sum up: "The investigation of 100 subtractive word formations resulted in an average source-word similarity of 0.1775 (median, *IQR* = 0.10875). 63 of the 100 source words were more similar to each

other than expected from random word pairs, which, according to a two-tailed sign test is a significant deviation from the average similarity of random word pairs (median =0.151, *IQR* range = 0.125): $p_{\text{binomial}} = 0.012$."

Recall that this one-sample sign test only uses nominal information, whether each data point is larger or smaller than the expected reference median. If the distribution of the data is rather symmetrical – as it is here – then there is an alternative test that also takes the sizes of the deviations into account, i.e. uses at least ordinal information. This so-called one-sample signed-rank test can be computed using the function `wilcox.test`. Apart from the vector to be tested, the following arguments are relevant:

- `alternative`: a character string saying which H_1 you want to test: the default is `"two.sided"`, other possible values for one-tailed tests are `"less"` or `"greater"`, which specify how the first-named vector relates to the specified reference median;
- `mu=...`: the reference median expected according to H_0;
- `exact=TRUE`, if you want to compute an exact test (only when your sample size is smaller than 50 and there are no ties) or `exact=FALSE`, if an asymptotic test is sufficient; the default amounts to the latter;
- `correct=TRUE` (the default) for a continuity correction or `correct=FALSE` for none;
- `conf.level`: a value between 0 and 1 specifying the size of the confidence interval; the default is 0.95.

Since you have a non-directional H_1, you do a two-tailed test by simply adopting the default setting for `alternative`:

```
> wilcox.test(DICE, mu=0.151, correct=FALSE)¶
 Wilcoxon signed rank test
data:  DICE
V = 3454.5, p-value = 0.001393
alternative hypothesis: true location is not equal to 0.151
```

The test confirms the previous result: both the one-sample sign test, which is only concerned with the directions of deviations, and the one-sample signed rank test, which also considers the sizes of these deviations, indicate that the source words of the subtractive word-formations are more similar to each other than expected from random source words. This should however, encourage you to make sure you formulate exactly the hypothesis you are interested in (and then use the required test).

Recommendation(s) for further study
− Baayen (2008: Section 4.1.2), Sheskin (2011: Test 9b, 6)

3.2. Tests for differences/independence

A particularly frequent scenario requires you to test two groups of elements with regard to whether they differ in their central tendency. As discussed above, there are several factors that determine which test to choose:

- the kind of samples: dependent or independent (cf. Section 1.3.4.1 and the beginning of Chapter 4);
- the level of measurement of the dependent variable: interval/ratio-scaled vs. ordinal;
- the distribution of (interval/ratio-scaled) dependent variable: normal vs. non-normal;
- the sample sizes.

To reiterate the discussion at the beginning of this chapter: is the dependent variable ratio-scaled as well as normally-distributed or both sample sizes are larger than 30 or are the differences between variables normally distributed, then you can usually do a *t*-test (for independent or dependent samples, as required) – otherwise you should do a *U*-test (for independent samples) or a Wilcoxon test (for dependent samples) (or, maybe, computationally intense exact tests). The reason for this decision procedure is that while the *t*-test for independent samples requires, among other things, normally distributed samples, we have seen that samples of 30+ elements can be normally distributed even if the underlying distribution is not. Therefore, it is sometimes sufficient, though not conservative, if the data meet one of the two conditions. Strictly speaking, the *t*-test for independent samples also requires homogenous variances, which we will also test for, but we will discuss a version of the *t*-test that can handle heterogeneous variances, the *t*-test after Welch.

3.2.1. One dep. variable (ratio-scaled) and one indep. variable (nominal) (indep. samples)

The *t*-test for independent samples is one of the most widely used tests. To

explore it, we use an example from the domain of phonetics. Let us assume you wanted to study the (rather trivial) non-directional H_1 that the first formants' frequencies of men and women differed. You plan an experiment in which you record men's and women's pronunciation of a relevant set of words and/or syllables, which you then analyze. This study involves

- one dependent ratio-scaled variable, namely F1-FREQUENCIES, whose averages you are interested in;
- one independent nominal variable, namely SEX: *MALE* vs. SEX: *FEMALE*;
- independent samples since, if every subject provides just one data point, the data points are not related to each other.

The test to be used for such scenarios is the *t*-test for independent samples and it involves the following steps:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test:
  - the population from which the samples whose means are tested have been drawn or at least the samples itself from which the means are computed are normally distributed (esp. with samples of $n < 30$)
  - the variances of the populations from which the samples have been drawn or at least the variances of the samples are homogeneous
  - the samples are independent of each other
- Computing the test statistic *t*, *df*, and *p*

You begin with the hypotheses.

H_0: The average F1 frequency of men is the same as the average F1 frequency of women: $mean_{\text{F1 frequency of men}} = mean_{\text{F1 frequency of women}}$, or $mean_{\text{F1 frequency of men}} - mean_{\text{F1 frequency of men}} = 0$, or $t = 0$;

H_1: The average F1 frequency of men is not the same as the average F1 frequency of women: $mean_{\text{F1 frequency of men}} \neq mean_{\text{F1 frequency of women}}$, or $mean_{\text{F1 frequency of men}} - mean_{\text{F1 frequency of men}} \neq 0$, or $t \neq 0$.

The data you will investigate here are part of the data borrowed from a similar experiment on vowels in Apache. First, you load the data from <_inputfiles/04-3-2-1_f1-freq.csv> into R:

```
> Vowels<-read.delim(file.choose()¶
> str(Vowels); attach(Vowels)¶
```

Then, you compute the relevant means and the standard deviations of the frequencies. As usual, we use the more elegant variant with `tapply`.

```
> tapply(HZ_F1, SEX, mean)¶
       F        M
528.8548  484.2740
> tapply(HZ_F1, SEX, sd)¶
        F         M
110.80099  87.90112
```

To get a better impression of the data, you also immediately generate a boxplot. You set the limits of the *y*-axis such that it ranges from 0 to 1,000 so that all values are nicely represented; in addition, you use `rug` to plot the values of the women and the men onto the left and right *y*-axis respectively; cf. Figure 51 and the code file for an alternative that includes a stripchart.

```
> boxplot(HZ_F1~SEX, notch=TRUE, ylim=(c(0, 1000)),
   xlab="Sex", ylab="F1 frequency"); grid()¶
> rug(HZ_F1[SEX=="F"], side=2); rug(HZ_F1[SEX=="M"], side=4)¶
```

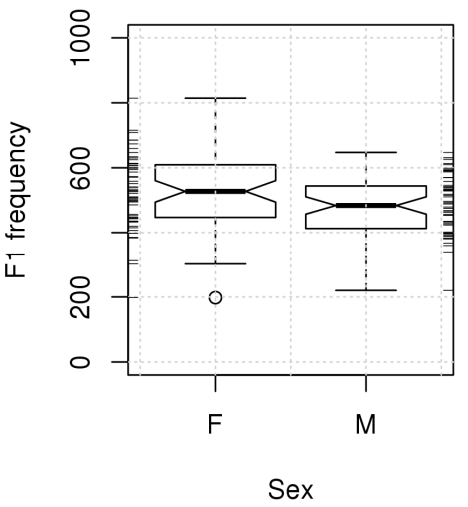

Figure 51. Boxplot for HZ_F1~SEX

The next step consists of testing the assumptions of the *t*-test. Figure 51 suggests that these data meet the assumptions. First, the boxplots for the

men and the women appear as if the data are normally distributed: the medians are in the middle of the boxes and the whiskers extend nearly equally long in both directions. Second, the variances seem to be very similar since the sizes of the boxes and notches are very similar. However, of course you need to test this and you use the familiar Shapiro-Wilk test:

```
> tapply(HZ_F1, SEX, shapiro.test)¶
$F
 Shapiro-Wilk normality test
data:  X[[1L]]
W = 0.987, p-value = 0.7723
$M
 Shapiro-Wilk normality test
data:  X[[2L]]
W = 0.9724, p-value = 0.1907
```

The data do not differ significantly from normality. Now you test for variance homogeneity with the *F*-test from Section 4.2.2 (whose assumption of normality we now already tested). This test's hypotheses are:

H_0: The variance of the first sample equals that of the second; $F = 1$.
H_1: The variance of one sample is larger than that of the second; $F \neq 1$.

The *F*-test with R yields the following result:

```
> var.test(HZ_F1~SEX) # with a formula¶
 F test to compare two variances
data:  HZ_F1 by SEX
F = 1.5889, num df = 59, denom df = 59, p-value = 0.07789
alternative hypothesis: true ratio of variances is not
   equal to 1
95 percent confidence interval:
 0.949093 2.660040
sample estimates:
ratio of variances
          1.588907
```

The second assumption is also met: since the confidence interval includes 1 and $p > 0.05$ so the variances are not significantly different from each other and you can compute the *t*-test for independent samples. This test involves three different statistics: the test statistic *t*, the number of degrees of freedom *df*, and of course the *p*-value. In the case of the *t*-test we discuss here, the *t*-test after Welch, the *t*-value is computed according to the formula in (46), where sd^2 is the variance, *n* is the sample size, and the subscripts 1 and 2 refer to the two samples of men and women.

$$(46) \qquad t = \left| \left(\bar{x}_1 - \bar{x}_2 \right) \div \sqrt{sd_1^{\,2} / n_1 + sd_2^{\,2} / n_2} \right|$$

```
> t.numerator<-mean(HZ_F1[SEX=="M"])-mean(HZ_F1[SEX=="F"])¶
> t.denominator<-sqrt((var(HZ_F1[SEX=="M"])/
   length((HZ_F1[SEX=="M"])))+(var(HZ_F1[SEX=="F"])/
   length((HZ_F1[SEX=="F"]))))¶
> t.value<-abs(t.numerator/t.denominator)¶
```

You get $t = 2.441581$. The formula for the degrees of freedom is somewhat more complex. First, you need to compute a value called c, and with c, you can then compute df. The formula to compute c is shown in (47), and the result of (47) gets inserted into (48).

$$(47) \qquad c = \frac{sd_1^{\,2} / n_1}{sd_1^{\,2} / n_1 + sd_2^{\,2} / n_2}$$

$$(48) \qquad df = \left(\frac{c^2}{n_1 - 1} + \frac{(1-c)^2}{n_2 - 1} \right)^{-1}$$

```
> c.numerator<-var(HZ_F1[SEX=="M"])/length(HZ_F1[SEX=="M"])¶
> c.denominator<-t.denominator^2¶
> c.value<-c.numerator/c.denominator¶
> df.summand1<-c.value^2/(length(HZ_F1[SEX=="M"])-1)¶
> df.summand2<-((1-c.value)^2)/(length(HZ_F1[SEX=="F"])-1)¶
> df<-(df.summand1+df.summand2)^-1¶
```

You get $c = 0.3862634$ and $df \approx 112.195$. You then look up the t-value in the usual kind of t-table (cf. Table 34) or you compute the critical t-value (with `qt(c(0.025, 0.975), 112, lower.tail=FALSE)¶`; as before, for a two-tailed test you compute the t-value for $p = 0.025$).

Table 34. Critical t-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $111 \leq df \leq 113$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 111$	1.9816	2.6208	3.3803
$df = 112$	1.9814	2.6204	3.3795
$df = 113$	1.9812	2.62	3.3787

As you can see, the observed *t*-value is larger than the one tabulated for $p = 0.05$, but smaller than the one tabulated for $p = 0.01$: the difference between the means is significant. The exact *p*-value can be computed with `pt` and for the present two-tailed case you simply enter this:

```
> 2*pt(t.value, 112.195, lower.tail=FALSE)¶
[1] 0.01618534
```

In R, you can use the function `t.test`, which takes several arguments, the first two of which – the relevant samples – can be given by means of a formula or with two vectors. These are the other relevant arguments:

- `alternative`: a character string that specifies which H_1 is tested: the default value, which can therefore be omitted, is `"two.sided"`, other values for one-tailed hypotheses are again `"less"` or `"greater"`; as before, R considers the alphabetically first variable level (i.e., here "F") as the reference category so that the one-tailed hypothesis that the values of the men are smaller than those of the women would be tested with `alternative="greater"`;
- `paired=FALSE` for the *t*-test for independent samples (the default) or `paired=TRUE` for the *t*-test for dependent samples (cf. the next section);
- `var.equal=TRUE`, when the variances of the two samples are equal, or `var.equal=FALSE` if they are not; the latter is the useful default, which should hardly be changed;
- `conf.level`: a value between 0 and 1, which specifies the confidence interval of the difference between the means; the default is 0.95.

Thus, to do the *t*-test for independent samples, you can enter either variant listed below. You get the following result:

```
> t.test(HZ_F1~SEX, paired=FALSE)¶
 Welch Two Sample t-test
data:  HZ_F1 by SEX
t = 2.4416, df = 112.195, p-value = 0.01619
alternative hypothesis: true difference in means is
   not equal to 0
95 percent confidence interval:
 8.403651 80.758016
sample estimates:
mean in group F mean in group M
       528.8548        484.2740
> t.test(HZ_F1[SEX=="F"], HZ_F1[SEX=="M"], paired=FALSE)¶
```

The first two lines of the output provide the name of the test and the data to which the test was applied. Line 3 lists the test statistic *t* (the sign is irrelevant and depends on which mean is subtracted from which, but it must of course be considered for the manual computation), the *df*-value, and the *p*-value. Line 4 states the H_1 tested. Then, you get the confidence interval for the differences between means (and our test is significant because this confidence interval does not include 0). Finally, you get the means again.

You can sum up your results as follows: "In the experiment, the average F1 frequency of the vowels produced by men was 484.3 Hz (*sd* = 87.9), the average F1 frequency of the vowels produced by the women was 528.9 Hz (*sd* = 110.8). According to a *t*-test for independent samples, the difference of 44.6 Hz between the means is statistically significant, but not particularly strong: $t_{Welch} = 2.4416$; $df = 112.2$; $p_{two\text{-}tailed} = 0.0162$."

In Section 5.2.2, we will discuss the extension of this test to cases where you have more than one independent variable and/or where the independent variable has more than two levels.

Recommendation(s) for further study

- Crawley (2007: 289ff.), Baayen (2008: Section 4.2.2), Sheskin (2011: Test 11)
- see the code file for a function `exact.t.test.indep` that I wrote to compute an exact version of this *F*-test, which you can use when your sample sizes are very small (maybe <15); careful, this test may take quite some time (and it requires the library `combinat`)

3.2.2. One dep. variable (ratio-scaled) and one indep. variable (nominal) (dep. samples)

The previous section illustrated a test for means from two independent samples. The name of that test suggests that there is a similar test for dependent samples, which we will discuss in this section on the basis of an example from translation studies. Let us assume you want to compare the lengths of English and Portuguese texts and their respective translations into Portuguese and English. Let us also assume you suspect that the translations are on average longer than the originals. This question involves

- one dependent ratio-scaled variable, namely the LENGTH of the texts;
- one independent nominal/categorical variable, namely TEXTSOURCE: *ORIGINAL* vs. TEXTSOURCE: *TRANSLATION*;

- dependent samples since the LENGTH values for each translation are connected to those of each original text.

Performing a *t*-test for dependent samples requires the following steps:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test: the differences of the paired values of the dependent samples are normally distributed
- Computing the test statistic *t*, *df*, and *p*

As usual, you formulate the hypotheses, but note that this time the H_1 is directional: you suspect that the average length of the originals is *shorter* than those of their translations, not just different (i.e., shorter *or* longer). Therefore, the statistical form of H_1 does not just contain a "≠", but something more specific, "<":

H_0: The average of the pairwise differences between the lengths of the originals and the lengths of the translations is 0; $mean_{\text{pairwise differerences}} = 0$; $t = 0$.

H_1: The average of the pairwise differences between the lengths of the originals and the lengths of the translations is smaller than 0; $mean_{\text{pairwise differerences}} < 0$; $t < 0$.

Note in particular (i) that the hypotheses do not involve the values of the two samples but the pairwise differences between them and (ii) how these differences are computed: original minus translation, not the other way round (and hence we use "< 0"). To illustrate this test, we will look at data from Frankenberg-Garcia (2004). She compared the lengths of eight English and eight Portuguese texts, which were chosen and edited such that their lengths were approximately 1,500 words, and then she determined the lengths of their translations. You can load the data from <_inputfiles/04-3-2-2_textlengths.csv>:

```
> Texts<-read.delim(file.choose()¶
> str(Texts); attach(Texts)¶
```

Note that the data are organized so that the order of the texts and their translations is identical: case 1 is an English original (hence, TEXT is 1,

TEXTSOURCE is *ORIGINAL*, LANGUAGE is *ENGLISH*), and case 17 is its translation (hence, TEXT is again 1, but TEXTSOURCE is now *TRANSLATION*, and LANGUAGE is *PORTUGUESE*), etc. First, you compute the means and generate a plot.

```
> tapply(LENGTH, TEXTSOURCE, mean)¶
    Original  Translation
    1500.062     1579.938
> boxplot(LENGTH~TEXTSOURCE, notch=TRUE, ylim=c(0, 2000))¶
> rug(LENGTH, side=2)¶
```

The median translation length is a little higher than that of the originals and the two samples have *very* different dispersions (only because the lengths of the originals were 'set' to approximately 1,500 words and thus exhibit very little variation while the lengths of the translations were not controlled like that).

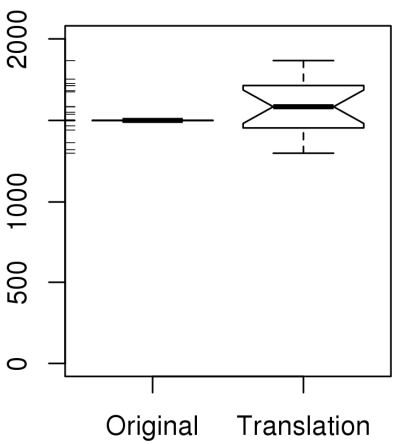

Figure 52. Boxplot for LENGTH~TEXTSOURCE

Now, this is actually a bad plot to represent the data – why?

THINK
BREAK

This plot does not portray the information that the data points from the left part – the lengths of the originals – are related to those from the right

part – the lengths of their translations! Thus, see the code file for three better plots (esp. the third). Given the controlled original lengths, the difference here is not that huge, but in other applications, a boxplot for dependent samples like the above can be very misleading.

Unlike the *t*-test for independent samples, the *t*-test for dependent samples does not presuppose a normal distribution or variance homogeneity of the sample values, but a normal distribution of the differences between the pairs of sample values. You can create a vector with these differences and then apply the Shapiro-Wilk test to it in one line with this shortcut.

```
> shapiro.test(differences<-LENGTH[1:16]-LENGTH[17:32])¶
 Shapiro-Wilk normality test
data:  differences
W = 0.9569, p-value = 0.6057
```

The differences do not differ significantly from normality so you can in fact do the *t*-test for dependent samples. First, you compute the *t*-value according to the formula in (49), where *n* is the number of value pairs.

$$t = \left|\bar{x}_{differences}\right| \cdot \sqrt{n} \Big/ sd_{differences} \tag{49}$$

```
> t.value<-(abs(mean(differences))*
   sqrt(length(differences)))/sd(differences)¶
> t,value
[1] 1.927869
```

Second, you compute the degrees of freedom *df*, which is the number of differences *n* minus 1:

```
> df<-length(differences)-1; df¶
[1] 15
```

First, you can now compute the critical values for $p = 0.05$ – this time *not* for $^{0.05}/_{2} = 0.025$ because you have a directional H_1 – at $df = 15$ or, in a more sophisticated way, create the whole *t*-table.

```
> qt(c(0.05, 0.95), 15, lower.tail=FALSE)¶
[1]  1.753050 -1.753050
```

Second, you can look up the *t*-value in such a *t*-table, repeated here as Table 35. Since such tables usually only list the positive values, you use the

absolute value of your *t*-value. As you can see, the differences between the originals and their translations is significant, but not very or highly significant: 1.927869 > 1.7531, but 1.927869 < 2.6025.

Table 35. Critical *t*-values for $p_{\text{one-tailed}} = 0.05$, 0.01, and 0.001 (for $14 \leq df \leq 16$)

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 14$	1.7613	2.6245	3.7874
$df = 15$	1.7531	2.6025	3.7328
$df = 16$	1.7459	2.5835	3.6862

Alternatively, you can compute the exact *p*-value. Since you have a directional H_1, you only need to cut off 5% of the area under the curve on one side of the distribution. The *t*-value following from H_0 is 0 and the *t*-value you computed is approximately 1.93 so you must compute the area under the curve from 1.93 to $+\infty$; cf. Figure 53. Since you are doing a one-tailed test, you need not multiply the *p*-value with 2.

```
> pt(t.value, 15, lower.tail=FALSE)¶
[1] 0.03651146
```

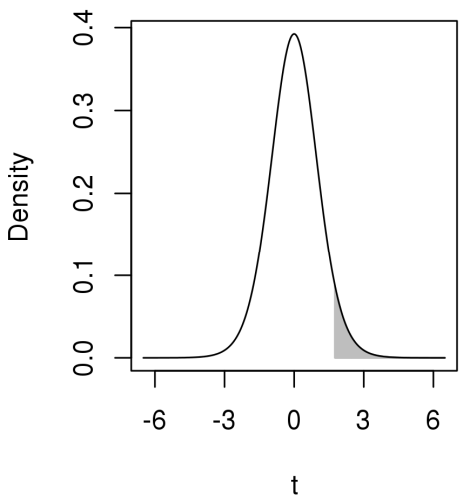

Figure 53. Density function for a *t*-distribution for $df = 15$, one-tailed test

Note that this also means that the difference is only significant because you did a one-tailed test –a two-tailed test with its multiplication with 2 would not have yielded a significant result but $p = 0.07302292$.

Now the same test with R. Since you already know the arguments of the function `t.test`, we can focus on the only major differences to before, the facts that you now have a directional H_1 and need to do a one-tailed test and that you now do a paired test. To do that properly, you must first understand how R computes the difference. As mentioned above, R proceeds alphabetically and computes the difference 'alphabetically first level minus alphabetically second level' (which is why H_1 was formulated this way above). Since "Original" comes before "Translation" and we hypothesized that the mean of the former would be smaller than that of the latter, the difference is smaller than 0. You therefore tell R that the difference is "less" than zero.

Of course you can use the formula or the vector-based notation. I show the output of the formula notation but both ways result in the same output. You get the *t*-value (ours was positive only because we used `abs`), the *df*-value, a *p*-value, and a confidence interval which, since it does not include 0, also reflects the significant result.

```
> t.test(LENGTH~TEXTSOURCE, paired=TRUE, alternative="less")¶
 Paired t-test
data:  LENGTH by TEXTSOURCE
t = -1.9279, df = 15, p-value = 0.03651
alternative hypothesis: true difference in means is less
   than 0
95 percent confidence interval:
      -Inf -7.243041
sample estimates:
mean of the differences
                -79.875
> t.test(LENGTH[TEXTSOURCE=="Original"], LENGTH[TEXTSOURCE==
   "Translation"], paired=TRUE, alternative="less")¶
```

To sum up: "On average, the originals are approximately 80 words shorter than their translations (the 95% confidence interval of this difference is -Inf, -7.24). According to a one-tailed *t*-test for dependent samples, this difference is significant: $t = -1.93$; $df = 15$; $p_{\text{one-tailed}} = 0.0365$. However, the effect is relatively small: the difference of 80 words corresponds to only about 5% of the length of the texts."

Recommendation(s) for further study

- Crawley (2007: 298ff.), Baayen (2008: Section 4.3.1), Sheskin (2011: Test 17)
- see the code file for a function `exact.t.test.dep` that I wrote to compute an exact version of this *F*-test, which you can use when your sam-

ple sizes are very small (maybe <15); careful, this test may take quite some time (for this example, it returns nearly the exact same *p*-value)

3.2.3. One dep. variable (ordinal) and one indep. variable (nominal) (indep. samples)

In this section, we discuss a non-parametric test for two independent samples of ordinal data, the *U*-test. Since I mentioned at the beginning of Section 4.3.2 that the *U*-test is not only used when the samples to be compared consist of ordinal data, but also when they violate distributional assumptions, this section will again involve an example where only a test of these distributional assumptions allows you to decide which test to use.

In Section 4.3.1.2 above, you looked at the similarities of source words entering into subtractive word formations and you tested whether these similarities were on average different from the known average similarity of random words to each other. The data you used were of the kind studied in Gries (2006) but in the above example no distinction was made between source words entering into different kinds of subtractive word formations. This is what we will do here by comparing similarities of source words entering into blends to similarities of source words entering into complex clippings. If both kinds of word-formation processes differed according to this parameter, this would provide empirical motivation for distinguishing them in the first place. This example, thus, involves

- one dependent ratio-scaled variable, namely the SIMILARITY of the source words whose averages you are interested in;
- one independent nominal variable, namely PROCESS: *BLEND* vs. PROCESS: *COMPLCLIP*;
- independent samples since the Dice coefficient of any one pair of source words has nothing to do with any one other pair of source words.

This kind of question would typically be investigated with the *t*-test for independent samples we discussed above. According to the above procedure, you first formulate the hypotheses (non-directionally, since we may have no a priori reason to assume a particular difference):

H_0: The mean of the Dice coefficients of the source words of blends is the same as the mean of the Dice coefficients of the source words of complex clippings; $mean_{\text{Dice coefficients of blends}} = mean_{\text{Dice coefficients of}}$

complex clippings, or $mean_{\text{Dice coefficients of blends}} - mean_{\text{Dice coefficients of complex clippings}} = 0$.

H_1: The mean of the Dice coefficients of the source words of blends is not the same as the mean of the Dice coefficients of the source words of complex clippings; $mean_{\text{Dice coefficients of blends}} \neq mean_{\text{Dice coefficients of complex clippings}}$, or $mean_{\text{Dice coefficients of blends}} - mean_{\text{Dice coefficients of complex clippings}} \neq 0$.

You can load the data from the file <_inputfiles/04-3-2-3_dices.csv>. As before, this file contains the Dice coefficients, but now also in an additional column the word formation process for each Dice coefficient.

```
> Dices<-read.delim(file.choose())¶
> str(Dices); attach(Dices)¶
```

As usual, you should begin by exploring the data graphically:

```
> boxplot(DICE~PROCESS, notch=TRUE, ylim=c(0, 1),
   ylab="Dice")¶
> rug(jitter(DICE[PROCESS=="Blend"]), side=2)¶
> rug(jitter(DICE[PROCESS=="ComplClip"]), side=4)¶
> text(1:2, tapply(DICE, PROCESS, mean), "x")¶
```

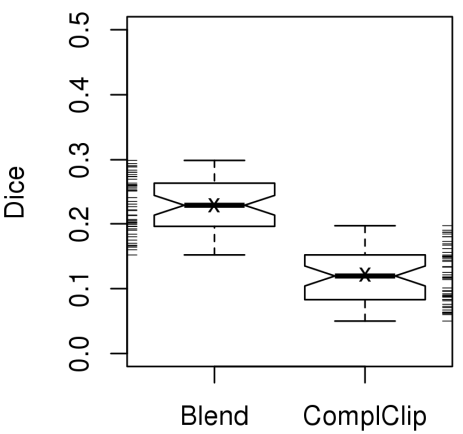

Figure 54. Boxplot for SIMILARITY~PROCESS

As usual, this graph already gives away enough information to nearly obviate the need for statistical analysis. The probably most obvious aspect is the difference between the two medians, but since the data are ratio-

scaled you also need to explore the means. These are already plotted into the graph and here is the usual line of code to compute them; note how large the difference is between the two.

```
> tapply(DICE, PROCESS, mean)¶
    Blend ComplClip
  0.22996   0.12152
> tapply(DICE, PROCESS, sd)¶
      Blend  ComplClip
  0.4274985 0.04296569
```

In order to test whether the *t*-test for independent samples can be used here, we need to test both of its assumptions, normality in the groups and variance homogeneity. Since the *F*-test for homogeneity of variances presupposes normality, you begin by testing whether the data are normally distributed. The rugs in Figure 54 suggest they are not, which is supported by the Shapiro-Wilk test.

```
> tapply(DICE, PROCESS, shapiro.test)¶
$Blend
 Shapiro-Wilk normality test
data:  X[[1L]]
W = 0.9455, p-value = 0.02231
$ComplClip
 Shapiro-Wilk normality test
data:  X[[2L]]
W = 0.943, p-value = 0.01771
```

Given these violations of normality, you can actually not do the regular *F*-test to test the second assumption of the *t*-test for independent samples. You therefore do the Fligner-Killeen test of homogeneity of variances, which does not require the data to be normally distributed and which I mentioned in Section 4.2.2 above.

```
> fligner.test(DICE~PROCESS)¶
 Fligner-Killeen test of homogeneity of variances
data:  DICE by PROCESS
Fligner-Killeen:med chi-squared=3e-04, df=1, p-value=0.9863
```

The variances are homogeneous, but normality is still violated. It follows that even though the data are ratio-scaled and even though the sample sizes are larger than 30, it may safer to compute a test that does not make these assumptions, the *U*-test.

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test:
  - the samples are independent of each other
  - the populations from which the samples whose central tendencies are tested have been drawn are identically distributed [26]
- Computing the test statistic U, z, and p

The two boxplots look relatively similar and the variances of the two groups are not significantly different, and the U-test is robust (see above) so we use it here. Since the U-test assumes only ordinal data, you now compute medians, not just means. You therefore adjust your hypotheses and compute medians and interquartile ranges:

H_0: The median of the Dice coefficients of the source words of blends is as large as the median of the Dice coefficients of the source words of complex clippings; $median_{\text{Dice coefficients of blends}} = median_{\text{Dice coefficients of complex clippings}}$, or $median_{\text{Dice coefficients of blends}} - median_{\text{Dice coefficients of complex clippings}} = 0$.

H_1: The median of the Dice coefficients of the source words of blends is not as large as the median of the Dice coefficients of the source words of complex clippings; $median_{\text{Dice coefficients of blends}} \neq median_{\text{Dice coefficients of complex clippings}}$, or $median_{\text{Dice coefficients of blends}} - median_{\text{Dice coefficients of complex clippings}} \neq 0$.

```
> tapply(DICE, PROCESS, median)¶
    Blend ComplClip
    0.2300     0.1195
> tapply(DICE, PROCESS, IQR)¶
    Blend ComplClip
   0.0675    0.0675
```

Here, the assumptions can be tested fairly unproblematically: The values are independent of each other since no word-formation influences another one, the distributions of the data in Figure 54 appear to be rather similar, and a Kolmogorov-Smirnov test of the z-standardized Dice values for both word-formation processes is completely insignificant ($p = 0.9972$).

Unfortunately, computing the U-test is more cumbersome than many

26. According to Bortz, Lienert, and Boehnke (1990:211), the U-test can discover differences of measures of central tendency well even if this assumption is violated.

other tests. First, you transform all Dice coefficients into ranks, and then you compute the sum of all ranks for each word-formation process. Then, both of these *T*-values and the two sample sizes are inserted into the formulae in (50) and (51) to compute two *U*-values, the smaller one of which is the required test statistic.

```
> Ts<-tapply(rank(DICE), PROCESS, sum)¶
```

$$(50) \qquad U_1 = n_1 \cdot n_2 + \frac{n_1 \cdot (n_1 + 1)}{2} - T_1$$

$$(51) \qquad U_2 = n_1 \cdot n_2 + \frac{n_2 \cdot (n_2 + 1)}{2} - T_2$$

```
> n1<-length(DICE[PROCESS=="Blend"])¶
> n2<-length(DICE[PROCESS=="ComplClip"])¶
> U1<-n1*n2+((n1*(n1+1))/2)-Ts[1]¶
> U2<-n1*n2+((n2*(n2+1))/2)-Ts[2]¶
> U.value<-min(U1, U2)¶
```

The *U*-value, 84, can be looked up in a *U*-table or, because there are few *U*-tables for large samples,[27] converted into a normally-distributed *z*-score. This *z*-score is computed as follows. First, you use the formulae in (52) and (53) to compute an expected *U*-value and its dispersion.

$$(52) \qquad U_{\text{expected}} = 0.5 \cdot n_1 \cdot n_2$$

$$(53) \qquad \textit{Dispersion } U_{\text{expected}} = \sqrt{\frac{n_1 \cdot n_2 \cdot (n_1 + n_2 + 1)}{12}}$$

Second, you insert these values together with the observed *U* into (54).

$$(54) \qquad z = \frac{U - U_{expected}}{\textit{Dispersion } U_{expected}}$$

```
> expU<-n1*n2/2¶
> dispersion.expU<-sqrt(n1*n2*(n1+n2+1)/12)¶
> z<-abs((U.value-expU)/dispersion.expU)¶
```

27. Bortz, Lienert and Boehnke (1990:202 and Table 6) provide critical *U*-values for $n \leq 20$ and mention references for tables with critical values for $n \leq 40$ – I at least know of no *U*-tables for larger samples.

To decide whether H_0 can be rejected, you look up this value, 8.038194, in a *z*-table such as Table 36 or you compute a critical *z*-score for $p_{\text{two-tailed}} = 0.05$ with `qnorm` (as mentioned in Section 1.3.4.2 above). Since you have a non-directional H_1, you apply the same logic as above and compute *z*-scores for half of the $p_{\text{two-tailed}}$-values you are interested in:

Table 36. Critical *z*-scores for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001

z-score	*p*-value
1.96	0.05
2.575	0.01
3.291	0.001

```
> qnorm(c(0.9995, 0.995, 0.975, 0.025, 0.005, 0.0005),
   lower.tail=FALSE)¶
[1] -3.290527 -2.575829 -1.959964  1.959964  2.575829
   3.290527
```

It is obvious that the observed *z*-score is not only much larger than the one tabulated for $p_{\text{two-tailed}} = 0.001$ but also very distinctly in the grey-shaded area in Figure 55: the difference between the medians is highly significant, as the non-overlapping notches already anticipated. Plus, you can compute the exact *p*-value with the usual 'mirror function' of `qnorm`.

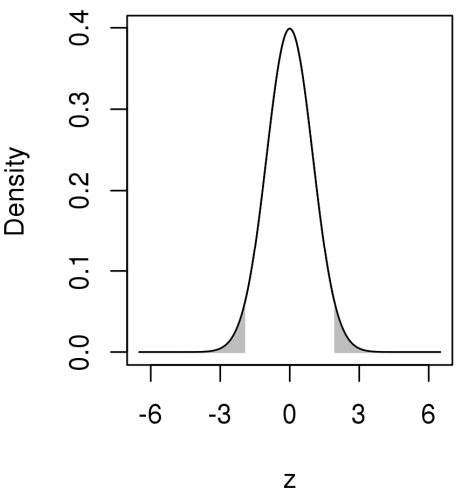

Figure 55. Density function of the standard normal distribution; two-tailed test

```
> 2*pnorm(z, lower.tail=FALSE)¶
[1] 9.117223e-16
```

In R, you compute the *U*-test with the same function as the Wilcoxon test, `wilcox.test`, and again you can either use a formula or two vectors. Apart from these arguments, the following ones are useful, too:

- `alternative`: a character string specifying which H_1 you want to test: the default is `"two.sided"`, other possible values for one-tailed tests are again `"less"` or `"greater"`, which specify how the first-named vector or factor level relates to the second;
- `paired=FALSE` for the *U*-Test for independent samples or `paired=TRUE` for the Wilcoxon test for dependent samples (cf. the following section);
- `exact=TRUE`, if you want to compute an exact test, or `exact=FALSE` if you don't (if you don't change `exact`'s default setting of `NULL` and your data set has fewer than 50 data points and no ties, an exact *p*-value is computed automatically);
- `correct=TRUE` for a continuity correction (the default) and `correct=FALSE` for none;
- `conf.level`: a value between 0 and 1 specifying the size of the confidence interval; the default is 0.95.

The standard version to be used here is this:

```
> wilcox.test(DICE~PROCESS, paired=FALSE, correct=FALSE)¶
 Wilcoxon rank sum test
data:  DICE by PROCESS
W = 2416, p-value = 9.072e-16
alternative hypothesis: true location shift is not equal to 0
```

You get a *U*-value (here referred to as *W*) and a *p*-value; *W* is not the minimum of U_1 and U_2, but the maximum here, which value you get depends on which vector or factor level comes first in the alphabet. The *p*-value here is a bit different from yours since R uses a slightly different algorithm. You can now sum up: "According to a *U*-test, the median Dice coefficient of the source words of blends (0.23, $IQR = 0.0675$) and the median of the Dice coefficients for complex clippings (0.12, $IQR = 0.0675$) are very significantly different: $U = 84$ (or $W = 2416$), $p_{\text{two-tailed}} < 0.0001$. The creators of blends appear to be more concerned with selecting source words that are similar to each other than the creators of complex clippings."

Recommendation(s) for further study:
Dalgaard (2002: 89f.), Crawley (2007: 297f.), Baayen (2008: Section 4.3.1), Sheskin (2011: Test 12)

3.2.4. One dep. variable (ordinal) and one indep. variable (nominal) (dep. samples)

Just like the *U*-test, the test in this section has two major applications. First, you really may have two dependent samples of ordinal data such as when you have a group of subjects perform two rating tasks to test whether each subject's first rating differs from the second. Second, the probably more frequent application arises when you have two dependent samples of ratio-scaled data but cannot do the *t*-test for dependent samples because its distributional assumptions are not met. We will discuss an example of the latter kind in this section.

In a replication of Bencini and Goldberg (2000), Gries and Wulff (2005) studied the question which verbs or sentence structures are more relevant for how German foreign language learners of English categorize sentences. They crossed four syntactic constructions and four verbs to get 16 sentences, each verb in each construction. Each sentence was printed onto a card and 20 advanced German learners of English were given the cards and asked to sort them into four piles of four cards each. The question was whether the subjects' sortings would be based on the verbs or the constructions. To determine the sorting preferences, each subject's four stacks were inspected with regard to how many cards one would minimally have to move to create either four completely verb-based or four completely construction-based sortings. The investigation of this question involves

- one dependent ratio-scaled variable, namely SHIFTS, the number of times a card had to be shifted from one stack to another to create the perfectly clean sortings, and we are interested in the average of these numbers;
- one independent nominal variable, namely CRITERION: *CONSTRUCTION* vs. CRITERION: *VERB*;
- dependent samples since each subject 'generated' two numbers of shifts, one to create the verb-based sorting, one to create the construction-based sorting.

To test some such result for significance, you should first consider a *t*-test for dependent samples since you have two samples of ratio-scaled values. As usual, you begin by formulating the relevant hypotheses:

H_0: The average of the pairwise differences between the numbers of rearrangements towards perfectly verb-based stacks and the numbers of rearrangements towards perfectly construction-based stacks is 0; $mean_{\text{pairwise differerences}} = 0$.

H_1: The average of the pairwise differences between the numbers of rearrangements towards perfectly verb-based stacks and the numbers of rearrangements towards perfectly construction-based stacks is not 0; $mean_{\text{pairwise differerences}} \neq 0$.

Then, you load the data that Gries and Wulff (2005) obtained in their experiment from <_inputfiles/04-3-2-4_sortingstyles.csv>:

```
> SortingStyles<-read.delim(file.choose())¶
> head(SortingStyles, 3); attach(SortingStyles)¶
```

As usual, you compute means and standard deviations and generate a graph of the results.

```
> tapply(SHIFTS, CRITERION, mean)¶
Construction          Verb
        3.45          8.85
> tapply(SHIFTS, CRITERION, sd)¶
Construction          Verb
    4.346505      4.107439
> differences<-SHIFTS[CRITERION=="Construction"]-
   SHIFTS[CRITERION!="Construction"]¶
> stripchart(differences, method="stack", xlim=c(-12, 12),
   xlab="Differences: ->construction minus ->verb");
   abline(v=0, lty=2, col="grey")¶
```

Note: since the two samples are dependent, we are plotting the differences, just as in Section 4.3.2.2 above. You then test the assumption of the *t*-test for dependent samples, the normality of the pairwise differences. Given Figure 56, those are obviously not normal:

```
> shapiro.test(differences)¶
 Shapiro-Wilk normality test
data:  differences
W = 0.7825, p-value = 0.0004797
```

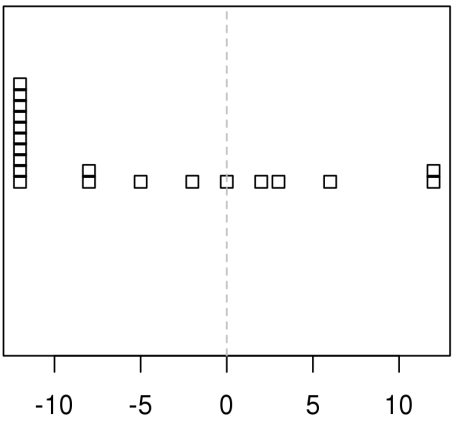

Figure 56. Strip chart of the differences of shifts

You cannot use the *t*-test. Instead, you compute a test for two dependent samples of ordinal variables, the Wilcoxon test.

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test:
  - the pairs of values are independent of each other
  - the populations from which the samples whose central tendencies are tested have been drawn are identically distributed
- Computing the test statistic *T* and *p*

As a first step, you adjust your hypotheses to the ordinal level of measurement, you then compute the medians and their interquartile ranges:

H_0: $median_{\text{pairwise differerences}} = 0$
H_1: $median_{\text{pairwise differerences}} \neq 0$

```
> tapply(SHIFTS, CRITERION, median)¶
Construction         Verb
           1           11
> tapply(SHIFTS, CRITERION, IQR)¶
Construction         Verb
        6.25         6.25
```

The assumptions appear to be met because the pairs of values are independent of each other (since the sorting of any one subject does not affect any other subject's sorting) and, somewhat informally, there is little reason to assume that the populations are distributed differently especially since most of the values to achieve a perfect verb-based sorting are the exact reverse of the values to get a perfect construction-based sorting. Thus, you compute the Wilcoxon test; for reasons of space we only consider the standard variant. First, you transform the vector of pairwise differences, which you already computed for the Shapiro-Wilk test, into ranks:

```
> ranks<-rank(abs(differences))¶
```

Second, all ranks whose difference was negative are summed to a value *T*-, and all ranks whose difference was positive are summed to *T*+; the smaller of the two values is the required test statistic *T*:[28]

```
> T.minus<-sum(ranks[differences<0])¶
> T.plus<-sum(ranks[differences>0])¶
> T.value<-min(T.minus, T.plus)¶
```

This *T*-value of 41.5 can be looked up in a *T*-table (Table 37), but note that here, for a significant result, the observed test statistic must be *smaller* than the tabulated one.

Table 37. Critical *T*-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $14 \leq df \leq 16$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$n = 19$	46	32	18
$n = 20$	52	37	21
$n = 21$	58	42	25

The observed *T*-value of 41.5 is smaller than the one tabulated for $n = 20$ and $p = 0.05$ (but larger than the one tabulated for $n = 20$ and $p = 0.01$): the result is significant.

Let us now do this test with R: You already know the function for the Wilcoxon test so we need not discuss it again in detail. The relevant difference is that you now instruct R to treat the samples as dependent/paired. As nearly always, you can use the formula or the vector-based function call.

28. The way of computation discussed here is the one described in Bortz (2005). It disregards ties and cases where the differences are zero; cf. also Sheskin (2011:812).

```
> wilcox.test(SHIFTS~CRITERION, paired=TRUE, exact=FALSE,
   correct=FALSE)¶
 Wilcoxon signed rank test
data:  SHIFTS by CRITERION
V = 36.5, p-value = 0.01527
alternative hypothesis: true location shift is not equal to 0
```

R computes the test statistic differently but arrives at the same kind of decision: the result is significant, but not very significant. To sum up: "On the whole, the 20 subjects exhibited a strong preference for a construction-based sorting style: the median number of card rearrangements to arrive at a perfectly construction-based sorting was 1 while the median number of card rearrangements to arrive at a perfectly verb-based sorting was 11 (both *IQR*s = 6.25). According to a Wilcoxon test, this difference is significant: $V = 36.5$, $p_{\text{two-tailed}} = 0.0153$. In this experiment, the syntactic patterns were a more salient characteristic than the verbs (when it comes to what triggered the sorting preferences)."

Recommendation(s) for further study:
- Dalgaard (2002:92), Sheskin (2011: Test 18)

4. Coefficients of correlation and linear regression

In this section, we discuss the significance tests for the coefficients of correlation discussed in Section 3.2.3.

4.1. The significance of the product-moment correlation

While the manual computation of the product-moment correlation above was a bit complex, its significance test is not. It involves these steps:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test: the population from which the sample was drawn is bivariately normally distributed. Since this criterion *can* be hard to test (cf. Bortz 2005: 213f.), we simply require both samples to be distributed normally
- Computing the test statistic *t*, *df*, and *p*

Let us return to the example in Section 3.2.3, where you computed a correlation coefficient of 0.9337 for the correlation of the lengths of 20 words and their reaction times. You formulate the hypotheses and we assume for now your H_1 is non-directional.

H_0: The length of a word in letters does not correlate with the word's reaction time in a lexical decision task; $r = 0$.
H_1: The length of a word in letters correlates with the word's reaction time in a lexical decision task; $r \neq 0$.

You load the data from <_inputfiles/04-4_reactiontimes.csv>:

```
> ReactTime<-read.delim (file.choose())¶
> str(ReactTime); attach(ReactTime)¶
```

Since we already generated a scatterplot above (cf. Figure 35 and Figure 36), we will skip plotting for now. We do, however, have to test the assumption of normality of both vectors. You can either proceed in a stepwise fashion and enter `shapiro.test(LENGTH)¶` and `shapiro.test(MS_LEARNER)¶` or use a shorter variant:

```
> apply(ReactTime[,2:3], 2, shapiro.test)¶
$LENGTH
 Shapiro-Wilk normality test
data:  newX[, i]
W = 0.9748, p-value = 0.8502
$MS_LEARNER
 Shapiro-Wilk normality test
data:  newX[, i]
W = 0.9577, p-value = 0.4991
```

This line of code means 'take the data mentioned in the first argument of `apply` (the second and third column of the data frame `ReactTime`), look at them column by column (the 2 in the second argument slot – a 1 would look at them row-wise; recall this notation from `prop.table` in Section 3.2.1), and apply the function `shapiro.test` to each of these columns. Clearly, both variables do not differ significantly from a normality.

To compute the test statistic *t*, you insert the correlation coefficient *r* and the number of correlated value pairs *n* into the formula in (55):

(55) $$t = \left| \frac{r \cdot \sqrt{n-2}}{\sqrt{1-r^2}} \right|$$

```
> r<-cor(LENGTH, MS_LEARNER, method="pearson")¶
> numerator<-r*sqrt(length(LENGTH)-2)¶
> denominator<-sqrt(1-r^2)¶
> t.value<-abs(numerator/denominator)¶
```

This *t*-value, 11.06507, has $df = n\text{-}2 = 18$ degrees of freedom.

```
> df<-length(LENGTH)-2¶
```

Just as with the *t*-tests before, you can now look this *t*-value up in a *t*-table, or you can compute a critical value: if the observed *t*-value is higher than the tabulated/critical one, then *r* is significantly different from 0. Since your *t*-value is much larger than even the one for $p = 0.001$, the correlation is highly significant.

```
> qt(c(0.025, 0.975), 18, lower.tail=FALSE)¶
[1]  2.100922 -2.100922
```

Table 38. Critical *t*-values for $p_{\text{two-tailed}} = 0.05$, 0.01, and 0.001 for $17 \leq df \leq 19$

	$p = 0.05$	$p = 0.01$	$p = 0.001$
$df = 17$	2.1098	2.8982	3.9561
$df = 18$	2.1009	2.8784	3.9216
$df = 19$	2.093	2.8609	3.8834

The exact *p*-value can be computed as follows, and do not forget to again double the *p*-value.

```
> 2*pt(t.value, 18, lower.tail=FALSE)¶
[1] 1.841060e-09
```

This *p*-value is obviously much smaller than 0.001. However, you will already suspect that there is an easier way to get all this done. Instead of the function `cor`, which we used in Section 3.2.3 above, you simply use `cor.test` with the two vectors whose correlation you are interested in (and, if you have a directional H_1, you specify whether you expect the correlation to be less than 0 (i.e., negative) or greater than 0 (i.e., positive)

using `alternative=…`):

```
> cor.test(LENGTH, MS_LEARNER, method="pearson")¶
 Pearson's product-moment correlation
data:  LENGTH and MS_LEARNER
t = 11.0651, df = 18, p-value = 1.841e-09
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8370608 0.9738525
sample estimates:
      cor
0.9337171
```

Here are the (edited) results of the corresponding linear regression:

```
> model<-lm(MS_LEARNER~LENGTH)¶
> summary(model)¶
Call:
lm(formula = MS_LEARNER ~ LENGTH)

Residuals:
     Min       1Q   Median       3Q      Max
-22.1368  -7.8109   0.8413   7.9499  18.9501
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  93.6149     9.9169    9.44 2.15e-08 ***
LENGTH       10.3044     0.9313   11.06 1.84e-09 ***
---                                                […]
Multiple R-Squared: 0.8718,     Adjusted R-squared: 0.8647
F-statistic: 122.4 on 1 and 18 DF,  p-value: 1.841e-09
```

We begin at the bottom: the last row contains information we already know. The *F*-value is our *t*-value squared; we find the 18 degrees of freedom and the *p*-value we computed. In the line above that, you find the coefficient of determination you know plus an adjusted version we will only talk about later (cf. Section 5.2). We ignore the edited-out line about the residual standard error for now and the legend for the *p*-values. The table above that shows the intercept and the slope we computed in Section 3.2.3 (in the column labeled "Estimate"), their standard errors, *t*-values – do you recognize the *t*-value from above? – and *p*-values. The *p*-value for LENGTH says whether the slope of the regression line is significantly different from 0; the *p*-value for the intercept says whether the intercept of 93.6149 is significantly different from 0. We skip the info on the residuals because we discussed above how you can investigate those yourself (with `residuals(model)¶`).

There is one final but immensely useful thing to be discussed. Recall that above we used the function `predict` to get the predicted reaction times

for every observed word length, but also predicted reaction times for non-observed word lengths. The function `predict` can return more than this, however: it can also return confidence intervals for the predictions, which also allows to plot the regression line with its confidence interval. Since we will use this frequently in Chapter 5, we will go over one example here, which will involve three steps.

The first step repeats what we did above: we generate a data frame `preds.hyp` that contains a range of values covering the observed word lengths and that we will pass on to `predict`, and we do that as in Section 3.2.3 with `expand.grid()`. I call it `preds.hyp` to indicate that these are predictions from the model not for the actually observed lengths but for a range of hypothetical values. Note again that the column in `preds.hyp` has the same name as the independent variable in `model`.

```
> preds.hyp<-expand.grid(LENGTH=min(LENGTH):max(LENGTH))¶
```

The second step is also similar to Section 3.2.3 above, but with two small changes. We not only use `predict` to generate the predictions from `model` for this data frame, but (i) we also let R compute the confidence intervals for all predictions and (ii) we make the predictions and the confidence intervals columns 2 to 4 in `preds.hyp`:

```
> preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-predict(
   model, newdata=preds.hyp, interval="confidence")¶
```

If you look at the data frame `preds.hyp` now, you will see we now have a very nice result: the independent variable is in the column `preds.hyp$LENGTH`, the predicted dependent variable is in the column `preds.hyp$PREDICTIONS`, and the lower and upper confidence intervals for each prediction are in the columns `preds.hyp$LOWER` and `preds.hyp$UPPER` respectively.

The third step now involves generating a nice plot. The following code pulls many things together and introduces the function `matlines`:

```
> plot(MS_LEARNER~LENGTH, xlab="Word length in letters",
   ylab="Reaction time of learners in ms", pch=16,
   col=rgb(0, 0, 0, 70, maxColorValue=255)); grid()¶
> matlines(preds.hyp[,1], preds.hyp[,2:4], lwd=c(2, 1,
   1), lty=c(1, 2, 2), col=c("black", "blue", "blue"))¶
```

The first line just generates a regular scatterplot – the only new thing is the use of the function `rgb` to use a semi-transparent greyshade to avoid

information loss through overplotting. The second line uses `matlines`: the first argument is the first column of `preds.hyp` and provides the *x*-values for the lines to be plotted. The second argument is columns 2 to 4 of `preds.hyp` and provides three different sets of *y*-values to plot with separate lines: first the predicted values (= the regression line), second and third the lower and upper limits of the confidence intervals. The arguments `lwd` (line width), `lty` (line type), and `col` (color) describe what the lines should look like, in the order in which they appear in `preds.hyp`. The result you see when you run the code: a scatterplot with a regression line and its confidence band, and we can see again why the correlation is so high: not only is the regression line a good summary of the data, the confidence band is quite narrow around it and many points are right in it or very close to it.

This was a very detailed description, but since we will use this many times in Chapter 5, this is time well spent. To sum up: "The lengths of the words in letters and the reaction times in the experiment correlate highly positively with each other: $r = 0.9337$; adjusted $R^2 = 0.8647$. This correlation is highly significant: $t = 11.07$; $df = 18$; $p < 0.001$. The linear regression shows that every additional letter increases the reaction time by approximately 10.3 ms."

In Section 5.2, we deal with the extensions of linear regression to cases where we include more than one independent variable, and we will also discuss more comprehensive tests of the regression's assumptions (using `plot(model)¶`).

4.2. The significance of Kendall's Tau

If you need a *p*-value for Kendall's tau τ, you follow this procedure:

Procedure
- Formulating the hypotheses
- Computing descriptive statistics and visualizing the data
- Testing the assumption(s) of the test: the data from both samples are at least ordinal
- Computing the test statistic *z* and *p*

Again, we simply use the example from Section 3.2.3 above (even though we know we can actually use the product-moment correlation; we use this example again just for simplicity's sake). How to formulate the hypotheses should be obvious by now:

H_0: The length of a word in letters does not correlate with the word's reaction time in a lexical decision task; $\tau = 0$.

H_1: The length of a word in letters correlates with the word's reaction time in a lexical decision task; $\tau \neq 0$.

As for the assumption: we already know the data are ordinal – after all, we know they are even interval/ratio-scaled. You load the data again from <_inputfiles/03-2-3_reactiontimes.csv> and compute Kendall's τ:

```
> ReactTime<-read.delim (file.choose())¶
> str(ReactTime); attach(ReactTime)¶
> tau<-cor(LENGTH, MS_LEARNER, method="kendall")¶
```

To test Kendall's tau τ for significance, you compute a z-score of the kind that is by now familiar. You insert τ and the number of value pairs n into the formula in (56).

(56) $$z = |\tau| \div \sqrt{\frac{2 \cdot (2 \cdot n + 5)}{9 \cdot n \cdot (n - 1)}}$$

In R:

```
> numerator.root<-2*(2*length(LENGTH)+5)¶
> denominator.root<-9*length(LENGTH)*(length(LENGTH)-1)¶
> z.score<-abs(tau)/sqrt(numerator.root/denominator.root)¶
> z.score¶
[1] 5.048596
```

This value can be looked up in a z-table (cf. Table 36) or you generate these values yourself. The z-score for a significant two-tailed test must cut off at least 2.5% of the area under the standard normal distribution:

```
> qnorm(c(0.9995, 0.995, 0.975, 0.025, 0.005, 0.0005),
   lower.tail=FALSE)¶
[1] -3.290527 -2.575829 -1.959964  1.959964  2.575829
   3.290527
```

For a result to be significant, the z-score must be larger than 1.96. Since the observed z-score is even larger than 5, this result is highly significant:

```
> 2*pnorm(z.score, lower.tail=FALSE)¶
[1] 4.450685e-07
```

The function to get this result much faster is again `cor.test`. Since R uses a slightly different method of calculation, you get a slightly different *z*-score and *p*-value, but for all practical purposes the results are identical.

```
> cor.test(LENGTH, MS_LEARNER, method="kendall")¶
 Kendall's rank correlation tau
data:  LENGTH and MS_LEARNER
z = 4.8836, p-value = 1.042e-06
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.8189904
```

(The warning refers to ties such as that the length value 11 occurs more than once). To sum up: "The lengths of the words in letters and the reaction times in the experiment correlate highly positively with each other: $\tau = 0.819$, $z = 5.05$; $p < 0.001$."

4.3. Correlation and causality

Especially in the area of correlations, but also more generally, you need to bear in mind a few things even if H_0 is rejected: First, one can often hear a person A making a statement about a correlation (maybe even a significant one) by saying "The more X, the more Y" and then hear a person B objecting to that correlation on the grounds that B knows of an exception. This argument is flawed. The exception quoted by B would only invalidate A's statement if A considered the correlation to be perfect ($r = 1$ or $r = -1$) – but if A did not mean that (and A never does!), then there may be a strong and significant correlation although there is one exception (or more). The exception or exceptions are the reason why the correlation is not 1 or -1 but 'only', say, 0.9337. Second, a correlation as such does not necessarily imply causality. As is sometimes said, a correlation between X and Y is a *necessary* condition for a causal relation between X and Y, but not a *sufficient* one, as you can see from many examples:

– There is a positive correlation between the number of firefighters trying to extinguish a fire and the amount of damage that is caused at the site where the fire was fought. This does of course not mean that the firefighters arrive at the site and destroy as much as they can – the correlation results from a third, confounding variable, the size of the fire: the larger the fire, the more firefighters are called to help extinguish it *and*

the more damage the fire causes.
- There is a negative correlation between the amount of hair men have and their income which is unfortunately only due to the effect of a third variable: the men's age.
- There is a positive correlation such that the more likely a drug addict was to go to therapy to get off of his addiction, the more likely he was to die. This is not because the therapy leads to death – the confounding variable in the background correlated with both is the severity of the addiction: the more severely addicted addicts were, the more likely they were to go to therapy, but also the more likely they already were to die.

Thus, beware of jumping to conclusions …
Now you should do the exercise(s) for Chapter 4 …

Recommendation(s) for further study
- the functions `ckappa` and `lkappa` (from the library `psy`) to compute the kappa coefficient and test how well two or more raters conform in their judgments of stimuli
- the function `cronbach` (from the library `psy`) to compute Cronbach's alpha and test how consistently several variables measure a construct the variables are supposed to reflect
- Crawley (2007: Ch. 10), Baayen (2008: Section 4.3.2), Johnson (2008: Section 2.4), Sheskin (2011: Test 28, 30, 31, 32)
- the function `hints` (from the library `hints`) to get ideas about what to do next with a particular object

Chapter 5
Selected multifactorial and multivariate methods

All models are wrong, but some are useful.
George E.P. Box

So far we have only been concerned with monofactorial methods, i.e., methods in which we investigated how maximally one independent variable is correlated with the behavior of one dependent variable. In many cases, proceeding like this is the beginning of the empirical quantitative study of a phenomenon. Nevertheless, such a view on phenomena is usually a simplification: we live in a multifactorial world in which probably no phenomenon is really monofactorial – probably just about everything is correlated with several things at the same time. This is especially true for language, one of the most complex phenomena resulting from human evolution. In this section, we will therefore discuss several multifactorial techniques, which can handle this kind of complexity better than the monofactorial methods discussed so far. You should know, however, each section's method below could easily fill courses for several quarters or semesters, which is why I can unfortunately not discuss every aspect or technicality of the methods and why I will have to give you a lot of references and recommendations for further study. Also, given the complexity of these methods, there will be no discussion of how to compute them manually.

Before we can begin to discuss multifactorial methods, however, there is a lot to discuss. On a very abstract level, this discussion involves the notions of *interaction* and *model (selection)* and will be the subject of Section 5.1. However, as you will see soon, these notions will quickly lead to a variety of interrelated concepts and, ultimately, important analytical strategies for the subsequent, more hands-on sections.

1. The notions of interaction and model (selection)

1.1. Interactions

As was mentioned at the beginning of the previous chapter, multifactorial methods involve a dependent variable and two or more independent varia-

bles, not just one as in all of Chapter 4. This presence of more than one independent variable brings about potentially interesting findings, but also raises the question of how the two or more independent variables jointly relate to the dependent variable.

There are basically two different ways in which several independent and dependent variables may be related, which we will explore on the basis of the example involving constituent lengths from Chapter 1. Let us again assume you wished to study whether the lengths of constituents – captured in the dependent variable LENGTH – are correlated with two independent variables, the variable GRMRELATION (with the two levels *SUBJECT* and *OBJECT*) and the variable CLAUSETYPE (with the two levels *MAIN* and *SUBORDINATE*). Let us further assume you did a small a pilot study in which you investigated 120 constituents that are distributed as shown in Table 39.

Table 39. A fictitious data set of subjects and objects

	GRMRELATION: *SUBJ*	GRMRELATION: *OBJ*	Totals
CLAUSETYPE: *MAIN*	30	30	60
CLAUSETYPE: *SUBORD*	30	30	60
Totals	60	60	120

Let us finally assume you determined the syllabic lengths of all 120 constituents to compute the means for the variable level combinations – subjects in main clauses, subjects in subordinate clauses, objects in main clauses, objects in subordinate clauses – and obtained the following results:

- the average length of all subjects (i.e., across main and subordinate clauses) is less than that of all direct objects;
- the average length of all constituents (i.e., across subjects and objects) in main clauses is less than that of constituents in subordinate clauses.

The interesting thing is that these monofactorial results – recall from Section 3.2.2.2 that these are often referred to as *main effects* – can come in different forms. On the one hand, the effects of the two independent variables can be *additive*. That means the combination of the two variables has the effect you would expect from each main effect. Since subjects are short(er), as are constituents in main clauses, additivity predicts that main clause subjects should be the shortest constituents, and subordinate clause objects should be longest. This result, which is what H_0 would predict, is represented in Figure 57: black and grey dots indicate mean lengths of ob-

jects and subjects respectively in the two grammatical relations, but also averaged across both, and the "m" and the "s" represent the means of main and subordinate clause constituents across the two grammatical relations.

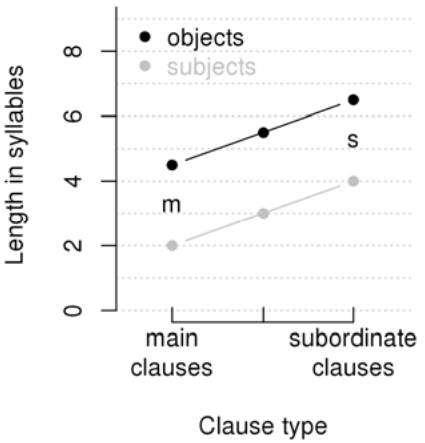

Figure 57. Interaction plot for LENGTH ~ GRMRELATION * CLAUSETYPE 1

This result is in fact perfectly additive because the two lines are perfectly parallel. That means, if I tell you that

- the difference main clause subject length minus main clause object length is -2.5 syllables;
- the difference main clause subject length minus subordinate clause subject length is -2 syllables;
- the average main clause subject length is 2 syllables,

then you can perfectly predict the average subordinate clause object length: 2 + 2.5 + 2 = 6.5.

However, with the exact same kinds of main effects, it is also possible that the two independent variables *interact*. Two or more variables interact if their joint effect on the dependent variable is not predictable from their individual effects on the same dependent variable. One such scenario is represented in Figure 58. Consider first the left panel. You can see that there are still the same kinds of main effect of GRMRELATION (subjects are again shorter than objects) and CLAUSETYPE (main clause constituents are again shorter than subordinate clause constituents), but now the lines are not parallel anymore but intersect.

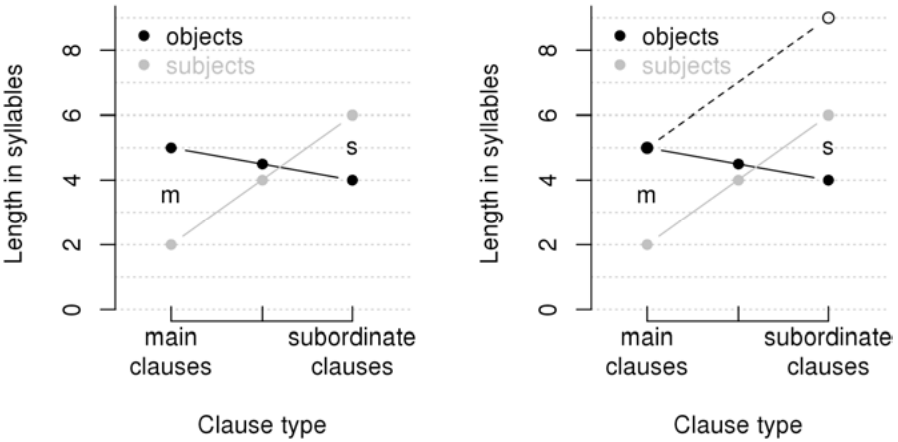

Figure 58. Interaction plot for LENGTH ~ GRMRELATION * CLAUSETYPE 2

What does that mean? It means, if I tell you that

- the difference main clause subject length minus main clause object length is -3 syllables;
- the difference main clause subject length minus subordinate clause subject length is -4 syllables;
- the average main clause subject length is 2 syllables,

then you can absolutely *not* predict the average subordinate clause object length: you would predict 2 + 3 + 4 = 9 syllables (as indicated in the right panel with the dashed line ending in a circle, which is parallel to the grey one), whereas the real average subordinate clause object length in the data is 4 syllables. *That* is an interaction: you cannot predict the average subordinate clause object length using the two main effects but need an additional interaction term that 'corrects down' the prediction from your predicted 9 to the real 4; a test of that interaction term would test whether that term is significantly different from zero or not.

Yet another kind of interaction is shown in Figure 59. Again, we have the by now familiar main effects but even though the lines do not intersect, this is still an interaction for the same reason as above. If I tell you that

- the difference main clause subject length minus main clause object length is -2 syllables;
- the difference main clause subject length minus subordinate clause sub-

ject length is -2 syllables;
– the average main clause subject length is 2 syllables,

then you can again *not* predict the average subordinate clause object length: you would predict 2 + 2 + 2 = 6 syllables (as again indicated in the right panel with the dashed line, which is parallel to the grey one), whereas the real average subordinate clause object length in the data is 8 syllables.

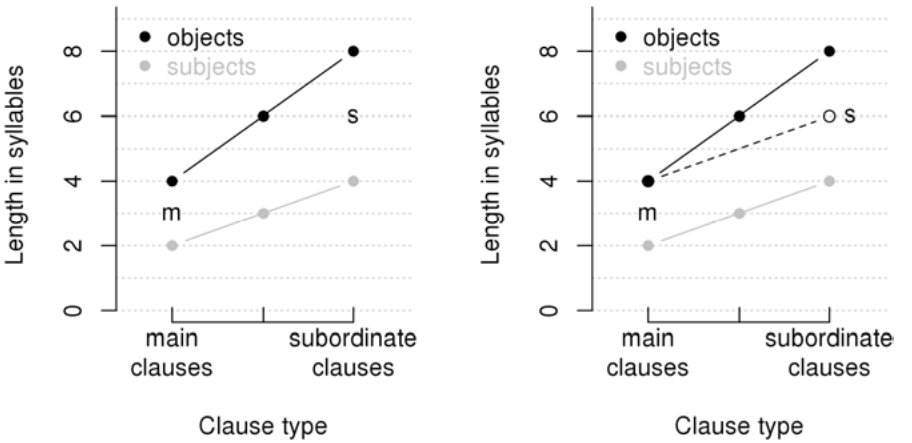

Figure 59. Interaction plot for LENGTH ~ GRMRELATION * CLAUSETYPE 3

Again an interaction: you cannot predict the average subordinate clause object length using the two main effects but need an additional interaction term that corrects up the prediction from your predicted 6 to the real 8, which again may be a significant interaction effect.

Before we move on, let me very briefly give a second example of an interaction, one that you are actually already familiar with, even if you may not have thought about it like this. The above example involved means, this one involves frequencies. Imagine you do a corpus study of 60 *of*-. vs. 80 *s*-genitives in which you try to determine whether the genitive choice is correlated with the animacy of the possessor NP (e.g., *John* in *John's car*). Imagine now you presented your results to a colleague in an overview table, but you leave out the main body of the table, as in Table 40. If you now asked your colleague to complete Table 40 without assuming anything particular going on in the data, that colleague should – maybe implicity – assume H_0 and adopt the logic of the chi-squared test and compute frequencies expected from H_0, as in Table 41.

Table 40. A fictitious data set of genitive choices (totals only)

	Animate possessor	Inanimate possessor	Totals
of-genitive			**60**
s-genitive			**80**
Totals	**70**	**70**	**140**

Table 41. A fictitious data set of genitive choices 1

	Animate possessor	Inanimate possessor	Totals
of-genitive	30	30	**60**
s-genitive	40	40	**80**
Totals	**70**	**70**	**140**

That's because if your colleague is explicitly told to not assume anything special, any deviation from Table 41 is really hard to motivate. Yes, your colleague could create something like Table 42 and say, "there's always a bit of chance variation", but … how could he possibly motivate Table 43 without assuming something special? That "something special" would be an interaction.

Table 42. A fictitious data set of genitive choices 2

	Animate possessor	Inanimate possessor	Totals
of-genitive	33	27	**60**
s-genitive	37	43	**80**
Totals	**70**	**70**	**140**

Table 43. A fictitious data set of genitive choices 3

	Animate possessor	Inanimate possessor	Totals
of-genitive	10	50	**60**
s-genitive	60	20	**80**
Totals	**70**	**70**	**140**

Thus, what in the chi-squared test scenario corresponds to the frequencies expected from H_0 are in fact the frequencies that result from assuming additive behavior of the two variables. And what a chi-squared test does is assess whether the data deviate from the distribution assuming *no* interaction so much that the *p*-value from the chi-squared test becomes < 0.05, which in turn means you will reject H_0 and assume there *is* an interaction.

This was probably a painstakingly detailed characterization but the notion of interaction is a very important one (and often misunderstood and/or

underutilized) so it is absolutely crucial you understand it. This is because the presence of a significant interaction means you cannot take the main effects of the independent variables in the interaction at face value! In Figure 58, while there is a main effect of objects being longer than subjects, the interaction shows that this is really only true in main clauses, but not in subordinate clauses. This property of significant interactions – that they qualify main effects – is one of the most important reasons for why their inclusion in a model is often essential, a topic to which we will turn now.

1.2. Model (selection)

The last section ended with a sentence using the word *model*, a word you also encountered when we discussed linear models and regression. I have used this word without a formal definition so far but you probably still had an intuitive understanding of what I meant. Now, more formally, I want to define a model as a formal characterization of the relationship between *predictors* – independent variables and their interactions – and one or more dependent variables. This 'characterization' typically comes in the form of a (regression) equation of the type you saw in Sections 3.2.3 and 4.4, and also schematically in the captions of Figure 57, Figure 58, and Figure 59, where the purpose of the regression equation is to quantify the relationship between predictors and dependent variable(s) and to generate predictions of the dependent variable(s). The development of an appropriate model, or regression equation, is called *modeling* or *model selection*, and different types of modeling are what's at the heart of most of this chapter.

One word of caution already: this chapter, as short as it is, will hopefully show that with multifactorial data, the cookbook-recipe type of approach used in Chapter 4 will not work: analyzing multifactorial data often requires leaving well-trodden paths and cherished distinctions (e.g., between exploratory and hypothesis-testing approaches). The analysis of a complex data set is much like detective work or peeling an onion, where at every step multiple avenues are possible, and I only wish I could claim I had all the solutions for all the data sets I ever explored … Ok, let's get to it!

1.2.1. Formulating the first model

The first step in model selection would seem to be the formulation of a first model, an equation that tries to model the relationship between *predictors*

and, for now, one dependent variable. However, there are a variety of threats to modeling that need to be taken into consideration. One of these has to do with something as mundane as recognizing the nature of the dependent variable: is it binary? categorical? numeric? numeric but only covering a particular range of discrete values (e.g., 0 and positive integers as with frequencies)? or just positive but with a floor as with reaction times?

We have only talked about modeling in a linear-modeling context (with the function `lm`), which is typically used when the dependent variable is numeric and spans a large range of values. However, since a linear regression will virtually always predict continuous values, it is not really well-suited to be applied to binary dependent variables (although this is still common) or categorical ones. Also, since a linear regression will virtually always predict negative values, it may not be well-suited to predict frequencies. Below, I will discuss different models for different dependent variables; thankfully, much of the logic of linear models, which you already know, can be applied to most of these cases.

A second threat is concerned with whether predictors are used on the most useful information value and scale. As for the former, there is still a lot of work out there in which continuous predictors are factorized. That means, instead of using the continuous predictor as is, researchers break it down into a categorical variable with only a few number of levels (maybe by using `cut`). This can not only lose a lot of information especially if the cutting is not done after a very careful analysis, but it also increases the *df* for the analysis, potentially making it harder to get significant results. If possible, keeping numeric variables numeric is probably a good idea.

As for the latter, the scale, it is important to realize, say, that not all numeric predictors should be entered into model as is. For example, frequency effects often operate on a logarithmic scale such that, even if $word_1$ is ten times as frequent as $word_2$, the effect of $word_1$ on the dependent variable, e.g. reaction time, may only be log (10) times as strong. Thus, what one should maybe put into the regression equation is log (frequency) (and to interpret results more easily, it may be good to use logs to the base of 2!

A third threat is concerned with the fact that probably most statistical modeling in linguistics is some sort of (generalized) linear modeling in which the effect of a predictor can be summarized with a straight regression line (in some numerical space). However, relations between predictors may differ with regard to how they are best characterized, as the two panels in Figure 60 exemplify. It's not a good idea to just force a straight regression line through the data in the right panel …

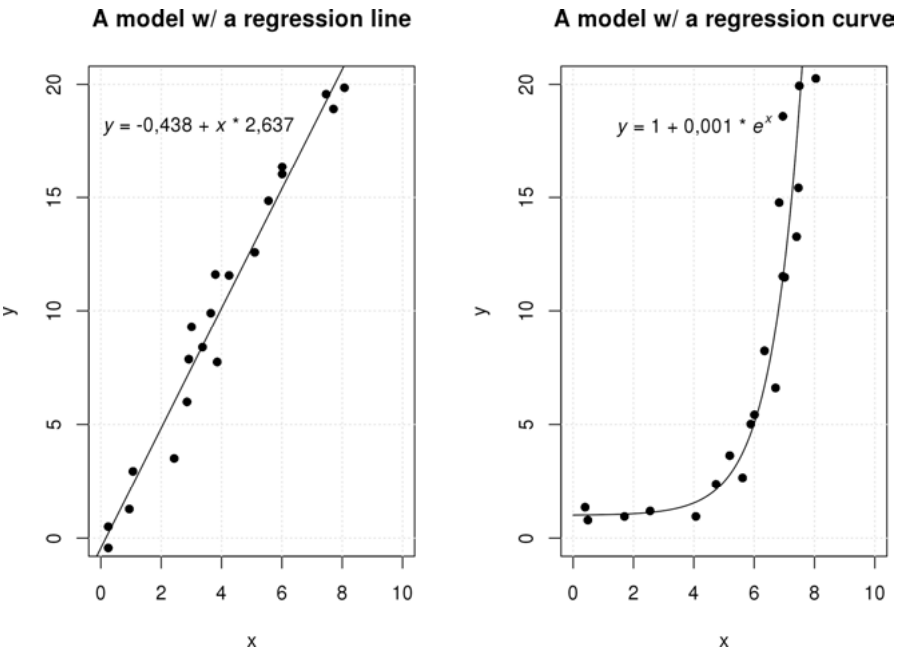

Figure 60. Models involing a regression line and a regression curve

Sometimes, data also exhibit interrupted trends that are best characterized with two or more regression lines/curves, etc. Again, just fitting one straight line through such data is risky, to say the least. The good part about the above three threats is that, if you proceed along the lines of Chapter 4, you won't usually make such mistakes. One reason why nearly every section in Chapter 4 involved some visualization was to hammer into your brain the fact that exploratory visualization should be an integral part and at the beginning of any statistical analysis, and proper visualization will reveal logarithmic relations, curvilinear trends, interrupted trends, and so on.

In addtition to these three risks I want to mention two others. One is rather trivial, one less so. The former is that your model is going to do a worse job at accounting for the data (the predictive aspect of the modeling process) and at allowing you to explain the data (the explanatory aspect of the modeling process) if you leave out important predictors. The latter is less trivial and brings us back to the notion of interaction, more specifically, to the question of whether or not to include interactions in your models. One can probably distinguish three different positions on this matter.

One is that interactions between independent variables should be included right from the start. This is because (i) if you do not include interac-

tions in the model equation, they do not get tested and you don't know whether the interactions(s) would in fact help account for the data much better, and (ii) if you included only interactions for which there was a clear theoretical motivation, it would become harder to find unexpected things; this would be a (not uncontroversial) way in which exploratory work seeps into what is usually a hypothesis-testing approach.

A second position is that you only include interactions you can motivate theoretically *a priori*. This has the above disadvantage, but the advanatage that this makes it hard to fish for something in the data.

The third position may still be the most frequent one: interactions are not included because the importance of the concept is not clear to the user or, just as bad, because the software that is being used makes including interactions hard (Varbrul is a case in point).

This issue of whether or not to include interactions is important enough to merit a short example (which you may recognize as a previous exercise). Let's assume 80 students (L1 speakers of German from two school classes A and B of 40 students each, a predictor called CLASS) had participated in one dictation in their L1 German and one in an L2 they are learning, English. Then, the numbers of mistakes in English (ENGLISH) and German (GERMAN) were counted to determine whether one can predict the numbers of mistakes made in the L2 on the basis of the numbers of mistakes in the L1 and the class the students attended. Two multifactorial models might be fitted to the data, one with the interaction between GERMAN and CLASS, one without (recall from Section 3.2.2 the two models in (58) are mere notational variants):[29]

(57) ENGLISH ~ GERMAN + CLASS

(58) a. ENGLISH ~ GERMAN + CLASS + GERMAN:CLASS

b. ENGLISH ~ GERMAN * CLASS

The results of the models in (57) and (58) are shown in Table 44 and Table 45 respectively. Both models are highly significant and explain the data really well: look at the huge and significant R^2-values. However, there are several important and interrelated problems with the model without the interaction (in (57)). First, this model does a worse job at accounting for the data than the model with it (in (58)): the bold figures in the rows called "Residual var(iance)" show how much variability in the data the models

29. I do not provide the data here but you will see this example again in one of the exercises for Chapter 5.

leave unaccounted for and you can see that that value is much higher in Table 44; a significance test would show that it is in fact significantly higher, which is another way of saying that the model in (57) is significantly worse than the one in (58).

Table 44. The results of the linear model in (57)

	SumSq	Estimate	Std. error	*t*	*p*
Intercept	23.61	2.75	1.52	1.8	0.08
GERMAN	2931.69	**1.75**	0.09	**20.1**	<0.001
CLASS	3010.30	**-8.72**	0.43	-20.37	<0.001
Residual var.	**558.68**				
overall R^2 / *p*	mult. R^2= 0.974	adj. R^2= 0.973		$F_{2, 77}$= 1416	*p*<0.001

Table 45. The results of the linear model in (58)

	SumSq	Estimate	Std. error	*t*	*p*
Intercept	24.9	2.82	1.15	2.44	0.017
GERMAN	2461.42	**1.64**	0.07	**24.29**	<0.001
CLASS	0.25	**-0.28**	1.15	-0.25	0.807
GERMAN:CLASS	241.73	-0.515	0.07	-7.61	<0.001
Residual var.	**316.95**				
overall R^2 / *p*	mult. R^2= 0.985	adj. R^2= 0.984		$F_{3, 76}$= 1661	*p*<0.001

Second, the *p*-values for the regression coefficients, or estimates, are very different. The main and crucial difference is that the model that explains the data better ((58)) says CLASS is not significant on its own but only in the interaction whereas the one that explains the data worse (in (57)) says CLASS is a significant main effect. This is not an unimportant technicality: CLASS is a binary variable, which means that, if it is a significant, its coefficient is a difference in means between the two classes, and GERMAN is a numeric variable, which means that, if it is significant, its coefficient is a slope of a regression line. Thus, what the model in (57) leads you to believe is this: students from the two classes are differently good on average (differing by 8.72 mistakes), but you can use one and the same slope for both classes to predict ENGLISH from GERMAN. This is represented in the left panel of Figure 61, with GERMAN and ENGLISH on the *x* and *y*-axis respectively, and CLASS is indicated by the letters.

However, the model in (58) says something very different, namely that there is no difference in *means* between the two classes (the *p*-value of

CLASS is huge). However, GERMAN:CLASS is significant – but what does that mean? It means that the *slope* between GERMAN and ENGLISH differs significantly across classes, which is represented in the right panel of Figure 61, and even if we did not already know from the first comment above that this model is better, the fit of the two regression lines with their separate slopes certainly seems better. Thus, the two models say very different things about what CLASS does …

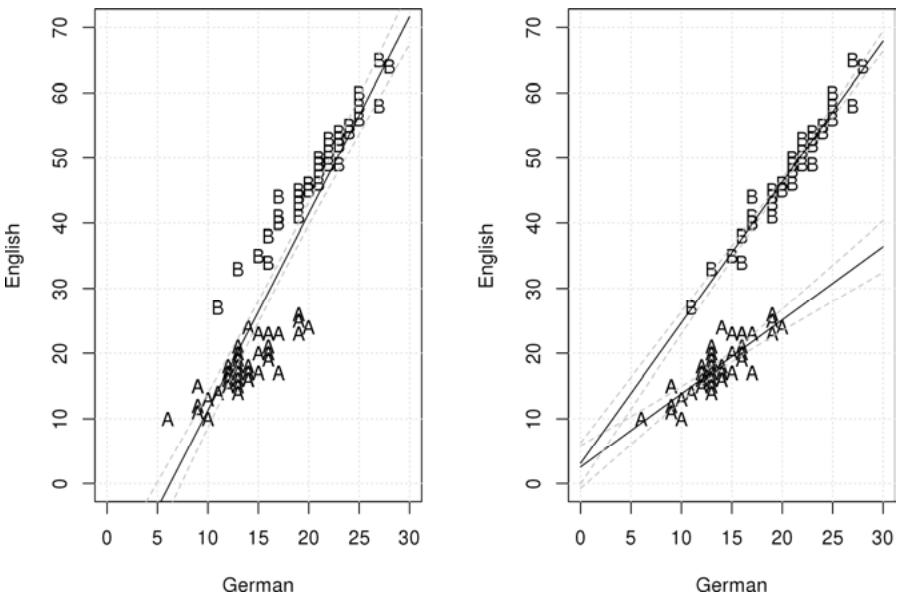

Figure 61. ENGLISH ~ GERMAN + CLASS (+ GERMAN:CLASS)

Finally and related to the previous point, the coefficients in the two models, and thus their predictions, differ a lot. The residuals of the worse model are on average more than 36% higher than those of the better one.

In sum, in this case, leaving out the interaction would leave you with a model that looks great on the surface (large R^2 and highly significant) but that is significantly worse than the model with the interaction, which tells a very different explanatory story about the data, and which is much worse at 'predicting' the data points. Against this background, it is amazing how often interactions are still not explored (properly).

One thing I have seen is that researchers seem aware of such issues but that their tools are not equipped to handle interactions (or continuous data) well; again, Varbrul is a case in point. So how might they then try to ad-

dress interactions? By fitting a separate model for each class:

(59) $\text{ENGLISH}_{\text{CLASS A}} \sim \text{GERMAN}_{\text{CLASS A}}$
(60) $\text{ENGLISH}_{\text{CLASS B}} \sim \text{GERMAN}_{\text{CLASS B}}$

If one does that, one does indeed get two significant simple regressions and the correct slopes of 1.13 for class A and 2.16 for class B. But why is this still a bad idea?

THINK
BREAK

Well, with this approach how do you know whether the difference between these two slopes is significant or not? The interaction does not show up in either model in (59) and (60) so the slopes never get compared to each other so you don't get a *p*-value so you don't know whether that is a significant difference or not. I have seen plenaries and papers and more where Varbrul weights for different time periods were compared to each other without any test of whether the difference between the Varbrul weights of different time periods were significant and thus indicative of change over time or not … You either have to include the interaction and get a *p*-value for it, or you have to at least check the confidence intervals of the slopes for whether they do not overlap (which, here, they do not).

In sum, I advise you to consider carefully the nature of the variables involved,to spend a considerable amount of time exploring your data (especially visually) before you start doing anything else, and to be very aware of the potential importance of interactions.

1.2.2. Selecting a final model

Once the above issues have been considered, a first model is formulated, and often this model is what is called a *maximal model*, i.e. a model including all independent variables, all of their interactions (often only up until interactions of three independent variables, because interactions of an even higher order are extremely difficult to understand). However, this is usually only the starting point since the maximal model usually contains predictors that do not contribute enough to the model, and since the famous dictum

called Occam's razor (*entia non sunt multiplicanda praeter necessitatem*) essentially requires you to discard predictors that don't pull their own weight or, more formally, do not contribute enough to the model's success.

Model selection is then influenced by two parameters: the *direction* of model selection and the *criterion* determining whether or not a predictor gets to be in the model. As for the former, there are three approaches:

- *backward selection*: here, you start with the maximal model as outlined above and successively test whether you have to discard predictors which do not contribute enough to the model. The selection process ends when no predictor can be discarded anymore with making the model too much worse or when no predictors are left in the model. The elimination of predictors begins with the highest level of interactivity and proceeds downwards in the direction of main effects, and you cannot discard a predictor that participates in a required higher-order interaction. That means, you cannot delete even an insignificant predictor B if the interaction A:B is significant.
- *forward selection*: here, you start with a very small model (maybe even just one that consists of the overall mean) and successively test whether you can add predictors. The selection process ends when no addition of a predictor improves the model enough anymore or when all available predictors are already in the model. The addition of predictors begins with main effects and moves up to higher-order interactions (if their main effects have already been included, as above).
- *bidirectional*: here, you start with some model and allow a usually automatic algorithm to add and subtract predictors as warranted.

I think the first approach is most widely used in linguistics but there are also good arguments not to do model selection at all (cf. Harrell 2001: Section 4.3 or Faraway 2005: Section 8.2).

As for the latter, I have been intentionally vague above when it came to describing when predictors are added or discarded: I always just said "good enough." This is because there are again at least two possible ways (who would want life to be easy …):

- a *significance-based approach*, according to which a predictor can be added to a model if it makes the model significantly better, and according to which a predictor should be discarded if its deletion does not make the model significantly worse.
- a *criterion-based approach*: the *AIC* (Akaike Information Criterion), for

instance, is one measure that relates the quality of a model to the number of predictors it contains (and thus operationalizes Occam's razor). If two models explain data equally well, then the model with fewer predictors will have a smaller *AIC*. Thus, in this approach, a predictor can be added to, or deleted from, a model if that lower *AIC*.

Once the model selection process has been completed, you have what is sometimes called the minimal adequate model, which can then be explored in terms of (i) whether the model as a whole is significant or not and how well it accounts for the data and (ii) what each predictor in that model contributes to the model: is it significant, what is the direction of its effect(s), and what is the strength of its effect(s). After this lengthy, but necessary theoretical introduction, the following sections will discuss all these matters – (different types of) regression models, main effects, interactions, model selection. prediction accuracy etc. – on the basis of many practical examples. Section 5.2 discusses linear models for (multiple) linear regression, ANOVAs, and ANCOVAs.

Recommendation(s) for further study
– Good and Hardin (2012: Part III) and Crawley (2007: Ch. 9)

2. Linear models

In Sections 3.2.3 and 4.4.1, we looked at how to compute and evaluate the correlation between an independent ratio-scaled variable and a dependent ratio-scaled variable using the Pearson product-moment correlation coefficient *r* and linear regression. In this section, we will extend this to the case of multiple independent variables. The data we will explore involve the question of how to predict speakers' reaction times to nouns in a lexical decision task and involves the following variables:[30]

– a dependent variable, namely the reaction time (RT) to words in a lexical decision task REACTTIME, whose correlation with the following in-

30. The words (but not the reaction times) are borrowed from a data set from Baayen's comprehensive (2008) book; the other characteristics of these words were taken from, or made up / modified based on, the MRC Psycholinguistic Database; cf. <http://www.psy.uwa.edu.au/ mrcdatabase/mrc2.html> for more detailed explanations regarding the variables in general.

dependent variables you are interested in (in this case, the dependent variable is an average of reaction times, which would not normally be the case; this is for expository reasons only and doesn't matter here);
- an independent numeric variable FREQUENCY, which corresponds to their logged frequency (according to Kučera and Francis 1967);
- an independent categorical variable FAMILIARITY, which is an index summarizing subjects' rated familiarity with the referent of the word;
- an independent binary variable IMAGEABILITY, which is an index summarizing subjects' rated imageability of the referent of the word;
- an independent numeric variable MEANINGFULNESS, which indicates subjects' average meaningfulness rating of the stimulus word.

This is the overall procedure of the linear modeling process we will use:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a linear model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of observed and/or predicted values
- Testing the main assumption(s) of the test:[31]
  - the variances of the residuals are homogeneous and normally distributed in the populations from which the samples were taken or, at least, in the samples themselves
  - the residuals are normally distributed (with a mean of 0) in the populations the samples come from or, at least, in the samples themselves

As you can see, in this section, we will test some assumptions of the linear modeling only after we have fit a model, which is because you can only check residuals when you have a model from which they can be computed. Also, this section is quite different from those in Chapter 4. It has been my experience – both in teaching and in my own research – that one of the greatest difficulties in linear modeling is not to get a significant result, but to understand what the regression coefficients (or estimates, I will use these terms interchangeably) in the results mean, a problem aggravated by the

31. There are other requirements – e.g., independence of residuals and absence of collinearity (!) – but for reasons of space I cannot discuss them all. See Fox and Weisberg (2011: Ch. 6) and Field, Miles, and Field (2012: Section 7.7) for exhaustive discussion.

fact that few books (i) say very explicitly in a language that beginners understand what the output means and (ii) discuss what the output means for two different ways to do linear modeling. In order to address both of these issues, this section will walk you through six fairly simple linear models that differ in terms of the predictors they involve to show you exactly – both numerically and visually – what regression coefficients mean. In addition, each of these linear models will be computed in two ways. One is a frequently-used standard in some commercial software applications that are unfortunately still in wide use, the other is the standard way in R.

Before we begin with the modeling, it is important to you to realize that, if this was a real study, you would not *run* many different models on the data to test different but overlapping hypotheses as I will do here. I will walk you through these models only so that you see how these are fit, interpreted, and visualized – what *you* would do if this was a real study is a model selection process of the type discussed in Section 5.2.7.

Let's begin by formulating the hypotheses, which will be applicable to all linear models in this section. We use an extension of the coefficient of determination r^2, namely its multiple regression equivalent multiple R^2:

H_0: There is no correlation between REACTTIME on the one hand and the predictors (independent variables and their interactions) on the other hand: multiple $R^2 = 0$.

H_1: There is a correlation between REACTTIME on the one hand and the predictors (independent variables and their interactions) on the other hand: multiple $R^2 > 0$.

Let us now load the data (from <_inputfiles/05-2_reactiontimes.csv>) such that the words for which have data become the row names, which is useful for some plots (output not shown):

```
> RTs<-read.delim(file.choose(), row.names=1)¶
> summary(RTs)¶
```

The summary shows you that this is a very small data set – a real study would better be based on more data. In addition, we find something that is only too realistic, namely that some variables have missing data, marked as NA, as they should be. For now, we will adopt a quick and dirty solution and make use of the fact that R's linear modeling function `lm` will automatically discard those cases of variables in the model that have missing data.

Before we begin with the modeling, there are two ways in which data

can often be prepared for better analysis. One of these is that it is sometimes useful to *z*-standardize numeric variables (with `scale`, recall Section 3.1.4), which may help with the problem of collinearity (the undesirable phenomenon that several of your predictors are highly correlated) and which may help with interpreting the results because the mean of standardized predictors is zero, which.e.g., makes intercepts and regression coefficients easy to understand. (On the other hand, it can also make results harder to understand because we lose the original units of the scale.) We will therefore not use this here, but it's a good thing to keep in mind for later.

The second thing we are going to do has to do with factors (and now you will see why I talked about them so much above). If you look at the summary output, you will see that the factor `FAMILIARITY` has levels that are ordered alphabetically but that that order is not compatible with the ordinal information that the levels communicate. We would want either lo, med, and hi, or hi, med, and lo, but not hi, lo, med. Thus, for both `FAMILIARITY` and `IMAGEABILITY`, we reorder their levels in an ordinally reasonable and homogeneous way:

```
> RTs$FAMILIARITY< factor(RTs$FAMILIARITY, levels=
   levels(RTs$FAMILIARITY)[c(2, 3, 1)])¶
> RTs$IMAGEABILITY<-factor(RTs$IMAGEABILITY, levels=
   levels(RTs$IMAGEABILITY)[c(2, 1)])¶
> summary(RTs)¶
```

Since graphical or tabular exploration (e.g., with boxplots or ecdf plots). which I strongly recommend you always do on data, does not really yield anything else in need of correction/preparation, we can now attach `RTs` and load a few packages we will use. Finally, the code file defines a few functions we will use a few times – `se.mean`, `ci.mean`, and `error.bar` – so just copy and paste that code into R so that you can use these functions below.

```
> attach(RTs)¶
> library(aod); library(car); library(effects); library(gvlma);
   library(multcomp); library(rgl)¶
```

2.1. A linear model with a binary predictor

Although this first linear model is the simplest of all, this section will be a bit longer because all the things having to do with linear models will show up for the first time. So, don't despair, everything else later will be shorter. To test whether IMAGEABILITY is correlated with REACTTIME, we fit what

is about the simplest possible linear model. However, to get results that are comparable with what some commercial software outputs, we first set the way R computes contrasts as shown here (more on that in a while), then we fit a linear model (where we also tell R which data frame the variables are from with the `data` argument), and then we inspect the output:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY, data=RTs)¶
> summary(model.01)¶
                                                                    […]
Residuals:
    Min      1Q  Median      3Q     Max
-84.629 -40.016   2.145  26.975 160.799

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    620.666      6.998  88.693   <2e-16 ***
IMAGEABILITY1   12.987      6.998   1.856   0.0693 .
---                                                                 […]
Residual standard error: 50.5 on 51 degrees of freedom
  (24 observations deleted due to missingness)
Multiple R-squared: 0.06326,     Adjusted R-squared:  0.0449
F-statistic: 3.444 on 1 and 51 DF,  p-value: 0.06925
```

Let's start at the bottom: the model as a whole is not significant, as the *p*-value shows, which in turn is computed from the *F*-value at *df*=1, 51 (here it is: `pf(3.444, 1, 51, lower.tail=FALSE)¶`). The multiple correlation between IMAGEABILITY and REACTTIME, multiple R^2, ranges theoretically from 0 to 1 and quantifies the variability accounted for, so a value of 0.06326 is really small. In addition, the value usually reported is the adjusted R^2. This R^2-value is adjusted such that you incur a slight penalty for every predictor included in your model. Thus, if, in a desperate attempt to explain more variability, you were to add a useless variable into the model, then it is very likely that whatever little bit of random variation that useless variable accounts for will be eaten up by the penalty. Thus, this adjustment brings Occam's razor into modeling. Obviously, adjusted R^2 is also really small. Then, there is a warning that R deleted 24 observations because these cases had NA in the variables in the model.

We ignore the residual standard error and briefly skip to the top of the output,[32] where we get a summary output regarding the residuals and we can already see that these are hardly normally distributed – whatever we learn here must be interpreted cautiously (We'll get back to this.)

32. The residual standard error is the root of the quotient of the residual sums of squares divided by the residual *df* (in R: `sqrt(sum(residuals(model.01)^2)/51)`.

While we have not talked about what the coefficients mean, let me already point out the obvious: they are just estimates, which is how R labels them, which means you can get confidence intervals for them, and the fact that the confidence interval for IMAGEABILITY includes 0 already suggests that, whatever it is – to be discussed in a moment – it's not significant:

```
> confint(model.01)¶
                   2.5 %    97.5 %
(Intercept)   606.617138 634.71498
IMAGEABILITY1  -1.061602  27.03624
```

Before we turn to the coefficients and their *p*-values, let us run two more lines of code, which are very useful for predictors with more than one *df*, i.e. predictors that are neither binary nor numeric (i.e., this does not apply here, I mention it here anyway for the sake of consistency).

```
> drop1(model.01, test="F")¶
Single term deletions
Model:
RT ~ IMAGEABILITY
             Df Sum of Sq    RSS    AIC F value  Pr(>F)
<none>                    130059 417.69
IMAGEABILITY  1    8783.6 138843 419.15  3.4443 0.06925 .
---                                                    […]
> Anova(model.01, type="III")¶
Anova Table (Type III tests)
Response: RT
               Sum Sq Df   F value  Pr(>F)
(Intercept)  20060844  1 7866.4268 < 2e-16 ***
IMAGEABILITY     8784  1    3.4443 0.06925 .
Residuals      130059 51                                […]
```

These functions are important ways to get *p*-values for predictors. The first, `drop1`, looks at all the predictors in the model and checks which predictor could theoretically be deleted from the model at this stage in the model selection process, and for the predictors that could be deleted at this point, it returns a *p*-value for the test of the original model, `model.01`, against the model that you would get without that predictor. The second, `Anova`, is available from the library `car`. It computes a *p*-value for predictors that is the same as commercial software returns by default.[33] As you

33. The issue of sums of squares (the `type="III"` argument) is hotly debated. I will not engage in the discussion here which approach is better but use `type="III"` for reasons of comparability with other software even if `type="II"` may often be more useful; see Crawley (2002: Ch. 18, 2007: 368ff.), Larson-Hall (2010: 311-313), Fox and Weisberg (2011: Sections 4.4, 4.6), Field, Miles, and Field (2012: 475f.) and the R-help list.

can see, both return the already known *p*-value for the only predictor.

With this output, let us now turn to the coefficients. First the simpler part, the *p*-values, then, second, the coefficients. The *p*-value for the intercept is usually disregarded: it tests the H_0 that the intercept is 0, but there are few applications where that is relevant. More interesting is the *p*-value for the predictor IMAGEABILITY. (In fact, R writes `IMAGEABILITY1`, I will explain that in a moment.) In this simplest of cases, where our model only has one binary predictor, the *p*-value there is the same as the *p*-value of the whole model, and the same of that predictor in the `drop1` and in the `Anova` output: 0.06925. So, the predictor does not have a significant effect and, in a sense, the output of `drop1` says that most intuitively because what `drop1` is essentially saying is "if you drop IMAGEABILITY from `model.01`, then the resulting model is not significantly worse (p=0.06925)." A different way to view this is as showing that the regression coefficient is not significantly different from 0. All this is identical to what you get from a *t*-test.

While this model/predictor is not significant, we will proceed with the discussion and plotting as if it were, because at this point I want to show you how such a model output is interpreted and plotted; a more realistic model selection process follows in Section 5.2.7.

So – finally – what do the estimates mean, the 620.666 of `(Intercept)` and the 12.987 for `IMAGEABILITY1`? I recommend to approach this question on the basis of the values that the model predicts as in Section 4.4.1:

```
> preds.hyp<-expand.grid(IMAGEABILITY=levels(IMAGEABILITY));
   preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-predict(
   model.01, newdata= preds.hyp, interval="confidence");
   preds.hyp¶
  IMAGEABILITY PREDICTIONS    LOWER    UPPER
1           lo    633.6534 612.5138 654.7929
2           hi    607.6787 589.1691 626.1884
```

Thus, `model.01` predicts that, when the word is of low imageability, then people's reaction times will be about 26 ms slower than when the word is of high imageability. Just to make this clear: this means the model makes only two different predictions: when IMAGEABILITY is low, it always predicts an RT of 633.6534, and when IMAGEABILITY is high, it always predicts an RT of 607.6787, and these two predicted values are also the observed means: try `tapply(RT, IMAGEABILITY, mean)¶`. Note also how much the confidence intervals of the two predictions overlap.

If we look at `preds.hyp`, you may already suspect what the regression estimates mean. When you compute the linear model as we did here, i.e. with sum contrasts!, then these two values mean the following:

- the intercept, 620.666, is the unweighted (!) mean of the means of the dependent variable, when it is grouped by the independent variable. That is, 620.666 is the mean of of 633.6534 and 607.6787, and that is an unweighted mean because it does not take into consideration that the two levels of IMAGEABILITY are not equally frequent.
- the coefficient for `IMAGEABILITY1`, 12.987, is what you have to add to the intercept to get the predicted RT for the first level of IMAGEABILITY (hence the 1): 620.666 + 12.987 = 633.653. (And since the intercept is the mean of means, if you subtract the coefficient from the intercept, you get the predicted RT for the second level of IMAGEABILITY: 620.666 - 12.987 = 607.679.)

This is visually represented in Figure 62.

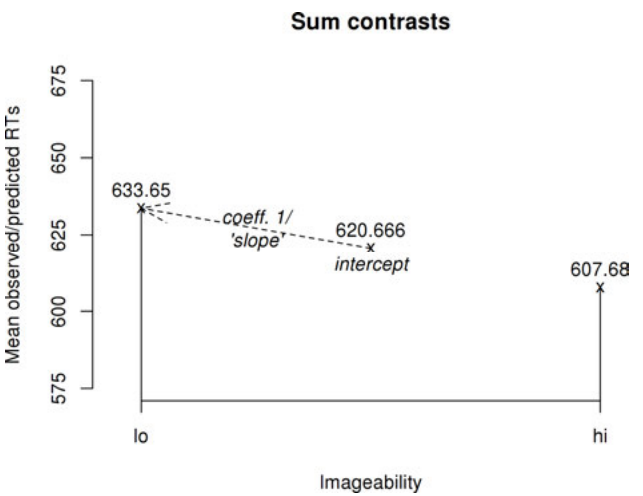

Figure 62. The regression estimates of `model.01` with sum contrasts

Now, you may wonder why it says "slope" in Figure 62. This is because you can conceptualize the intercept as an *x*-axis value of 0 and IMAGEABILITY:1 as an *x*-axis value of 1, which is pretty much what linear modeling does under the hood: For numeric variables, effects are given as slopes which represent how much the predicted *y*-value changes for every unit change on the *x*-axis anyway, but with the above perspective you can also understand coefficients for factor levels (e.g., 12.987) as slopes.

Finally, while this particular model is so simple that the coefficients etc. can be understood without any visualization, this can quickly change so I

will even here present two ways in which the data can be visualized. The code to generate the plots in Figure 63 is in the code file. The left is an ordinary barplot of means, the only thing I added are the confidence intervals for the means; the right plot is a very easy-to-generate effect plot.

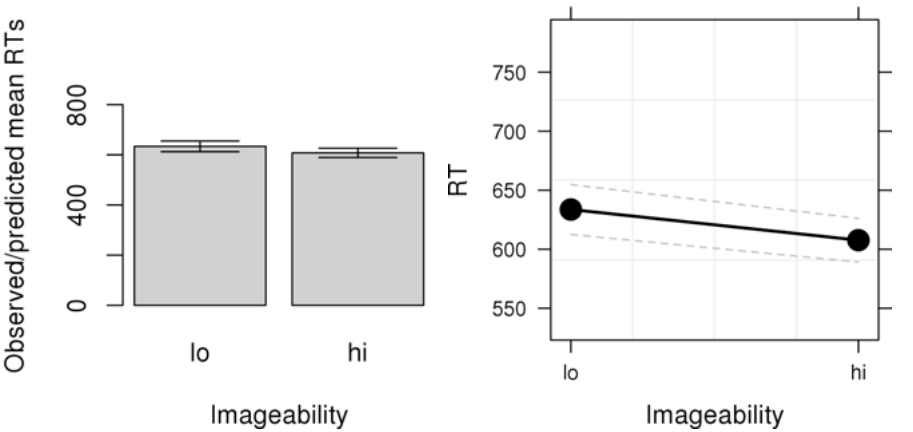

Figure 63. The effects of `model.01`: barplot with observed/predicted means and their 95% confidence-interval bars (left panel); effects plot from the library `effects` (right panel)

All the above was how much commercial software would report the results. However, the standard way in R is actually a bit different, thankfully it is really only a bit … Since I want you to know R's standard approach and since that approach will help you understand logistic regression later, I will now discuss it very briefly. The only real difference in execution for this second, R's standard approach, is that you now use R's default contrasts, treatment contrasts. If you then generate the model again, the R^2-values, the overall p-value, most is the same but not the coefficients:

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY, data=RTs)¶
> summary(model.01)¶
                                                                […]
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     633.65      10.53  60.177   <2e-16 ***
IMAGEABILITYhi  -25.97      14.00  -1.856   0.0693 .
---                                                             […]
> confint(model.01)¶
```

After what we have done above, you probably immediately see what the

intercept and the coefficient for `IMAGEABILITYhi` represent:

- the intercept, 633.65, is the observed/predicted mean of the dependent variable, when the independent variable IMAGEABILITY is its first level, *LO*.
- the coefficient for `IMAGEABILITYhi`, -25.97 is what you add to the intercept to get the predicted RT for the second level of IMAGEABILITY (hence the *HI*): 633.65 + -25.97 = 607.68; the *p*-value shows that the difference between the intercept (representing IMAGEABILITY: *LO*) and this predicted RT for IMAGEABILITY:*HI* is not significant.

This is also represented in Figure 64, where, as discussed above, the annotation of $x = 0$ and $x = 1$ motivate the use of the word *slope* in the plot.

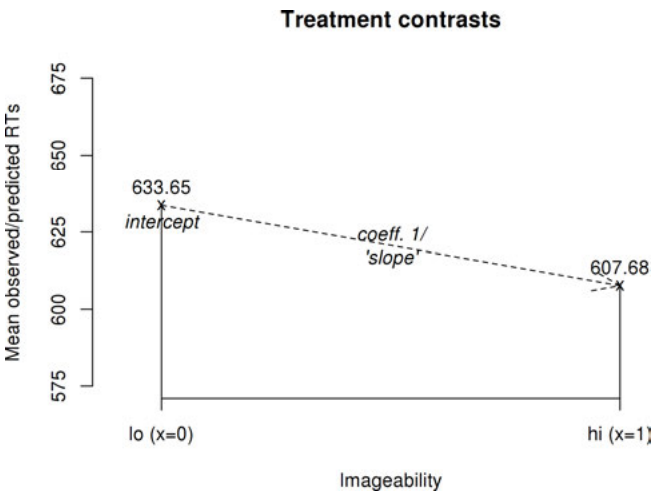

Figure 64. The regression estimates of `model.01` with treatment contrasts

As you can see, in this simple case both approaches yield different coefficients, but they amount to the same significance tests (with `drop1` again, see the code file) and the same predictions (in the new `preds.hyp`; see the code file). Also, note that I provide some extra code to get *p*-values for coefficients using `wald.test` and `glht` in the code file. You should always run that, too, since it will be very useful later; much later you may want to explore Bretz, Hothorn, and Westfall (2011).

You can summarize the results as follows: "A linear model was fit with REACTTIME as the dependent variable and IMAGEABILITY (low vs. high) as

the independent variable. The model was not significant ($F = 3.444$, $df_1 = 1$, $df_2 = 51$, p=0.069). There was only a marginally significant tendency such that low and high imageability correlated with slower and faster reaction times respectively. [Show graph(s)]."

2.2. A linear model with a categorical predictor

In this section, we still cover only one predictor – so actually, we are still not doing multifactorial analysis – but we make the model a bit more complex by studying a predictor with three levels (FAMILIARITY), which means you could not do a *t*-test anymore.[34] First again the approach using sum contrasts (from now on, I will not show all the output anymore):

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

This model is significant: the overall p-value is < 0.001. Since this is also a model with only one predictor, you know that this is now also the p-value for that one predictor. However, if you look at the table of coefficients, you don't find it there. Instead you have an intercept and then two quite different p-values. How do you get a p-value for FAMILIARITY other than by looking at the overall p-value (e.g., when you have more than one predictor)? This is a case where `drop1` and `Anova` are needed because – remember from above – here the (only) predictor has more than 1 *df* because it is neither binary nor numeric. Thus you use `drop1` and `Anova`:

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

There's the p-value for FAMILIARITY, and this time you can see how the significance-based and the criterion-based approach agree: FAMILIARITY is significant and taking it out increases *AIC* considerably.

34. Incidentally, this section as well as the previous cover linear models that some would refer to as ANOVAs, analyses of variance. However, since the underlying approach between linear regressions with only numerical independent variables, ANOVAs with only categorical independent variables, and ANCOVAs with both categorical and numeric independent variables is the same – in R they are all fit with `lm` – I will not topicalize the differences between these methods but rather focus on their commonalities.

So, what do the estimates mean? Again, we approach this via the predicted values. It turns out that there is a nice ordinal effect: as FAMILIARITY increases, RTs go down, which makes sense.

```
> preds.hyp<-expand.grid(FAMILIARITY=levels(FAMILIARITY));
   preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-
   predict(model.01, newdata=preds.hyp,
   interval="confidence"); preds.hyp¶
```

From `preds.hyp`, you can again guess what the estimates mean, and this is also visualized again in Figure 65:

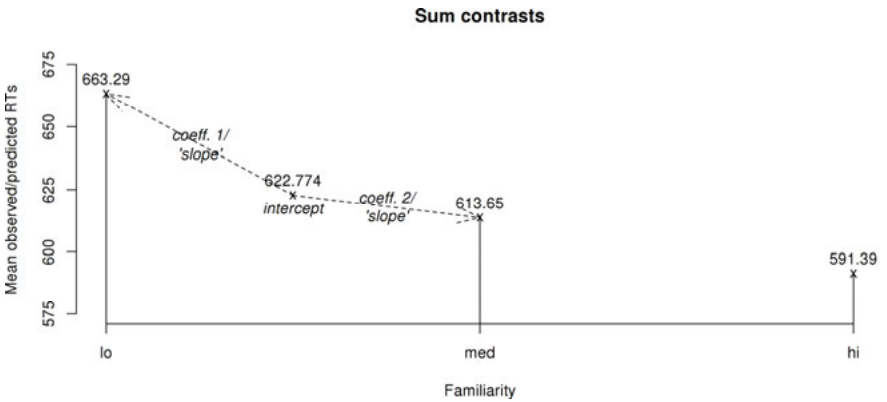

Figure 65. The regression estimates of `model.01` with sum contrasts

- the intercept, 622.774, is the unweighted (!) mean of the means of the dependent variable, when it is grouped by the independent variable. That is, 622.774 is the mean of 663.2880, 613.6471, and 591.3879, and that is an unweighted mean because it does not take into consideration that the levels of FAMILIARITY are not all equally frequent;
- the coefficient for `FAMILIARITY1`, 40.514, is what you add to the intercept to predict the RT for the first level of FAMILIARITY (hence the 1);
- the coefficient for `FAMILIARITY2`, -9.127, is what you add to the intercept to predict the RT for the second level of FAMILIARITY;
- and if you subtract both coefficients for FAMILIARITY from the intercept, you get the predicted RT for the third level of FAMILIARITY.

Note that you do not get *p*-values for *all* differences between the intercept and the levels, and sometimes you may want to run a variety of tests

on differences between means. One (rather conservative) way to approach this question involves the function `TukeyHSD`.

```
> TukeyHSD(aov(model.01), ordered=TRUE)¶
```

The main argument of this function is an object created by the function `aov` (an alternative to `anova`), which in turn requires the relevant linear model as an argument. As a result, you get a table for all three comparisons you can make between three means. You get the differences between the means, the lower and the upper confidence intervals for the differences, and *p*-values that have been adjusted for the fact that you are suddenly performing three significance tests on the same data set. Why would *p*-values have to be adjusted for that?

THINK
BREAK

The point of a significance level was to make sure that, if you accept an H_1, your probability to do that *in*correctly was < 0.05. Now, if you reject two independent H_0 at each $p = 0.05$, what is the probability that you do so correctly both times? It's 0.9025, i.e. 90.25% Why? Well, the probability you are right in rejecting the first H_0 is 0.95. But the probability that you are *always* right when you reject H_0 on two independent trials is $0.95^2 = 0.9025$. This is the same logic as if you were asked for the probability to get two sixes when you simultaneously roll two dice: $^1/_6{}^2 = {}^1/_{36}$. The probability that you are *always* right when you reject H_0 on three independent trials is $0.95^2 = 0.857375$. In fact if you look at 13 H_0s, then the probability that you do not err once if you reject all of them is in fact dangerously close to 0.5: $0.95^{13} \approx 0.5133$, a.k.a. pretty far away from 0.95. Thus, the probability of error you use to evaluate each of *n* H_0s should not be 0.05 – it should be smaller so that when you perform all *n* tests, your overall probability to be always right is 0.95. Thus, if you want to test *n* H_0s, you must use $p = 1\text{-}0.95^{(1/n)}$. For 13, that means $p \approx 0.00394$. Then, the probability that you are right on any one rejection is 1-0.00394 = 0.99606, and the probability that you are right with all 13 rejections is $0.99606^{13} \approx 0.95$. A shorter heuristic that is just as conservative (actually, too conservative) is the Bonferroni correction. It consists of just dividing the desired significance level – i.e., usually 0.05 – by the number of tests – here 13. You get $^{0.05}/_{13} \approx$

0.003846154, which is close (enough) to the exact probability of 0.00394 computed above. Thus, if you do multiple *post hoc* tests on a dataset, you usually adjust the significance level, which makes it harder for you to get significant results just by fishing around in your data, which should motivate you to formulate reasonable H_1s beforehand rather than excessive *post hoc* testing.

Back to the data at hand: we can see that the difference between medium and high levels of FAMILIARITY is not significant, but the other two differences are. What does that mean?

THINK
BREAK

It means that Occams razor would require that you now test whether you need to uphold the difference between medium and high familiarity or whether you must conflate the two, and we will do this in Section 5.2.7.

As a last step for this model, you can generate some plots again, and the code file will show you how to generate plots like Figure 63 for this model.

Now, let us very briefly explore this same model, but now with R's default of treatment contrasts again:

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY, data=RTs)¶
> summary(model.01)¶
                                                                    […]
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       663.29      13.23  50.118  < 2e-16 ***
FAMILIARITYmed    -49.64      15.59  -3.185 0.002449 **
FAMILIARITYhi     -71.90      18.72  -3.842 0.000334 ***
---                                                                 […]
> confint(model.01)¶
```

After what we have done above, the estimates are probably clear:

- the intercept, 663.29, is the observed/predicted mean when the independent variable FAMILIARITY is its first level, lo.
- the coefficient for `FAMILIARITYmed`, -49.64 is what you add to the intercept to predict the RT for the second level of FAMILIARITY.
- the coefficient for `FAMILIARITYhi`, -71.90 is what you add to the intercept to predict the RT for the third level of FAMILIARITY.

This is also represented in Figure 66, where, as discussed above, the annotation of $x = 0$ and $x = 1$ (two times, one for each estimate) help motivate the use of the word *slope* in the plot. Thus, in some sense, it's all the same as before in Section 5.2.1 and you can summarize this section's model along the lines of the one above.

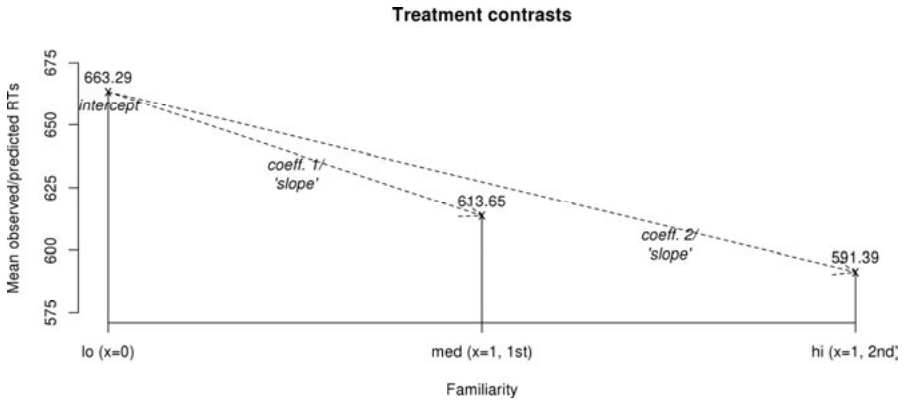

Figure 66. The regression estimates of `model.01` with treatment contrasts

2.3. A linear model with a numeric predictor

We are still only preparing for multifactorial models. In the last two monofactorial ones, the only predictor was a (binary or categorical) factor and, correspondingly, its effects were differences between means. However, we also began to approach that as a slope, by conceptualizing differences between means as slopes from the *y*-value at a reference level (at $x = 0$) to a *y*-value at a level defined as $x = 1$. In this section, we will very briefly revisit the case of a numeric predictor, i.e., what we discussed in Section 4.4.1. One nice thing is that, with just an interval-scaled predictor, we do not have to cover two types of contrasts. We are going to look at the correlation between FREQUENCY and REACTTIME.

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~FREQUENCY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

By now we have studied such cases both with `cor.test` and `lm` already so I won't go over all the results in detail again. Suffice it to say, that the

model is significant because its only predictor is, etc. Since the predictor is numeric, we do not really need the following two lines, but just to entrench them in your mind, here they are again, and they return the same *p*-value.

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

To determine what the estimates mean, we follow the same strategy as before and compute predictions for values from the attested range:

```
> preds.hyp<-expand.grid(FREQUENCY=floor(min(FREQUENCY)):
   ceiling(max(FREQUENCY))); preds.hyp[c("PREDICTIONS",
   "LOWER", "UPPER")]<-predict(model.01, newdata=
   preds.hyp, interval="confidence"); preds.hyp¶
```

You can recognize what you hopefully already guessed from above:

- the intercept, 667.03, is the predicted RT when the independent variable FREQUENCY is 0.
- the coefficient for FREQUENCY, -24.266 the increase in the predicted RT (i.e., given the minus, a decrease) for each unit increase of FREQUENCY.

As usual, you should plot the data to get an impression of the fit, and the code file provides a few examples of how you could do that.

Now that we have covered the basics, we can finally move on to multifactorial linear models. While this introductory part may have seemed long, having covered everything in that much detail will make things easier now.

2.4. A linear model with a two categorical predictors

We begin with a model in which we try to predict REACTTIME on the basis of two independent categorical variables, IMAGEABILITY and FAMILIARITY, and their interaction, IMAGEABILITY:FAMILIARITY. As before, we begin with a model based on sum contrasts. Recall the notation using the asterisk to say 'all these main effects and their interactions':

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY*FAMILIARITY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

The output becomes more complex … The model as a whole is significant ($p = 0.01138$), we can see that IMAGEABILITY is not significant, but we don't have individual *p*-values for FAMILIARITY and the interaction IMAGEABILITY:FAMILIARITY. Thus, before we try to understand the coefficients/estimates, a quick look at `drop1` and `Anova`:

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

This time, the output of the two is differently comprehensive. The output of `drop1` follows the above logic of backwards model selection and only returns *p*-values for those predictors that could be dropped *at this time*. Since there is an interaction of two variables and nothing more complex than that in the model, you can drop that interaction, but you cannot at this stage drop any of the variables from the interaction as long as the interaction is still in the model. Thus, `drop1` only returns the *p*-value for the model with vs. without the interaction and since the interaction is not significant, one should drop it (following Occam's razor).

The output of `Anova` is more comprehensive and returns *p*-values for all predictors in the model; you can recognize the *p*-values for IMAGEABILITY from the `summary(lm())` output, and the one for the interaction from the `drop1` output. We will not drop the interaction now because at this point I want to show you how such a model output is interpreted and plotted; again, the more realistic model selection process follows in Section 5.2.7.

Now to the predictors and their estimates:

```
> preds.hyp<-expand.grid(IMAGEABILITY=levels(IMAGEABILITY),
   FAMILIARITY=levels(FAMILIARITY)); preds.hyp[
   c("PREDICTIONS", "LOWER", "UPPER")]<-predict(model.01,
   newdata=preds.hyp, interval="confidence"); preds.hyp¶
```

I will not explain every coefficient in detail here –see the code file for painfully detailed definitions of each estimate – for two reasons. First, to save space: you will see how long and convoluted the definition of the estimates in the code file can become. Second, the whole point of generating `preds.hyp` is that we don't *have* to look at the coefficients that much. Of course you should still understand the explanation in the code file but in actual practice understanding the coefficients of a model with, say, five significant 3-way interactions and 10 other predictors on the basis of the coefficients is pretty much impossible. Thus, read the explanation of the coefficients in the code file carefully, run the code there to verify my ex-

planations, and try to recognize their effects in the data, but for now we will explore the model on the basis of its predictions, which show that

- the observed/predicted means don't do much as IMAGEABILITY changes (averaging across FAMILIARITY);
- the observed/predicted means decrease as FAMILIARITY increases (averaging across IMAGEABILITY)'
- there is a hint of an interaction (but we know from above it is not significant) because, when FAMILIARITY is *LO* or *MED*, then a change from IMAGEABILITY *LO* to *HI* speeds up reaction times, but has the opposite effect when FAMILIARITY is *HI*.

The Tukey test shows that, with a very conservative post-hoc testing approach, there is hardly anything significant in the data. But let us visualize the data. The code file shows you different kinds of plots, interaction plots using lines, a bar plot of means and confidence intervals, dot charts of means, and a (too?) colorful boxplot of the observed medians and their notches as well as means and their confidence intervals. Finally, the last one is an effect plot, which again shows clearly that this interaction is not significant: the lines for the means are nearly parallel.

Now, what about the same analysis with treatment contrasts?

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~IMAGEABILITY*FAMILIARITY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="F")¶
```

Again, everything is the same as above except for the estimates and I explain what they mean in detail in the code file. However, since understanding treatment contrasts will be *very* important for logistic regressions, I want to comment on them here as well. There are two central rules that, once internalized, help you understand all treatment contrast results easily:

(61) Each coefficient/estimate for a predictor *X* (main effect, interaction, or factor level) is the value you must add to the intercept to,
   a. in the case of categorical variables, predict the value for the level of *X* you are looking at;
   b. in the case of numeric variables, predict the value that results from a one-unit change of *X*;

   while, and this is the crucial point, all categorical predictors not

mentioned in *X* are set to their first level (usually the alphabetically first level, but it can also be the one you set first, as in this case), and all numeric predictors not mentioned in *X* are 0.

The second rule is just a special case of (61), namely the intercept:

(62) Therefore, the intercept, where *no* predictor is mentioned, is the predicted value when

a. *all* categorical variables in the model equation are set to their first level;

and/or (!)

b. *all* numerical variables are set to zero (which, if you centered or *z*-standardized them, corresponds to their mean)

Thus,

- the intercept is the predicted RT when both predictors are set to their first level (*LO*);
- the second coefficient is what you add to the intercept to predict the RT for when the predictor mentioned changes to the level mentioned here (i.e., IMAGEABILITY changes from *LO* to *HI*) and when the predictor not mentioned here stays at the level from the intercept (i.e., FAMILIARITY remains *LO*);
- the third coefficient is what you add to the intercept to predict the RT for when the predictor mentioned changes to the level mentioned here (i.e., FAMILIARITY changes from *LO* to *MED*) and when the predictor not mentioned here stays at the level from the intercept (i.e., IMAGEABILITY remains *LO*), similarly for the fourth coefficient;
- the fifth coefficient is for a predictor that is an interaction. Thus, to use it for a prediction, you do not just add this estimate to the intercept, but also the estimates for the main effects that are part of it. Thus, to predict the RT for when IMAGEABILITY is *HI* and FAMILIARITY is *MED*, you add to the intercept the second coefficient (for when IMAGEABILITY is *HI*), the third coefficient (for when FAMILIARITY is *MED*), and this fifth one (for the interaction):

```
> 676.30 + -26.11 + -58.94 + 18.91¶
[1] 610.16
```

Compare that to `preds.hyp[4,]`: the result is the same, and the same logic applies to the sixth coefficient. It is probably obvious by now why inspecting `preds.hyp` and plotting predicted values is easier than ploughing through the table of coefficients, especially since `preds.hyp` is the basis for the plotting, which you have done above.

You could now summarize `model.01` as before: overall model statistics, predictors and their *p*-values, and a plot.

2.5. A linear model with a categorical and a numeric predictor

In the last section, both variables were categorical so all effects were (adjustments to) means. Now we turn to mixed variables: one variable is categorical (FAMILIARITY), one is numeric (FREQUENCY). First, sum contrasts:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY*FREQUENCY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

The model is very significant ($p = 0.001728$), but as before you do not get all *p*-values for all predictors: you can see FREQUENCY is significant, but you do not get one *p*-value for FAMILIARITY and the interaction. Thus:

```
> drop1(model.01, test="F")¶
> Anova(model.01, type="III")¶
```

Both show that the interaction is not significant. On to the estimates:

```
> preds.hyp<-expand.grid(FAMILIARITY=levels(FAMILIARITY),
   FREQUENCY=floor(min(FREQUENCY)):ceiling(max(FREQUENCY)));
   preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-
   predict(model.01, newdata=preds.hyp, interval=
   "confidence"); preds.hyp¶
```

This data frame is not easy to process. (Given the length of this table, I show how to create a version that is easier to process in the code file.) One can see generally that, as FREQUENCY goes up, predicted RTs go down, but really what is needed is a graph. But a question first: What does the interaction represent and, therefore, how does this have to be plotted?

THINK
BREAK

As in Section 5.1.2, the interaction of a categorical and a numeric variable means that the there is not one slope for the effect of the numeric variable in the model but as many slopes as there are levels of that categorical variable. That is, the interaction reflects adjustments to slopes. Hence, we plot a graph that has different regression lines for FREQUENCY for each level of FAMILIARITY; the levels of FAMILIARITY are represented by their first letters. The plot here is quite minimalist (e.g., by not including the original data points), but the code file provides a variety of alternatives; the simplest one to do is, as usual, the effect plot.

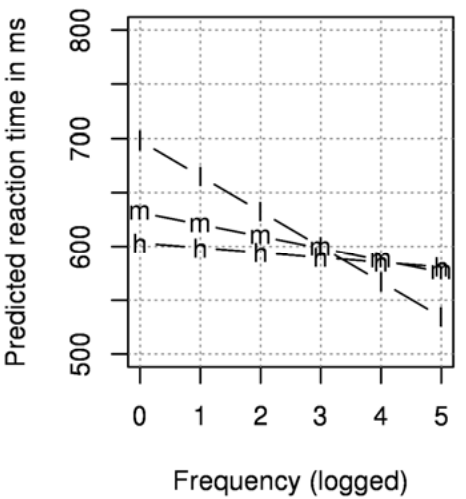

Figure 67. The interaction FAMILIARITY:FREQUENCY in `model.01`

The plot shows the results more efficiently than anything else (esp. when confidence intervals are added to show that interaction is not significant). The model shows that, on the whole, FREQUENCY speeds subjects up but especially when FAMILIARITY is *LO* compared to when it is not. This, together with the *p*-values etc., should be in your summary of the model.

Now again a quick glance at treatment contrasts:

```
> options(contrasts=c("contr.treatment", "contr.poly"))¶
> model.01<-lm(RT~FAMILIARITY*FREQUENCY, data=RTs)¶
```

```
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="F")¶
```

As usual, it's the coefficients that change, and they change in accordance with the rules in (61) and (62):

- the intercept is the predicted RT when all predictors are set to their first level or 0, i.e. when FAMILIARITY is *LO* and FREQUENCY is 0;
- the second coefficient is what you add to the intercept to predict the RT for when the predictor mentioned changes to the level mentioned here (i.e., FAMILIARITY changes from *LO* to *MED*) and when the predictor not mentioned here stays at the level from the intercept (i..e. FREQUENCY remains 0), similarly for the third coefficient;
- the fourth coefficient is what you add to the intercept to predict the RT for when the predictor mentioned increases by one unit (since FREQUENCY is numeric, it changes from 0 to 1) and when the predictor not mentioned here stays at the level from the intercept (i.e., FAMILIARITY remains *LO*);
- the fifth coefficient is for a predictor that is an interaction. Thus, to use it for a prediction, you do not just add this estimate to the intercept, but also the estimates for the main effects that are part of it. Thus, to predict the RT for when FAMILIARITY is *MED*, and FAMILIARITY increases by 1, you add to the intercept the second coefficient (for when FAMILIARITY is *MED*), the fourth coefficient (for when FAMILIARITY increases by 1), and this one (for the interaction):

```
> 697.96 + -66.40 + -32.73 + 21.65¶
[1] 620.48
```

Compare that to `preds.hyp[5,]`; same for the sixth coefficient.

2.6. A linear model with two numeric predictors

Now we are getting serious, enough fun and games. We are going to model REACTTIME as a function of two numeric variables, FREQUENCY and MEANINGFULNESS, and their interaction. This is somewhat tricky because of the interaction. An interaction between two categorical variables reflects adjustments to means, an interaction between a categorical variable and a numeric variable reflects adjustments to slopes – but what is an interaction

between two numeric variables? As you will see, it is that one numeric variable's slope effect changes across the range of the other numeric variable, which also means we will sometimes have to consider three-dimensional plots: one predictor on the *x*-axis, the other predictor on the *y*-axis, the prediction on the *z*-axis.

With only two numeric predictors, we need not distinguish between sum and treatment contrasts so let's get started. (You may also paste the `drop1` and `Anova` lines, but they are unnecessary: every predictor has 1 *df*.)

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~MEANINGFULNESS*FREQUENCY, data=RTs)¶
> summary(model.01)¶
> confint(model.01)¶
```

A just about significant model – although no predictor is significant, which is somewhat rare. We generate `preds.hyp`, which this time is a bit more cumbersome. Since we have two numeric variables, we generate ranges of values for both of them. For FREQUENCY we do this as before, for MEANINGFULNESS I do not just use eight values (an arbitrary choice, it could also be 20) from the attested range, but also 0 and 1 (so I can explain the coefficients).

```
> preds.hyp<-expand.grid(MEANINGFULNESS=c(0:1,
   seq(floor(min(MEANINGFULNESS, na.rm=TRUE)),
   ceiling(max(MEANINGFULNESS, na.rm=TRUE)), length.out=8)),
   FREQUENCY=floor(min(FREQUENCY)):ceiling(max(FREQUENCY)))¶
> preds.hyp[c("PREDICTIONS", "LOWER", "UPPER")]<-predict(
   model.01, newdata=preds.hyp, interval="confidence")¶
> preds.hyp¶
```

In fact, the coefficients mean what they always mean; cf. (61) and (62):

- the intercept is the predicted RT when both MEANINGFULNESS and FREQUENCY is 0;
- the second coefficient is what you add to the intercept to predict the RT for when the predictor mentioned increases by one unit (i.e., when MEANINGFULNESS increases from 0 to 1) and when the predictor not mentioned here stays at the level from the intercept (i.e., FREQUENCY remains 0);
- the third coefficient is what you add to the intercept to predict the RT for when FREQUENCY increases from 0 to 1 and when MEANINGFULNESS stays at the level from the intercept (i.e., remains 0);

– the fourth coefficient is for a predictor that is an interaction. Thus, to use it for a prediction, you do not just add this estimate to the intercept, but also the estimates for the main effects that are part of it. Thus, to predict the RT for when MEANINGFULNESS is 1 and FREQUENCY is 1, you add to the intercept all coefficients.

Now, in actual work you would not have added to the predictions values that are based on MEANINGFULNESS values as far away from the real values, which also affects the plotting. We therefore generate a data frame `preds.hyp.for.plot` with a huge number of predictions, namely all predictions based on all combinations of 100 MEANINGFULNESS and 100 FREQUENCY values, as shown in the code file (note the use of `seq(…, length.out=…)`. and the use of `na.rm=TRUE` to make sure that `min` and `max` don't have problems with the missing data.

Now you have several possibilities. The first two shown in the code involve something I cannot really demonstrate well in a book: The function `plot3d` generates rotatable 3-dimensional plots – you can click onto the plot and move the mouse to turn the coordinate system – and the `col` argument uses the function `grey` (see `?grey`) to make the darkness of the points dependent on the height of the predicted value. Usually, you have quite some turning of the plot to do before you can see what's happening in the data – I do recommend, however, to let the predicted values be on the vertical axis most of the time. (An alternative plot shows that you can use any color scaling you want.)

While this is very useful to interpret the data, you cannot usually publish such graphs. Thus, sometimes you can represent the predicted values not in a third dimension but using color or plotting symbols. The following plot is a scatterplot with MEANINGFULNESS and FREQUENCY on the *x*- and *y*-axis respectively, and the size of the predicted value is represented by the lightness: the lighter the grey, the slower subjects are predicted to be.

On the whole, but especially when MEANINGFULNESS is low, as FREQUENCY increases, predicted RT decreases – see how in the left half of the plot, the grey gets darker as you go up. Also on the whole, but especially when FREQUENCY is low, as MEANINGFULNESS increases, predicted RT decreases – see how in the lower half of the plot, the grey gets darker as you go to the right. However, and this is the slight hint at an interaction (and indicated by the slight bow upwards in the 3-dimensional plot), when both MEANINGFULNESS and FREQUENCY become very high, we do *not* get the fastest RTs:

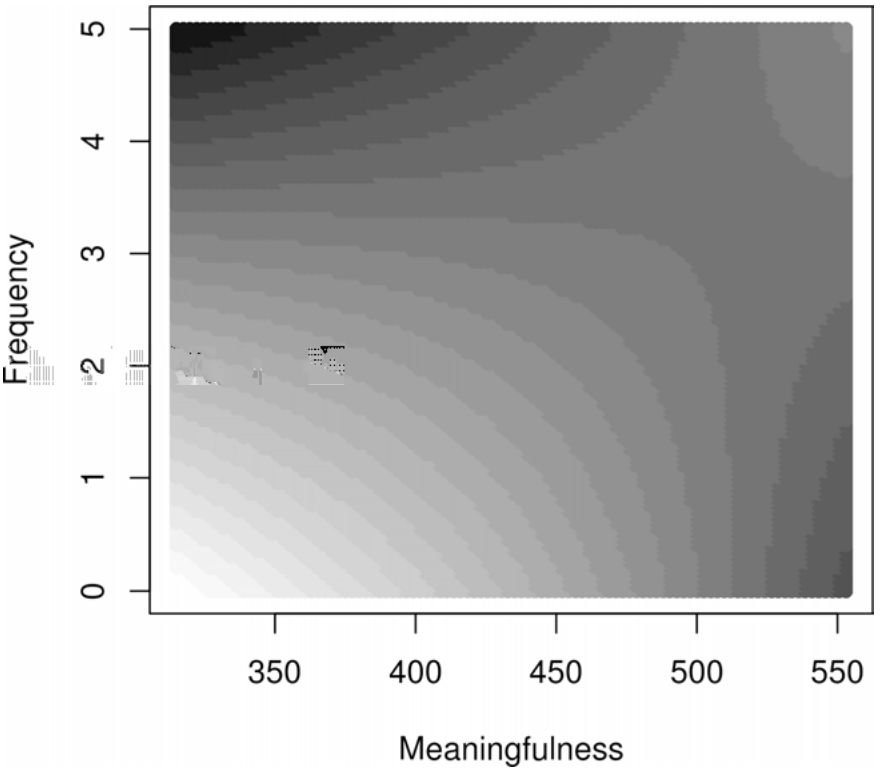

Figure 68. The interaction MEANINGFULNESS:FREQUENCY in `model.01`

In the upper right corner, the points are *not* the darkest. But, `model.01` showed that this bit of an interaction is in fact not significant, which you would also have to say in your results/discussion section.

Other graphical possibilities to play around with are exemplified in the code file including an effects plot. One of these uses numbers as plotting symbols and shows nicely how predictions change in the space spanned by MEANINGFULNESS and FREQUENCY.

2.7. A linear model selection process with multiple predictors

So far, we have ignored two things. First, for expository reasons we have ignored Occam's razor: when a predictor – a main effect or an interaction – was not significant, we left it in the model and plotted it anyway. In this section, we will look at how to do a backwards model selection process.

Second, we have ignored tests of the regression assumptions so we will also talk about this a bit at the end. The maximal model we will explore here involves all the independent variables you have seen so far and including all their interactions up till (and including) 3-way interactions; let me note in passing that this can only be a didactic example since the number of predictors is too high compared to the small number of data points:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> model.01<-lm(RT~(FREQUENCY+FAMILIARITY+IMAGEABILITY+
   MEANINGFULNESS)^3, data=RTs[complete.cases(RTs),])¶
> summary(model.01)¶
```

Note how we define the `data` argument to make sure only complete cases are entered into the process. Also note the syntax to say that we want to include main effects, 2-way, and 3-way interactions: variables are parenthesized and then we say `^3`.

The results show an overall insignicant model with some significant but many insignificant predictors in it. Part of the reason why the overall model is significant is because the large number of (insignificant predictors) increases the degrees of freedom, which makes it harder to get a significant result; note in this connection the huge difference between multiple R^2 and adjusted R^2. Also, we have a problem that is quite common especially with naturalistic data: one cell in our design has only one observation – the combination of FAMILIARITY:*HI* and IMAGEABILITY:*LO* – which leads to NAs in the coefficients, which in turn makes the `Anova` function not work.

```
> Anova(model.01, type="III")¶
```

There are three ways to handle this. The probably best one is to use `drop1`, which, as usual, will test for all predictors that could be omitted at this stage whether their deletion would make the model significantly worse:

```
> drop1(model.01, test="F")¶
```

As you can see, just as discussed in Section 5.1.2.2, `drop1` tests only the highest-order interactions and the one with the highest *p*-value would be the best one to be deleted first: FREQUENCY:FAMILIARITY:IMAGEABILITY.

A second possibility is to add an argument to `Anova`, which provides the same result and conclusion regarding which interaction to delete first:

```
> Anova(model.01, type="III", singular.ok=TRUE)¶
```

The final possibility would be the most laborious one. It involves identifying the four interactions that could be deleted, computing four models each of which differs from `model.01` only by missing one of these interactions – that is, the smaller model is a sub-model of the larger! – and then doing a model comparison to see how much worse the smaller model is. After this is done for all four candidate interactions to be deleted, you delete the one for which the largest non-significant *p*-value was obtained.

The first of these steps, generating a sub-model, is best done with the function `update`. The first argument is the model which you want to change, followed by `~.`, followed by what you want to do, e.g. here subtract a predictor: (I only show the first two updates; you should also explore the help for `update`, which can be used in other useful ways.)

```
> model.02a<-update(model.01, ~. -
   FREQUENCY:FAMILIARITY:IMAGEABILITY)¶
> model.02b<-update(model.01, ~. -
   FREQUENCY:FAMILIARITY:MEANINGFULNESS)¶
```

Then you compare the first model with everything to these sub-models using the function `anova` (small a!): (Again I only show the first two.)

```
> anova(model.01, model.02a)¶
> anova(model.01, model.02b)¶
```

You end up with the interaction to be deleted first. To now delete that interaction you again use `update` and now define `model.02` as `model.01` without FREQUENCY:FAMILIARITY:IMAGEABILITY:

```
> model.02<-update(model.01, ~. -
   FREQUENCY:FAMILIARITY:IMAGEABILITY)¶
```

This process is now repeated as often as needed and as shown in the code file. You of course only need to run one of the alternatives shown there. One comment: `drop1` will sometimes already return *p*-values for the deletion of predictors of a lower degree of interactivity than the one you are currently checking. We will stick to the above and only go to a lower level of interactivity, or to lower-order interactions, if no higher-order interactions is left to delete; cf. the sequence in the code file.

After quite some testing, you arrive at `model.14`, which, following Occam's razor, contains only FAMILIARITY as a predictor – everything else had to be thrown out.

```
> summary(model.14)¶
                                                                [...]
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   619.049      7.145  86.636  < 2e-16 ***
FAMILIARITY1   33.201     11.619   2.858  0.00644 **
FAMILIARITY2   -5.541      8.445  -0.656  0.51512
---                                                             [...]
Multiple R-squared: 0.1711, Adjusted R-squared: 0.1343
F-statistic: 4.645 on 2 and 45 DF,  p-value: 0.01465
```

But we are not done. FAMILIARITY has three levels, but maybe we don't need all of them, something which was above suggested already by `TukeyHSD(aov(…))`. We therefore continue with model comparison – not anymore by testing to discard variables, but now variable levels. Following the logic of Crawley (2007: 563), we create two new factors, each of which conflates two adjacent levels and add them to our data frame (to make sure we test the same number of cases), and then we compute two new models, one with each conflated version of FAMILIARITY, and then we do `anova` model comparisons:

```
> FAMILIARITY.conflat1<-FAMILIARITY.conflat2<-FAMILIARITY¶
> levels(FAMILIARITY.conflat1)<-c("lo", "med-hi", "med-hi")¶
> levels(FAMILIARITY.conflat2)<-c("lo-med", "lo-med", "hi")¶
> RTs<-cbind(RTs, FAMILIARITY.conflat1=FAMILIARITY.conflat1,
   FAMILIARITY.conflat2=FAMILIARITY.conflat2)¶
```

```
> model.15a<-lm(RT~FAMILIARITY.conflat1, data=
   RTs[complete.cases(RTs),])¶
> model.15b<-lm(RT~FAMILIARITY.conflat2, data=
   RTs[complete.cases(RTs),])¶
> anova(model.14, model.15a)¶
> anova(model.14, model.15b)¶
```

The results show that the first conflation – the one that also had the higher *p*-value in the TukeyHSD test – does not make the model significantly worse whereas the second one does. So, now Figure 69 is how the final model can be summarized (see the code and `?plotmath` for how the main heading can feature italics, superscripts, etc.):

Let me at this point briefly interrupt the discussion of this model and return to a more general point. In this case, we only have a significant main effect, and in the sections above we discussed how to plot interactions between two variables. Sometimes, users then raise the question, "ok, but I have a significant interaction of three variables – how do I plot that one?"

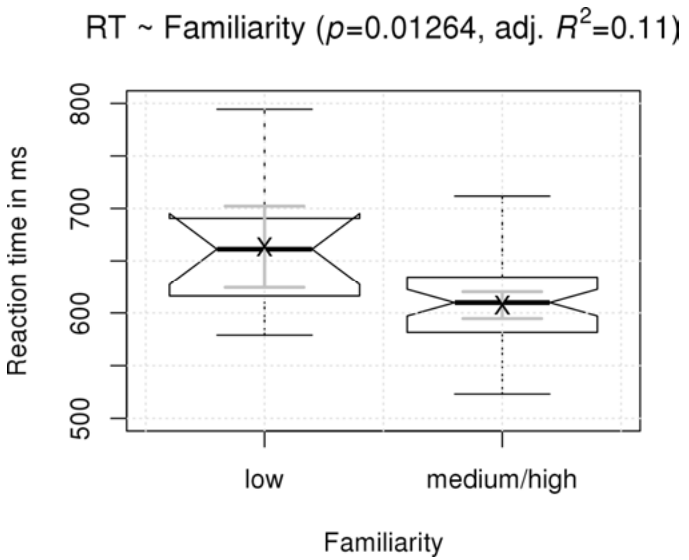

Figure 69. The final only significant effect from `model.15`

The usual answer is that you should plot these on the basis of the above plots. For example, if you have an interaction of three categorical variables X, Y and Z, then you do plots for X:Y as discussed in Section 5.2.4 for each level of Z (and ideally you try out different configurations to see which graph is easiest to interpret). For example, imagine an interaction of two categorical variables X and Y and one numeric variable Z. In that case, you might plot X:Z as discussed in Section 5.2.5 for every level of Y (or Y:Z for every level of X), etc. That is, you just take the plots discussed above and use them as building blocks for higher-order interactions.

A related and very important question is how to get something like `preds.hyp` for a predictor X in a model when X is not the only predictor left in the model. In such scenarios, the approach with `preds.hyp` from above is not ideal when another predictor in the model, say Y, has levels whose frequencies differ wildly, which often happens with observational data. For example, `model.10` in the model selection process involves the following formula:

```
> formula(model.10)¶
RT ~ FREQUENCY + FAMILIARITY + IMAGEABILITY + MEANINGFULNESS
   + IMAGEABILITY:MEANINGFULNESS
```

If you want to extract the predicted values for FAMILIARITY, then you

can use the function `effect` to create a list called, say, `fam`:

```
> fam<-effect("FAMILIARITY", model.10); fam¶
 FAMILIARITY effect
FAMILIARITY
      lo      med       hi
653.7191 613.6080 606.0162
```

While this output is exactly what you would need, getting these numbers out of there (maybe even with confidence intervals) is not as easy as it seems. You have to know that

- the predictor variables we created with `expand.grid` are in `fam$x`;
- the predicted values are now in `fam$fit`;
- the lower bounds of the confidence interval are in `fam$lower`;
- the upper bounds of the confidence interval are in `fam$upper`.

How do you then use this to create something like `preds.hyp` for the interaction? Check out the code file to see how it's done.

Back to `model.15`. The final thing to be done before you explain the model selection process you have done and summarize the results is to check the model assumptions. This can be done in many ways but two practical ones are the following. First, you can inspect some model-diagnostic graphs; second, you can use the function `gvlma` from the package with the same name to get a quick overview.

```
> par(mfrow=c(2, 2))¶
> plot(model.15)¶
> par(mfrow=c(1, 1))¶
```

The two left graphs test the assumptions that the variances of the residuals are constant. Both show the ratio of the fitted/predicted values on the *x*-axis to kinds of residuals on the *y*-axis. Ideally, both graphs would show a scattercloud without much structure; here we have only two fitted values (one for each level of `FAMILIARITY.conflat1`), but no structure such that the dispersion of the values increases or decreases from left to right: here, these graphs look ok.[35] Several words are marked as potential outliers. Also, the plot on the top left shows that the residuals are distributed well around the desired mean of 0.

35. You can also use `ncvTest` from the library `car`: `ncvTest(model.15)¶`, which returns the desired non-significant result.

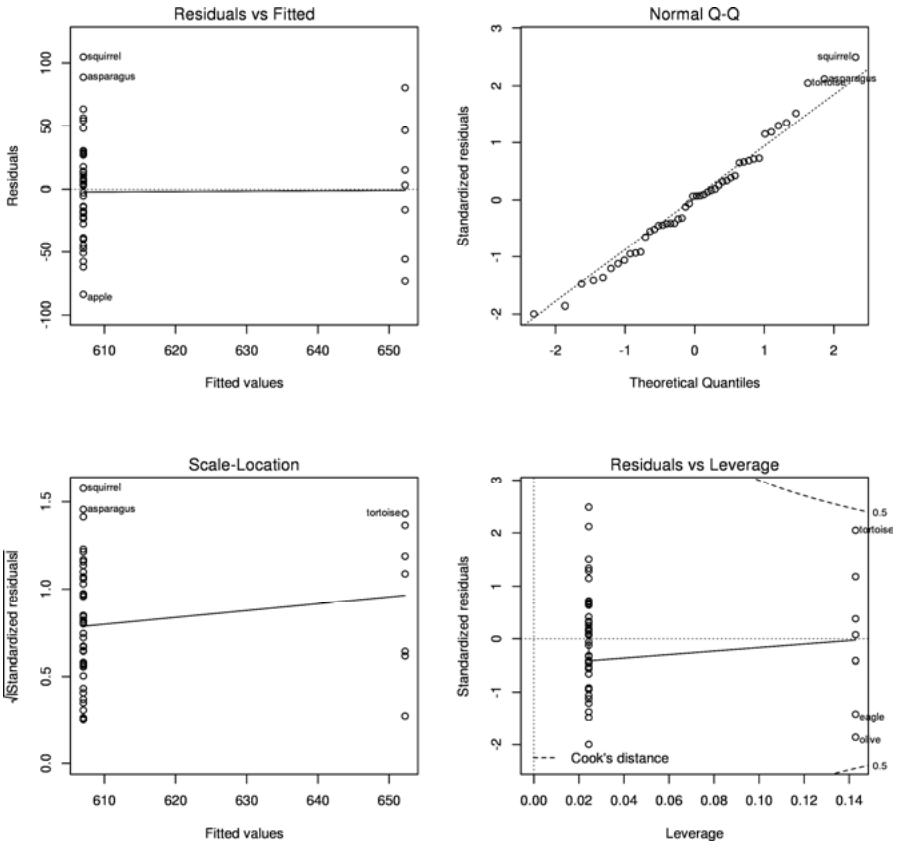

Figure 70. Model diagnostics for `model.15`

The assumption that the residuals are distributed normally also seems met: The points in the top right graph should be rather close to the dashed line, which they are; again, three words are marked as potential outliers. But you can of course also do a Shapiro-Wilk test on the residuals, which also yields the result hoped for.

Finally, the bottom right plot plots the standardized residuals against the so-called leverage. Leverage is a measure of how much a data point may influence a model (because it is far away from the center of the relevant independent variable). As you can see, there are a few words with a larger leverage, and these are all cases of FAMILIARITY:*LO*, which in this toy data set is a much smaller number of data points. Let me briefly also show one example of model-diagnostic plots pointing to violations of the model assumptions. Figure 71 below shows the upper two model plots I once found

when exploring the data of a student who had been advised (by a stats consultant!) to apply an ANOVA-like linear model to her data. In the left panel, you can clearly see how the range of residuals increases from left to right. In the right panel, you can see how strongly the points deviate from the dashed line especially in the upper right part of the coordinate system. Such plots are a clear warning (and the function `gvlma` mentioned above showed that four out of five tested assumptions were violated!). One possible follow-up would be to see whether one can justifiably ignore the outliers indicated; see Fox and Weisberg (2011: Chapter 6) for discussion.

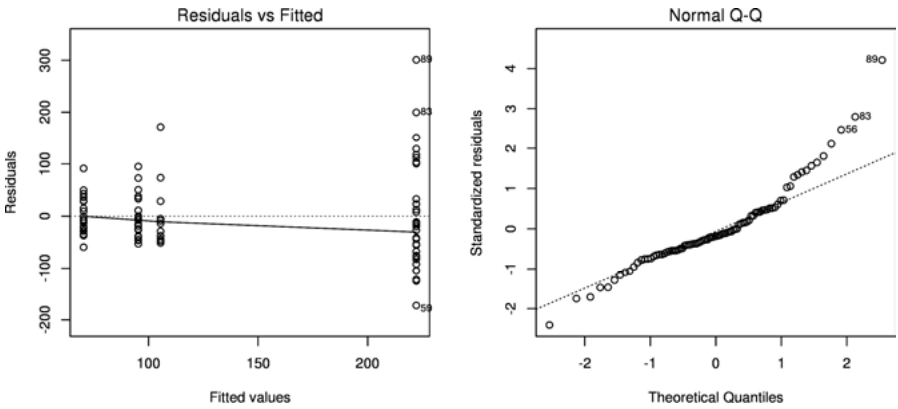

Figure 71. Problematic model diagnostics

Recommendation(s) for further study

- for model selection:
  - the function `step`, to have R perform model selection automatically based on *AIC* (by default) and the function `stepAIC` (from the library `MASS`). Warning: automatic model selection processes can be dangerous: different algorithms can result in very different results
- on model diagnostics:
  - the functions `residualPlots` and `marginalModelPlots` and `vif` from the library `car`, the former two as alternatives to `plot(model)`, the latter to test for collinearity, which is very important to explore; it is the threat posed by highly intercorrelated predictor variables; cf. Faraway (2005: Sections 5.3 and 9.3, Fox and Weisberg 2011: Chapter 6)
  - the functions `influence.measures` and other functions mentioned in this help file (esp. `dfbeta`) to identify leverage points and outliers
- the functions `oneway.test` and `kruskal.test` as alternatives to mono-

factorial ANOVAs
- the libraries `robust`, `MASS`, and `nls` for robust as well as nonlinear regressions; cf. esp. Crawley (2002, 2005) and Faraway (2005, 2006)
- the function `rpart` from the library `rpart`, and the function `ctree` from the library `party`, to compute classification and regression trees as alternatives to (generalized) linear models
- Harrell (2001), Crawley (2002: Ch. 13-15, 17), Faraway (2005: Ch. 14), Crawley (2007: Ch. 10-12, 14, 16), Gelman and Hill (2007: Ch. 3-4), Baayen (2008: 4.4, Ch. 6-7), Johnson (2008: Section 2.4, 3.2, Ch. 4), Zuur et al. (2009: Ch. 2-4, 6-7), Fox and Weisberg (2011), Baguley (2012: Ch. 5, 12-15)

3. Binary logistic regression models

In the last section, we dealt with linear methods, in which the dependent variable is interval-/ratio-scaled and covers a wide range of values. However, in many situations the dependent variable is binary, categorical, or numeric but maybe only ≥ 0 and/or discrete (as for frequencies) or … Since the 'normal' linear model discussed above predicts values between $-\infty$ and $+\infty$, it will predict values that do not make much sense for such dependent variables – what would a predicted value of -3.65 mean when you try to predict frequencies of something? For situations like these, other models are used, some falling under the heading of *generalized linear models*, leading to types of regression such as:

- binary logistic regression for binary dependent variables;
- ordinal logistic regression and multinomial regression for ordinal and categorical dependent variables respectively;
- Poisson/count regression for frequencies as dependent variables.

To be able to apply a linear modeling approach to such data, the dependent variable is transformed with a so-called link function, which transforms the predicted range of values of a linear model ($-\infty$ to $+\infty$) to a range more appropriate for the dependent variable. For binary logistic regression, for example, the inverse logit transformation in (63a) transforms values from the range of $-\infty$ to $+\infty$ to into values ranging from 0 to 1, which can then be interpreted as probabilities of a predicted event. For Poisson regression, the exponential transformation in (64a) transforms values from the range of $-\infty$ to $+\infty$ to into values ranging from 0 to $+\infty$.; the functions in

(63b) and (64b) transform in the opposite direction:

(63) a. inverse logit of x: $\frac{1}{1+e^{x}}$

b logit of x: $\log\frac{x}{1-x}$

(64) a. exponential function of x: e^{x}

b. logarithmic function of x: $\log_{natural} x$

Thus, there is good news and bad news … The bad news is that binary logistic regression is not easy to understand, because of how the link function transforms the dependent variable and because, as you will see, there are three different ways in which one can report results of such a regression, which makes it difficult to understand how textbooks or papers explain methods/results. The good news is that, once you have abstracted away from the link function, everything else is pretty much the same as above, and in this section we can work with R's default treatment contrasts all the time – no need for two types of contrasts.

The data set we will explore involves the question how a main and a subordinate clause in a sentence are ordered. It involves these variables:

- a dependent binary variable, namely ORDER: *MC-SC* vs. *SC-MC* indicating whether or not the main clause precedes the subordinate clause;
- an independent binary variable SUBORDTYPE: *CAUS* vs. *TEMP* indicating whether the subordinate clause is a causal or a temporal one;
- two independent numeric variables LENGTHMC and LENTHSC, representing the number of words of the main and the subordinate clause;
- an independent numeric variable LENGTHDIFF, which represents the difference main clause length minus subordinate clause length; that is, negative values indicate the main clause is shorter;
- a categorical independent variable CONJ, which represents the conjunction used in the subordinate clause. Since these data are from the study involving parallel corpus data, these are the levels: *ALS/WHEN*, *BEVOR/BEFORE*, *NACHDEM/AFTER*, and *WEIL/BECAUSE*;
- an independent binary variable MORETHAN2CL: *NO* vs. *YES* indicating whether or not there is more than just this main and subordinate clause in the sentence. This can be understood as a question of whether the sentence involves more complexity than just these two clauses.

A binary logistic regression involves the following procedure:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a logistic regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - independence of data points and residuals, no overly influential data points, no multicollinearity, and no overdispersion
  - fewer than 95% of the model'2 absolute standardized residuals > 2
  - few if any of the absolute dfbetas of any case and predictor > 1

First, the hypotheses:

H_0: There is no correlation between ORDER and the predictors (independent variables and their interactions): Nagelkerke's $R^2 = 0$.

H_1: There is a correlation between ORDER and the predictors (independent variables and their interactions): Nagelkerke's $R^2 > 0$.

Then you load the data from <_inputfiles/05-3_clauseorders.csv>:

```
> CLAUSE.ORDERS<-read.delim(file=file.choose())¶
> summary(CLAUSE.ORDERS); attach(CLAUSE.ORDERS)¶
```

In this case, no further preparation of the data will be undertaken, which is why the data frame has already been attached. However, we do want to write two helper functions (and define `error.bar` again as above), load a few packages, and make sure we're using treatment contrasts:

```
> logit<-function(x) { log(x/(1-x)) }¶
> ilogit<-function(x) { 1/(1+exp(-x)) }¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

Exploration of the data with cross-tabulations and spineplots of variables against ORDER does not raise any red flags so let's go ahead. In this section, I will show relatively little code/plots in the book so do follow along with the code file!

3.1. A logistic regression with a binary predictor

In this section, we will consider whether ORDER is correlated with SUBORDTYPE. As before, this first section will be longer than the ones that follow to lay the groundwork for the more complex things later.

Just like a linear model with one binary predictor reduces to a simpler test we already know – the *t*-test – so does a logistic regression with a binary predictor relate to a simpler test: the chi-squared test, since we really just have two binary variables (as in Section 4.1.2.2):

```
> orders<-table(SUBORDTYPE, ORDER); orders¶
           ORDER
SUBORDTYPE mc-sc sc-mc
      caus   184    15
      temp    91   113
> chi.orders<-chisq.test(orders, correct=FALSE); chi.orders¶
 Pearson's Chi-squared test
data:  orders
X-squared = 106.4365, df = 1, p-value < 2.2e-16
```

Two brief comments: First, remember the notions of odds and odds ratios from Section 4.1.2.2. Here's how from this table you would compute the odds of *MC*-*SC* first with causal, then with temporal subordinate clauses: Plus, we also said that you can compute an odds ratio from that and that sometimes you will see a logged odds ratio

```
> (184/199) / (15/199)¶
[1] 12.26667
> (91/204) / (113/204)¶
[1] 0.8053097
> 12.26667/0.8053097¶
[1] 15.23224
> log(15.23224)¶
[1] 2.723414
```

Finally, you can of course express the fact that, obviously, causal subordinate clauses prefer to follow the main clause whereas temporal subordinate clauses prefer to precede the main clause with percentages: 92.46% of all causal subordinate clauses, but only 44.61% of the temporal subordinate clauses, follow the main clause. In other words, we have three different but of course related ways to talk about this result – odds, log odds, and percentages/probabilities – something I will come back to in a moment.

The second comment has to do with an alternative to χ^2. Logistic regression does not use a χ^2-value as computed in a χ^2-test but a so-called likeli-

hood ratio test that results in a *G*-value. *G* is also χ^2-distributed and, in a logistic regression involving only one binary/categorical variable, can be computed in a way that is similar to χ^2; cf. (65) and also the code file for a little demonstration showing how similar χ^2 and *G* are.

$$G = 2 \cdot \sum_{i=1}^{n} observed \cdot \log \frac{observed}{expected} \tag{65}$$

```
> 2*sum(orders*log(orders/chi.orders$expected))¶
[1] 116.9747
```

With all this in mind, let us now run a logistic regression. The main function is `glm`, for generalized linear model, and it takes a formula and a data argument as before, but now also an argument that allows R to infer you want to use a link function for binary logistic regression (I am simplifying a bit). Also as before, the coefficients the regression will return are estimates so we immediately request confidence intervals with `confint`:

```
> model.01<-glm(ORDER~SUBORDTYPE, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
                                                              […]
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.2706  -0.3959  -0.3959   1.0870   2.2739

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)     -2.5069     0.2685  -9.336   <2e-16 ***
SUBORDTYPEtemp   2.7234     0.3032   8.982   <2e-16 ***
                                                              […]
    Null deviance: 503.80  on 402  degrees of freedom
Residual deviance: 386.82  on 401  degrees of freedom
AIC: 390.82
                                                              […]
> confint(model.01)¶
                   2.5 %    97.5 %
(Intercept)    -3.076455 -2.016328
SUBORDTYPEtemp  2.156967  3.352559
```

Similar to `lm` output, but also different. For example, you do not get an overall *p*-value. However, you can infer that the model is significant from the fact that the only predictor is significant (and that its confidence interval does not include 0). Also, at the bottom you find the so-called null deviance – informally speaking, the amount of overall variability in the data –

and the residual deviance – informally speaking, the amount of variability left in the data after the predictor has taken care of some of the variability – and the difference between the two is *G*. As mentioned above, *G* is χ^2-distributed with *df* as the difference between the *df*s of the deviances, i.e. 1. Thus, the model's overall *p*-value can be computed as follows:

```
> pchisq(503.80-386.82, 402-401, lower.tail=FALSE)¶
[1] 2.899771e-27
```

Again, we find summary statistics regarding the residuals at the top, and again, before we discuss the estimates, we run code that would help us to get *p*-values for predictors with more than one *df*:

```
> drop1(model.01, test="LR")¶
Single term deletions
Model:
ORDER ~ SUBORDTYPE
            Df Deviance    AIC    LRT  Pr(>Chi)
<none>           386.82 390.82
SUBORDTYPE  1   503.80 505.80 116.97 < 2.2e-16 ***
---                                                    […]
> anova(model.01, glm(ORDER~1, family=binomial), test="LR")¶
Analysis of Deviance Table
Model 1: ORDER ~ SUBORDTYPE
Model 2: ORDER ~ 1
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1       401     386.82
2       402     503.80 -1  -116.97 < 2.2e-16 ***          […]
```

The only change for `drop1` is that we now don't do an *F*-test but the likelihood ratio test (with `LR`). The line with `anova` does the same kind of model comparison: it compares `model.01` against a minimal model where `ORDER` is only regressed onto an overall intercept (`1`) and returns the same likelihood ratio test. We can also use `Anova` again, we just need to switch to sum contrasts just for this one test, and again we get the familiar result:

```
> options(contrasts=c("contr.sum", "contr.poly"))¶
> Anova(model.01, type="III", test.statistic="LR")¶
Analysis of Deviance Table (Type III tests)
Response: ORDER
            LR Chisq Df Pr(>Chisq)
SUBORDTYPE    116.97  1  < 2.2e-16 ***                  […]
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

As before, in the code file I provide some extra code to get *p*-values for predictors/coefficients using `wald.test` and `glht`.

Now, finally, on to the coefficients and one last time we generate `preds.hyp` in the same way as with linear models. However, for generalized linear models, `predict` does unfortunately not return confidence intervals for the predicted values.

```
> preds.hyp<-expand.grid(SUBORDTYPE=levels(SUBORDTYPE));
   preds.hyp["PREDICTIONS"]<-predict(model.01, newdata=
   preds.hyp); preds.hyp¶
  SUBORDTYPE PREDICTIONS
1       caus  -2.5068856
2       temp   0.2165283
```

Now what does that mean? Obviously, this is neither an ordering choice nor 0 vs. 1 choice … To understand what these values mean, you have to (i) recollect the three different ways we talked about the data above: odds and odds ratios, log odds, and probabilities, and (ii) you have to realize that these predicted values are log odds for the predicted ordering, and by default R predicts the second level of the dependent variable, i.e. here *SC*-*MC*. Once you know that, you can use the above to also realize how the three ways to consider these data are related, which is represented in Figure 72.

This graph represents the three perspectives on the results next to each other and it represents the possible numerical ranges of the three ways on the *y*-axes: odds range from 0 to +∞, log odds from -∞ to +∞, and probabilities from 0 to 1. Each of these perspectives expresses preference, dispreference, and lack of effect in different ranges. In numerical odds space, no preference is 1, in log odds space it's 0, and for predicted probabilities it's of course 0.5 (since we have two options). For odds, preferences are reflected by odds greater than 1, by positive log odds, and by predicted probabilies of > 0.5, and the opposites reflect dispreferences.

Now, if the predictions above are log odds, we can transform them to help us recognize what they mean. Let me show `orders` again first.

```
> orders¶
             ORDER
SUBORDTYPE mc-sc sc-mc
      caus   184    15
      temp    91   113
```

If `preds.hyp` contains log odds, anti-logging/exponentiating them should give us odds (cf. again Figure 72 and the code file), and it does.

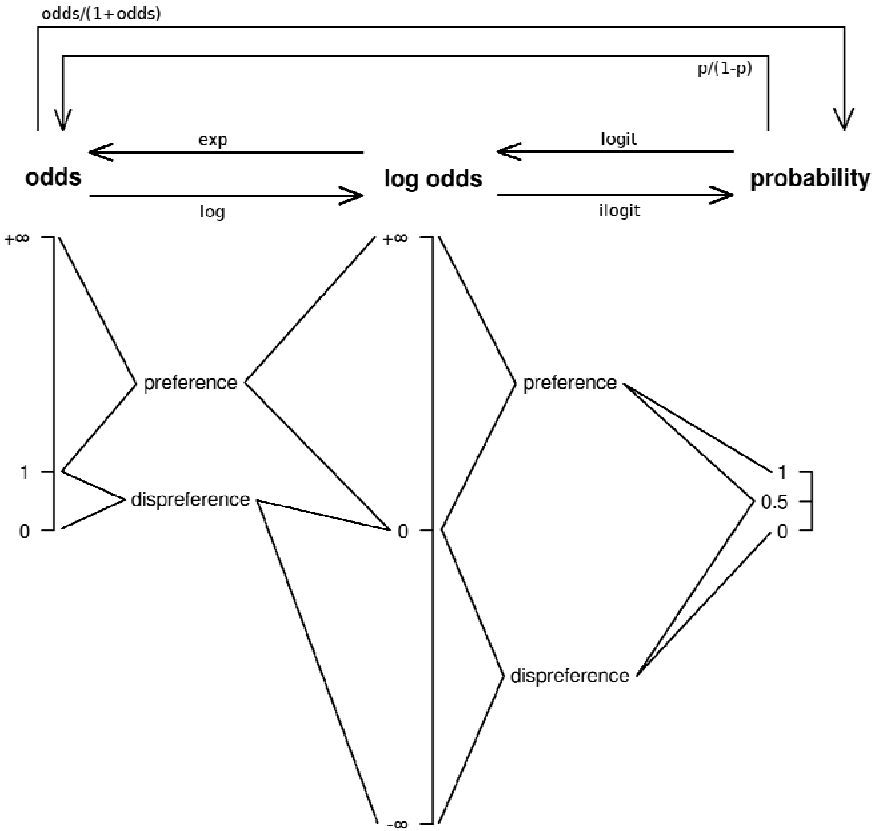

Figure 72. Three ways to look at results from a binary logistic regression

```
> exp(-2.5068856) # odds for SC-MC when SUBORDTYPE=="caus"¶
[1] 0.08152174
> exp(0.2165283) # odds for SC-MC when SUBORDTYPE=="temp"¶
[1] 1.241758
```

This also means, dividing the two gets us an odds ratio, which you also get from anti-logging the coefficient:

```
> exp(-2.5068856)/exp(0.2165283) # 1/odds ratio from above¶
[1] 0.06565025
> exp(0.2165283)/exp(-2.5068856) # odds ratio from above¶
[1] 15.23223
> exp(2.7234)¶
[1] 15.23202
```

Similarly, if `preds.hyp` contains log odds, applying `ilogit` should give us probabilities (cf. again Figure 72 and the code file), and it does:

```
> ilogit(-2.5068856) # prob. of sc-mc when SUBORDTYPE="caus"¶
[1] 0.07537688
> ilogit(0.2165283) # prob. of sc-mc when SUBORDTYPE="temp"¶
[1] 0.5539216
```

Finally, you can also get to these probabilities from the odds using the equation shown in Figure 72 and in (37) on p. 186:

```
> 0.08152174/(1+0.08152174)¶
[1] 0.07537689
> 1.241758/(1+1.241758)¶
[1] 0.5539215
```

A great part of what can be so confusing about logistic regression for beginners is that authors of papers or textbooks can and do use any one of these three perspectives: they are all right, but without something like Figure 72 it's hard to see how these map onto each other. The natural question now is, which of the three scales is best. As usual, people disagree, but I will tell you which one I am using in my own work and also here.

I myself don't like the odds scale on the left. The fact that the numerical space to express preference (of, say *SC-MC*) is from 1 to $+\infty$, but that the corresponding dispreferences are 'squeezed' into the range from 0 to 1 and the multiplicative nature of this scale make me disprefer it strongly. The log odds scale has attractive properties: it is additive and the numerical spaces for preference and dispreference are equally large and symmetric around 0. What I still do not like about the scale is that it is a scale of something as utterly unintuitive as log odds. Thus, I prefer the probability scale. I can think in terms of probabilities, and the numerical spaces for preference and dispreference are equally large and symmetric around 0.5. It may now seem that probabilities do not come with a disadvantage – but they do, which you will learn about in Section 5.3.3 (see p. 306f.), but I still prefer them. Thus, it is the rightmost scale that we will work with and plot here.

Let us get back to the predictions and re-work this example in a simpler way with probabilities and add confidence intervals, too. First, we generate a version of `preds.hyp` as above, but this time we immediately use the more powerful approach of the `effects` package: we generate an object with all the results for the relevant effect (`sot`) and extract all relevant info from it – the levels of the only predictor, the predicted values, and the confidence limits – and apply `ilogit` to the numeric results:

```
> sot<-effect("SUBORDTYPE", model.01)¶
> preds.hyp<-data.frame(sot$x, PREDICTIONS=ilogit(sot$fit),
   LOWER=ilogit(sot$lower), UPPER=ilogit(sot$upper))¶
> preds.hyp¶
  SUBORDTYPE PREDICTIONS     LOWER     UPPER
1       caus  0.07537688 0.0459497 0.1212547
2       temp  0.55392157 0.4851215 0.6207159
```

Again, these are the predicted probabilities of the second level of the dependent variable. Thus, the model predicts a low probability of *SC*-*MC* when the subordinate clause is causal and a much higher one when it is temporal. Since we have two options, it is only natural to make 0.5 the cutoff-point (as in Figure 72) and say when the predicted probability of *SC*-*MC* is < 0.5, then the model predicts *MC*-*SC* – otherwise the model predicts *SC*-*MC*. We can use this to determine how well the model is at predicting the ordering choices. We first generate a vector that contains a predicted probability of *SC*-*MC* for every data point in our data using `fitted`. Then we use `ifelse` to let R decide for each predicted probability which ordering it predicts. And then we tabulate the choices predicted by the model with the actual choices and compute how often the two were the same:[36]

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
       predictions.cat
ORDER   mc-sc sc-mc
  mc-sc   184    91
  sc-mc    15   113
> (184+113)/length(predictions.cat)¶
[1] 0.7369727
```

Is that good? What do you compare that to?

THINK
BREAK

Unlike what you might think, you should not compare it to a chance accuracy of 0.5 (because you have two orderings). Why? Because the two orderings are not equally frequent. The more frequent ordering, *MC*-*SC*, accounts for 68.24% of all data, so just by always guessing that, you al-

36. Harrell (2001:248) cautions against using classification accuracy as a way to measure how good a model is. We will use a better measure in a moment.

ready get much more than 50% right. From that perspective (see the code file for another one), the present result is not great: SUBORDTYPE only improves our accuracy by about 5%.

Let us now also visualize the results. I present two graphs here, nicer versions of which you will see when you run the code in the code file. The left panel of Figure 73 shows a bar plot of the predicted probabilities of *SC-MC*; the right panel shows a line plot of those probabilities.

```
> barplot(preds.hyp$PREDICTIONS, ylim=c(0, 1),
   names.arg=preds.hyp$SUBORDTYPE)¶
> plot(sot, ylim=c(0, 1), rescale.axis=FALSE)¶
```

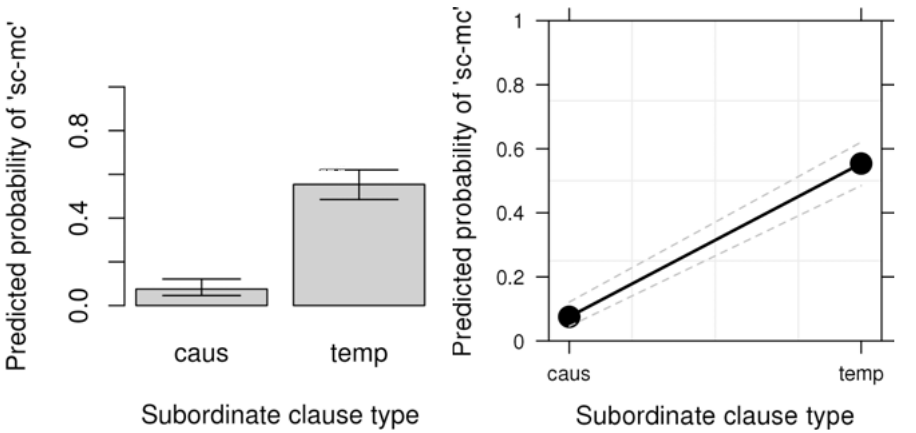

Figure 73. The effects of `model.01`: barplot with observed/predicted probabilities and their 95% confidence-interval bars (left panel); effects plot from the library `effects` (right panel)

Finally, I want to introduce a very useful tool for all sorts of regression modeling, the package `rms` and its function `lrm` (for logistic regression modeling). To use all that `lrm` has to offer, it is useful to first run the first line of code below so that functions from `rms` can access basic information about the ranges of variables etc. The second line uses the function `lrm` to fit the model with the formula from `model.01`. You do not have to specify `family`, but for many follow-up applications (later, when you become more proficient) several other arguments may be provided as shown:

```
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS,
   x=TRUE, y=TRUE, linear.predictors=TRUE, se.fit=TRUE);
   model.01.lrm¶
```

```
                                                                    […]
                      Model Likelihood     Discrimination
                         Ratio Test           Indexes
Obs            403    LR chi2     116.97    R2        0.353
 mc-sc         275    d.f.             1    g         1.365
 sc-mc         128    Pr(> chi2) <0.0001    gr        3.915
max |deriv| 2e-09                          gp        0.240
                                           Brier     0.159

Rank Discrim.
   Indexes
C       0.776
Dxy     0.552
gamma   0.877
tau-a   0.240
Brier   0.159                                                       […]
```

I am not showing all the output but some advantages of `lrm` should be clear: You get the significance test of the model (see the likelihood ratio test), you get an R^2-value (often given as Nagelkerke's R^2 and, as usual, ranging from 0 to 1), and you get a *C*-value, which can be used as an indicator of the classification quality of the model. This value ranges from 0.5 to 1 and values above 0.8 are considered good, which we don't quite achieve here. (Note in passing, $C = 0.5+(D_{xy}/2)$.

To sum up: "A binary logistic regression shows there is a highly significant but weak correlation betwween the type of subordinate clause and the order of main and subordinate clause (G = 116.97; df = 1; $p < 0.001$; Nagelkerke's $R^2 = 0.353$, $C = 0.776$); 73.7% of the orderings are classified correctly (against a chance accuracy of 68.24%). The model predicts that causal subordinate clauses prefer to follow main clauses whereas temporal ones prefer to preceed main clauses. [add a graph and maybe coefficients]"

3.2. A logistic regression with a categorical predictor

As before, we will build up the complexity of the regression models in a stepwise fashion. We therefore now turn to a categorical predictor, CONJ. Again, I will show much less output from now on. The model is significant, with a likelihood ratio value of 123.32 at df =3 (see above). Since FAMILIARITY has more than 1 df, you use `drop1` (or other functions, see the code file) to get one p-value, and FAMILIARITY is highly significant.

```
> model.01<-glm(ORDER~CONJ, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
```

```
> confint(model.01)¶
> drop1(model.01, test="LR")¶
```

To explore what the coefficients reveal, you turn to the predictions. These show that *als/when* and *nachdem/after* prefer *SC-MC*, whereas *bevor/before* and *weil/because* prefer *MC-SC*. I will not explain the meaning of the intercept and all coefficients in detail here but you will find all these explanations in the code file and should read them carefully! The logic and everything else is the same as explained above with (61) and (62), just that the coefficients now represent differences between the intercept – the first level of FAMILIARITY – and the other levels on the log odds scale.

```
> conj<-effect("CONJ", model.01)¶
> preds.hyp<-data.frame(conj$x, PREDICTIONS=ilogit(conj$fit),
   LOWER=ilogit(conj$lower), UPPER=ilogit(conj$upper))¶
> preds.hyp¶
           CONJ PREDICTIONS     LOWER     UPPER
1      als/when  0.60215054 0.4997995 0.6962850
2  bevor/before  0.39130435 0.2623180 0.5375019
3 nachdem/after  0.60000000 0.4773236 0.7112984
4  weil/because  0.07537688 0.0459497 0.1212547
```

Then we see how well the model classifies the orderings. We get an improvement over chance, but 76.18% does not seem like a huge step ahead.

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
> (212+95)/length(predictions.cat)¶
```

Finally, we represent the data visually as before and generate a model with `lrm` to get R^2 and C, which are 0.369 and 0.798 respectively. With these plots and summary statistics, we can now summarize the result of our regression model in the same way as above on p. 304. Note that, given the overlap of the temporal conjunctions, one should strictly speaking also test whether the fine resolution of three temporal conjunctions is warranted …

```
> barplot(preds.hyp$PREDICTIONS, ylim=c(0, 1),
   names.arg=preds.hyp$CONJ)¶
> plot(allEffects(model.01), ask=FALSE, ylim=c(0, 1),
   rescale.axis=FALSE)¶
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS);
   model.01.lrm¶
```

3.3. A logistic regression with a numeric predictor

As the final monofactorial logistic regression, we will now turn to a numeric predictor, LENGTH_DIFF.

```
> model.01<-glm(ORDER~LENGTH_DIFF, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
```

For the sake of consistency you also run `drop1` (and maybe `Anova`/`anova`) although LENGTH_DIFF has 1 *df* so it's not really necessary.

```
> drop1(model.01, test="LR")¶
```

Note there is a slight difference between the *p*-values from the `summary` output on the one hand and `drop1`, `anova`, and `Anova` on ther other hand; according to Fox and Weisberg (2011:239), the likelihood ratio test you get with `drop1` etc. may be more reliable.

The model is significant but the effect really seems quite weak. To understand the coefficients, we create the predictions, but this time around we have to discuss this in more detail, since it is here that a slight disadvantage of the probability perspective on logistic regression results manifests itself. In one sense at least, things are as before: the intercept still represents the probability of the predicted ordering when the independent variable is at its first level or, as here, 0, which you can see in `preds.hyp` and compare that to `ilogit(-0.77673)¶`:

```
> lendiff<-effect("LENGTH_DIFF", model.01,
   xlevels=list(LENGTH_DIFF=-max(abs(range(LENGTH_DIFF))):
   max(abs(range(LENGTH_DIFF))))); lendiff¶
> preds.hyp<-data.frame(lendiff$x, PREDICTIONS=
   ilogit(lendiff$fit), LOWER=ilogit(lendiff$lower),
   UPPER=ilogit(lendiff$upper)); preds.hyp¶
```

Similarly, the coefficient of LENGTH_DIFF still represents the change of the probability of the predicted order, *SC*-*MC*, for a unit change of LENGTH_DIFF. However, this change is linear/constant only on the log odds scale – once we check it on the probability scale, you can see that a change of 1 of LENGTH_DIFF does not bring about the same difference in probabilities: when you increase LENGTH_DIFF by 1

- from -20 to -19, this results in the predicted probability of *SC-MC* growing by 0.006019;
- from -10 to -9, this results in the predicted probability of *SC-MC* growing by 0.0078747;
- from 0 to 1, this results in the predicted probability of *SC-MC* growing by 0.0096109.

This is because the inverse logit transformation is not linear (in probability space). But let us now see how well this model predicts the orderings:

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
> (272+2)/length(predictions.cat)¶
```

In some sense, the performance is abysmal: it is worse than chance even though the direction of the effect makes sense – you should make sure you recognize that it amounts to 'short before long'? Also, you can see from the table that the model hardly ever predicts *SC-MC* – only five times. To visualize the effect of LENGTH_DIFF and explore this bad performance, let us plot the predicted probabilities against length differences from `preds.hyp` (cf. Figure 74).

```
> plot(preds.hyp$LENGTH_DIFF, preds.hyp$PREDICTIONS,
   xlim=c(-35, 35), ylim=c(0, 1))¶
```

As you can see, in probability space you do not get a straight regression line but a curve. Thus, the change of LENGTH_DIFF by one word has different effects depending on where it happens and this is the disadvantage of the probability scale I alluded to earlier. However, given how we can nicely plot such curves in R, this is a disadvantage I am happy to live with (compared to those of the odds or log odds scales).

Figure 74 also helps understand the bad classification accuracy. The horizontal line at the cut-off point of $y = 0.5$ only applied to very few points (see the rugs). One way to try to force the regression to make somewhat more diverse predictions is to choose a cut-off point other than 0.5, and one possibility is to use the median of all predicted probabilities (0.3150244; cf. Hilbe 2009: Section 7.2.2 for discussion).

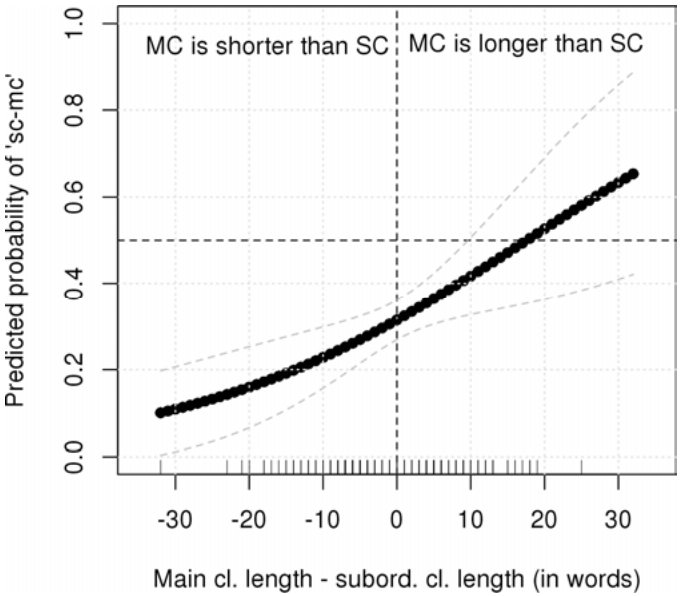

Figure 74. The effects of `model.01`: scatterplot with predicted probabilities and their 95% confidence-interval band

If you do that here, you do get more balanced frequencies of categorical predictions, but the accuracy decreases even further. The code file shows how, when you choose 0.42 as a cut-off point, you get a slightly more balanced frequency of predictions and still about 68% right; for now, we will use this value and I leave the topic of ROC curves and how they help identifying cut-off points for your future exploration Note, however, that if you do not choose the 'default' cut-off point of 0.5 for the categorical predictions, you should mention which one you chose and why. Finally, we do the regression again with `lrm` to get the really small R^2 (0.026) and C (0.603), and then we can summarize our results as above.

```
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS);
   model.01.lrm¶
```

3.4. A logistic regression with two categorical predictors

We now move on to the first logistic regression with more than one predictor: we will explore whether CONJ and MORETHAN2CL and their interac-

tion affect the clause orders. Since one of the predictors has more than 1 *df*, we fit the model and immediately add `drop1` and `Anova`:

```
> model.01<-glm(ORDER~CONJ*MORETHAN2CL, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="LR")¶
> options(contrasts=c("contr.sum", "contr.poly"))¶
> Anova(model.01, type="III", test.statistic="LR")¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

You can see that `drop1` now only returns the *p*-value for the non-significant interaction, which, in a normal model selection process, we would now omit. We leave it in here to explore how to understand and visualize the interaction – with `Anova`, however, we also get all other *p*-values; only CONJ seems significant.

On to the predictions using code I only show in the code file because there I have more space to explain both the code and the meanings of the coefficients, which as usual follow the rules in (61) and (62)). If you look at `preds.hyp`, *weil/because* sticks out, but the output also shows why the interaction is not significant: the confidence intervals are huge and, on the whole, the conjunctions seem to pattern alike across both levels of MORETHAN2CL. How good are these predictions?

```
> predictions.num<-fitted(model.01)¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(ORDER, predictions.cat)¶
> (216+93)/length(predictions.cat)¶
```

We get nearly 77.7% right, which is at least above chance again. Finally, we represent the data visually as before and generate a model with `lrm` for the overall model test (likelihood ratio χ^2=132.06, *df*=7, $p < 0.001$), R^2 (0.392), and *C* (0.82).

```
> dd<-datadist(CLAUSE.ORDERS); options(datadist="dd")¶
> model.01.lrm<-lrm(formula(model.01), data=CLAUSE.ORDERS);
   model.01.lrm¶
```

Since the data now involve an interaction, the code can become a bit more involved (at least when you do not use the functions from the effects package) and I show it only in the code file. The non-significant interaction is reflected by the large overlap of the confidence intervals and the similar

(differences of) values of the predicted probabilities, which is, with everything else, what you would discuss in your results section.

3.5. A logistic regression with a categorical and a numeric predictor

As in Section 5.2.5, we now turn to a case with an interaction between a categorical and a numeric predictor, which means again that the interaction coefficients will reflect adjustments to the slope of the numeric predictor.

```
> model.01<-glm(ORDER~CONJ*LENGTH_DIFF, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
> drop1(model.01, test="LR")¶
> options(contrasts=c("contr.sum", "contr.poly"))¶
> Anova(model.01, type="III", test.statistic="LR")¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

This time, both main effects are significant, and the interaction is only just about not significant. Therefore, it would be warranted to at least look at the interaction (as we will for didactic purposes anyway).

We generate the predictions the usual way and it is again prudent to maybe also generate a slightly flatter table that can be inspected before we turn to plots, as shown in the code file:

```
> intact<-effect("CONJ:LENGTH_DIFF", model.01,
   xlevels=list(LENGTH_DIFF=seq(-32, 32, length.out=9)))¶
> preds.hyp<-data.frame(intact$x, PREDICTIONS=
   ilogit(intact$fit), LOWER=ilogit(intact$lower),
   UPPER=ilogit(intact$upper)); preds.hyp¶
```

Especially the flatter representation of `preds.hyp.2` is now easier to read. You can see for each conjunction how the predicted probability of *SC-MC* changes as LENGTH_DIFF changes. A plot will make this even more obvious in a moment. Again, read the code file in detail to understand how the coefficients result in these predictions! How good are the predictions?

```
> predictions.num<-predict(model.01, type="response")¶
> predictions.cat<-ifelse(predictions.num>=0.5,
   "sc-mc", "mc-sc")¶
> table(predictions.cat, ORDER)¶
> (228+83)/length(predictions.cat)¶
```

Approximately 77.2% are classified correctly, and a cut-off point of 0.57 results in a slightly better value (of about 78.2%). But now, what do the results amount to? In this case, where the *p*-values of all effects are at least < 0.07, we will do some nice plotting, which will be partly based on `preds.hyp.2`, as it has separate columns for the conjunctions. Figure 75 is what we want to create (in the lower panel, the letters are the first letters of the German conjunctions). Remember, though: the interaction is significant so that is what you should focus on – not the main effects! The code file shows how exactly this is done; I know it's a lot of lines, but you should invest the time to see what every line is doing because, once you get it, you will be able to use this logic for many examples in your own work.

The first main effect is somewhat familiar from above. The three temporal conjunctions prefer *SC*-*MC* whereas *weil/because* strongly prefers *MC*-*SC*. The second main effect is also familiar: short-before-long. Now the interaction is interesting. Three observations can be made:

- *als/when* and *nachdem/after* exhibit similar average preferences for *SC*-*MC* and both react to LENGTH_DIFF in a way that is (more) compatible with short-before-long than the average;
- *bevor/before* not only has less of a preference for *SC*-*MC* but also the opposite tendency compared to the other two temporal conjunctions: when the main clause becomes longer, it want to precede the subordinate clause;
- *weil/because* subordinate clauses are pretty much completely immune to considerations of length: the want to come second no matter what.

(The code file also contains code for the interaction with confidence intervals, but those make the plot harder to read, defeating its purpose; the effects plot is more useful in that regard.) All the above, together with the *p*-values for the predictors, the *p*-value for the overall model (likelihood ratio χ^2=135.21, *df*=7, $p < 0.001$), R^2 (0.399) and *C* (0.818) from the corresponding model with `lrm` (see below) would be part of your results section.

3.6. A logistic regression with two numeric predictors

The final logistic regression, as before with two numeric predictors. We are going to check whether the lengths of the two clauses affect the ordering choice. First a question: how is this different from checking whether LENGTH_DIFF is significant?

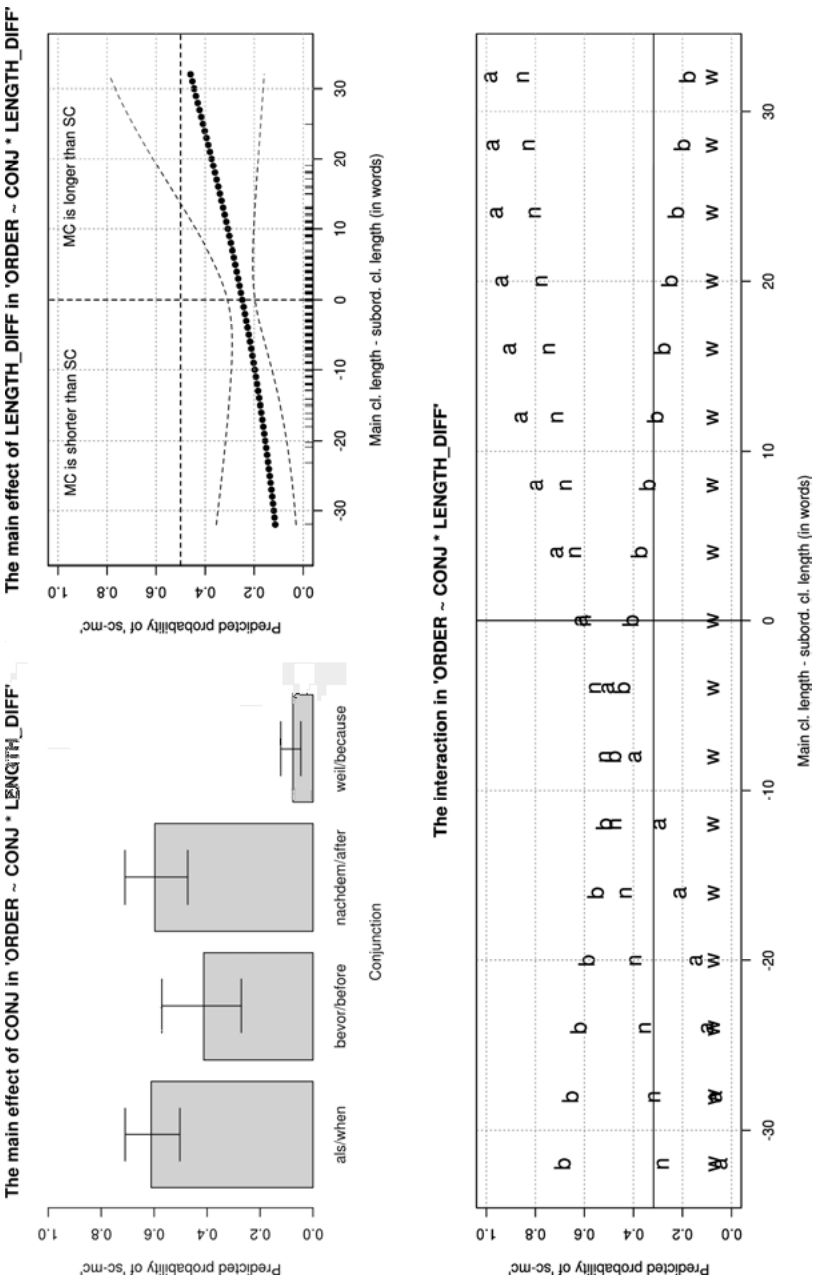

Figure 75. All effects in `model.01` (remember, however: if the interaction is significant, your discussion should focus on it, not the main effects)

THINK BREAK

It is different because it takes into account where a particular length difference may be observed: If LENGTH_DIFF is 1, that value does not reveal whether it arises from the main and the subordinate clause containing 10 and 9 or 20 and 19 words respectively. As usual, we fit the model; since both variables are numeric, `drop1` and `Anova` are not really necessary:

```
> model.01<-glm(ORDER~LEN_MC*LEN_SC, data=CLAUSE.ORDERS,
   family=binomial)¶
> summary(model.01)¶
> confint(model.01)¶
```

This is all as non-significant as it gets; it is only for didactic reasons that we proceed with the predictions, and with two numeric predictors spanning a wide range of values, both `preds.hyp` and the flatter `preds.hyp.2` (see the code file) are not easy to process. (Don't forget to read the code file's explanation of the coefficients.) Next, we check the cross-classification table, immediately using the median of the predicted probabilities as the cut-off point:

```
> predictions.num<-predict(model.01, type="response")¶
> predictions.cat<-ifelse(predictions.num>=
   median(predictions.num), "sc-mc", "mc-sc")¶
> table(predictions.cat, ORDER)¶
> (151+82)/length(predictions.cat)¶
```

The accuracy is quite low, even beyond chance, which is not surprising given the *p*-values of the predictors. But what do the effects look like? As before, you have basically two possibilities. First, you can again generate a 3-dimensional rotatable plot (with `plot3d`), and the code file shows an, I think, nice version where different colors and different letters (the first letter of the first clause) represent which ordering is predicted for which combination of lengths. The fact that the interaction is insignificant is reflected by the fact that the letters nearly form a straight plane in 3-dimensional space. The more publication-ready version is shown in Figure 76.

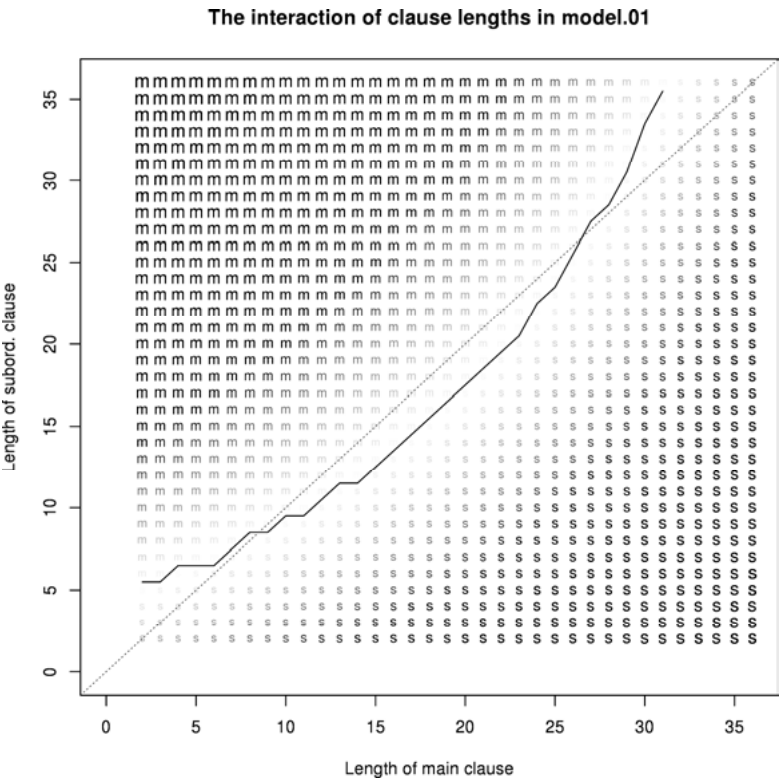

Figure 76. The interaction LENGTHMC:LENTHSC in `model.01`

LENGTHMC and LENGTHSC are on the *x*- and *y*-axis respectively. When the categorical prediction is *MC*-*SC*, I plot an *m* otherwise an *s*. The larger and the darker the letter, the more 'certain' is the model about the prediction in the sense that the prediction is based on a probability further away from the cut-off point. The straight grey line is the main diagonal where both clauses are equally long, and the curved black line indicates for every main clause length where the prediction flips to the other clause order, which is why there the letters are so light. While the interaction is not significant, we see short-before-long again: when the main clause is short (the left of the plot), then as the subordinate clause becomes longer, it wants to go in the hind position, and the same the other way round for subordinate clauses. The effects plots show the same kind of result, but by splitting up the subordinate clause lengths into ten different ranges and then plotting regression lines and their confidence intervals. While easier to generate in terms of code, I find that graph less easy to interpret than Figure 76.

Finally, you would generate the model with `lrm` and sum it all up; here of course, all predictors are weak and non-significant and R^2 as well as *C* are really low (0.028 and 0.606 respectively).

We have now completed the overview of the different logistic regression models. Again, as mentioned above on p. 263, you would of course not have done all these models separately, but a model selection process like the one in Section 5.2.7. One of the exercises for this chapter will have you do this for the present data, and you will find that the results shed some more light on the unexpected behavior of *bevor/before* in Section 5.3.5.

What remains do be covered, however, is again how to test whether the assumptions of the regression model are met. Above I mentioned three different criteria (but see Fox and Weisberg (2011: Ch. 6)). You already know about inspecting residuals but overdispersion is new. It requires that you look at the ratio of your model's residual deviance and its residual *df*s, which should not be much larger than 1. In this case, it is $^{495.58}/_{399}$=1.242. Several references just say that, if you get a value that is much larger than 1, e.g. > 2, then you would run the `glm` analysis again with the argument `family=quasibinomial` and take it from there. Baayen (2008: 199) uses as an approximation a chi-square test of the residual deviance at the residual *df*:

```
> pchisq(495.58, 399, lower.tail=FALSE)¶
[1] 0.0006880771
```

Thus, if this was a real analysis with a significant result, one might want to follow that advice. The other criteria I mentioned were concerned with the absolute values of the standardized residuals of the model and of the dfbetas. The former are a type of corrected residuals (see Fox and Weisberg 2011: 286f.) and Field, Miles, and Field (2012: Section 8.6.7.3) suggest that no more than 5% should be > 2 or < -2. This is easy to test:

```
> prop.table(table(abs(rstandard(model.01))>2))¶
     FALSE       TRUE
0.99751861 0.00248139
```

In this case not even 1% is > 2 or < -2. Similarly straightforward is the test of the dfbetas, which reflect how much a regression coefficient changes when each case is removed from the data. Again, testing this in R is simple:

```
> summary(dfbetas<-abs(dfbeta(model.01)))¶
```

The output (not shown here) indicates that in fact no absolute dfbeta is greater than 0.1 so this criterion also poses no problems to our model. Checking diagnostics carefully is an important component of model checking and R in general, and the library `car` in particular, has many useful functions for this purpose.

Recommendation(s) for further study

- just like in Section 5.2, it can also help interpreting the regression coefficients when the input variables are centered
- the function `hoslem.test` from the library `ResourceSelection` for the Hosmer-Lemeshow test (see Hilbe 2009: Section 7.2) (you want to see a non-significant result)
- Field, Miles, and Field (2012: Section 8.8.2) on the assumption of the linearity of the logit
- Pampel (2000), Jaccard (2001), Crawley 2005: Ch. 16), Crawley (2007: Ch. 17), Faraway (2006: Ch. 2, 6) , Zuur, Ieno, and Smith (2007: Section 6.1), Gelman and Hill (2007: Ch. 5), Baayen (2008: Section 6.3), Baguley (2012: Ch. 17)

4. Other regression models

The above two types of regression models have been the most widely-used ones in linguistics. In this section, I will introduce a variety of other regression models that are not that widespread yet, but which are bound to become used more in the near future: *ordinal logistic regression* (where the dependent variable is ordinal), *multinomial regression* (where the dependent variable is categorical with 3+ levels), and *Poisson regression* (where the dependent variable consists of frequencies). The logic of the exposition will be as above, but – for reasons of space – much abbreviated. Specifically, after a short introduction to each section and its data, I will only discuss one example for each regression in the book, namely the case of two independent variables, one categorical and one numeric. However, the code file will discuss six regression models for each, just like before, so that you get a nice homogeneous treatment of all models. I therefore recommend that you load the data, read the chapter in the book, and follow along with the fifth of the six examples in the code file, and then explore the other examples based on the code file as well.

4.1. An ordinal logistic regression with a categorical and a numeric predictor

The example we will explore to approach ordinal logistic regression is concerned with which of a set of independent variables allows us to predict which of three different end-of-term exams or assignments foreign-language learners of English will choose. It involves these variables:

- a dependent ordinal variable, namely ASSIGNMENT: *ORALEXAM* vs. *LABREPORT* vs. *THESIS*; crucially, these are ordered in ascending order of difficulty (based on a previous study);
- an independent binary variable SEX: *FEMALE* vs. *MALE* indicating the sex of the student whose choice has been recorded;
- a categorical independent variable REGION, which represents the geographic region where the student comes from: *CENTRAL-EUROPEAN*, *HISPANIC*, and *MIDDLE-EASTERN*;
- an independent numeric variable WORKHOURS, representing the numbers of hours/month the students claimed to have invested into the class;
- an independent numeric variable MISTAKES, which represents the numbers of mistakes the students made in their last assignment for this class.

Here are the steps of an ordinal logistic regression that we will follow:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a logistic regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - the usual suspects: independence of data points and residuals, no overly influential data points, no multicollinearity
  - the dependent variable "behaves in an ordinal fashion with respect to each predictor" (Harrell 2001:332)

First, the hypotheses:

H_0: There is no correlation between ASSIGNMENT and the predictors

(independent variables and their interactions): $R^2 = 0$.

H_1: There is a correlation between ASSIGNMENT and the predictors (independent variables and their interactions): $R^2 > 0$.

Then you load the data from <_inputfiles/05-4-1_assignments.csv> as well as the library `rms`, whose function `lrm` we will use here:

```
> rm(list=ls(all=TRUE)); library(rms)¶
> ASSIGNS<-read.delim(file=file.choose()); str(ASSIGNS)¶
```

If you inspect the summary provided by `str`, you will see that the levels of the factor `ASSIGNMENT` are not in the right order, and that that factor is not even an ordered factor, which means R treats it as a categorical variable (as all factors in this book so far), not as the desired ordinal variable. Thus, we change this (check `str` again), and then we can `attach` and, since we will use the `lrm` function again, create the required `datadist` object:

```
> ASSIGNS$ASSIGNMENT<-factor(ASSIGNS$ASSIGNMENT, ordered=
   TRUE, levels=levels(ASSIGNS$ASSIGNMENT)[c(2,1,3)])¶
> str(ASSIGNS); attach(ASSIGNS)¶
> ddist<-datadist(ASSIGNS); options(datadist="ddist")¶
```

As mentioned before, I will now skip the first four models discussed in the code file and go directly to the fifth one, where we explore the joint influence of REGION and WORKHOURS on ASSIGNMENT:

```
> model.01<-lrm(ASSIGNMENT ~ REGION*WORKHOURS, data=ASSIGNS,
   x=TRUE, y=TRUE, linear.predictors=TRUE, se.fit=TRUE)¶
> model.01¶
> anova.rms(model.01)¶
```

The model is highly significant (Likelihood ratio χ^2=493.09, df=5, p < 0.001) and shows there is a very strong correlation: R^2=0.908, C=0.938. The `anova.rms` output is a bit different. Rather than giving you a p-value for each main effect and each interaction (as `Anova` from the libray `car` did), you get two p-values for what each main effect does alone together with what it does in the interaction, and you get a p-value for the interaction. Since the interaction is nearly significant, we will focus on that. But what is its nature? The coefficients are now quite different from what we have seen before: there is more than one intercept. I explain the meanings of each coefficient in detail in the code file, but the easiest way to understand the results is again via predicted probabilities, which we will generate

using the same logic but slightly different code (the `effect` function does not work with `lrm` objects but you can sometimes use `polr` from the package MASS). Here are the first two lines:

```
> preds.hyp<-expand.grid(REGION=levels(REGION),
   WORKHOURS=c(0, 1, floor(min(WORKHOURS)):
   ceiling(max(WORKHOURS))))¶
> preds.hyp<-data.frame(preds.hyp, predict(model.01,
   newdata=preds.hyp, type="fitted.ind"))¶
```

This generates a data frame `preds.hyp` again, which contains for each combination of `REGION` and a large number of values of `WORKHOURS` the predicted probability of each kind of assignment. For example, when the student is from the Hispanic region and puts in 26 workhours, he is strongly predicted to choose the lab report:

```
> preds.hyp[preds.hyp$REGION=="hispanic" & preds.hyp$
   WORKHOURS==26,]¶
```

But we want it even nicer: we do not just want the predicted probabilities, but immediately also for each row what the categorical prediction is. As you can see, the `predict` function combined the name of the dependent variable, `ASSIGNMENT`, with the predicted levels by inserting a period between them. We do not want to see that so we use the following:

```
> preds.hyp<-data.frame(preds.hyp, ASSIGNMENT.pred=
   sub("^.*?\\.", "", names(preds.hyp)[-(1:2)][
   max.col(preds.hyp[,-(1:2)])]))¶
> preds.hyp[38:42,]¶
```

The function `sub` takes three arguments: what to look for (and the argument `".*\\."` means 'characters up to and including a period'), what to replace it with (and `""` means 'nothing', i.e., 'delete'), and where to do all this (in the three column names of `preds.hyp` that are not the first two). And then, these levels are subset with the vector of numbers that results from R checking for each row where the maximal predicted probability is (always excluding the first two columns of `preds.hyp`, which contain the independent variables!). Verify this by looking at these five lines of output.

We now first remove the first rows of `preds.hyp` because these were only included to explain the coefficients but were unrepresentative of the real values of `WORKHOURS`. Then, we check the classification accuracy:

```
> preds.hyp<-preds.hyp[-(1:6),]¶
> predictions.num<-predict(model.01, type="fitted.ind")¶
> predictions.cat<-sub("^.*?[\\.=]", "", colnames(
   predictions.num)[max.col(predictions.num)])¶
> table(predictions.cat, ASSIGNMENT)¶
```

As you can see, we achieve a good accuracy, nearly 85%, which is highly significantly better than the chance level of 33% (since the three assignments are equally frequent). Then we plot the predicted probabilities, which, given the multitude of results these types of regressions yield, becomes a bit more involved. One way to represent these results is shown in Figure 77. There is one panel for each region, the workhours are on each *x*-axis, the predicted probabilities on each *y*-axis, and the three assignments are represented by lines and their first letters. On the whole, there is a very strong effect of WORKHOURS: students who self-reported lower workhours are strongly predicted to choose the easiest exam/assignment, the oral exam. Those who report an intermediate number of workhours are strongly predicted to use the intermediately difficult exam/assignment, the lab report, and those who report the largest numbers of workhours are predicted to go with the thesis. The nearly significant interaction, however, indicates that this behavior is not completely uniform across the three regions: For example, the Hispanic students choose the more difficult exams/assignments with smaller numbers of workhours than the Middle Easterners. The Central Europeans stick more to the oral exams even if they work a number of hours where the other students have already begun to prefer the lab report, and only the most industrious Middle Easterners choose the thesis. (See the code file for other graphs.)

Let us finally check some assumptions of this type of regression: The first five plots represent the residuals and those are mostly quite close to 0, as required. The ordinality assumption looks a bit more problematic, though so this requires some more attention, which is beyond the scope of this book; see the recommendations for further study.

```
> par(mfrow=c(2, 4))¶
> residuals(model.01, type="score.binary", pl=TRUE)¶
> plot.xmean.ordinaly(ASSIGNMENT ~ REGION*WORKHOURS)¶
> par(mfrow=c(1, 1))¶
```

Leaving this issue aside for now, you would now be able to summarize the regression results numerically (Likelihood ratio χ^2, df, p, R^2, C) and discuss the graph and its implications along the lines discussed above.

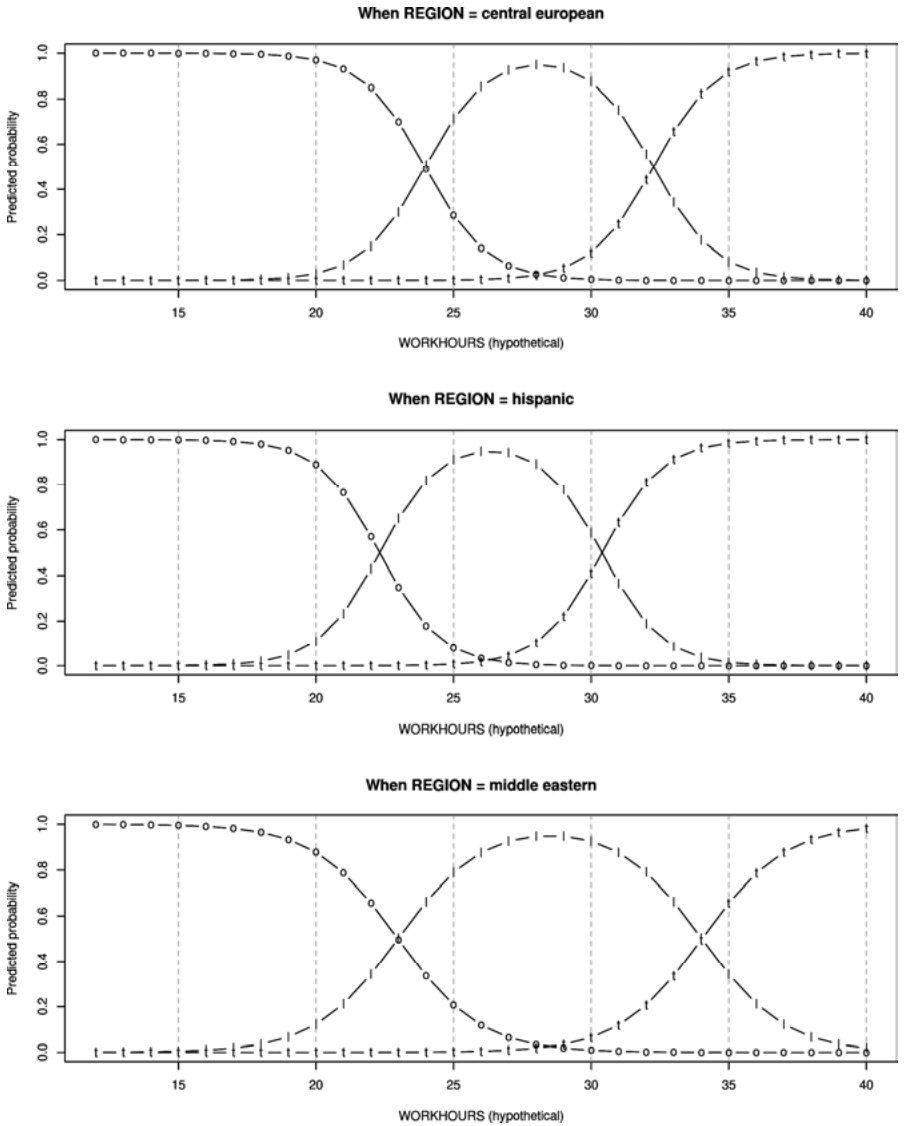

Figure 77. The interaction REGION:WORKHOURS in `model.01`

Recommendation(s) for further study

- the function `polr` from the library `MASS`, for ordinal logistic regressions
- Harrell (2001: Ch. 13-14), Baayen (2008: Section 6.3.2), Hilbe (2009: Ch. 10), Agresti (2010), Fox and Weisberg (2011: Section 5.9)

4.2. A multinomial regression with a categorical and a numeric predictor

After having discussed ordinal logistic regression, we now turn to multinomial regression. For the sake of simplicity, we will use the same data set and just *not* consider ASSIGNMENT an ordinal variable (and hence an ordered factor) but a categorical variable and hence a 'regular' unordered factor. This is the procedure we will follow:

Procedure
- Formulating the hypotheses
- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a multinomial regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - the usual suspects: independence of data points and residuals, no overly influential data points, no multicollinearity
  - independence of irrelevant alternatives, a non-significant Hasuman-McFadden test (which I will not discuss, see the references below)

Given that we're using the same data set, the hypotheses stay the same, too, plus you can load the file, change the levels of ASSIGNMENT as above, (without changing it to an ordered factor though), and we load a number of libraries. Then we fit a multinomial regression model as follows:

```
> model.01<-multinom(ASSIGNMENT ~ REGION*WORKHOURS,
   data=ASSIGNS)¶
> summary(model.01, Wald=TRUE)¶
> mlogit.display(model.01)¶
> confint(model.01)¶
```

The output of summary is a bit overwhelming because we get again get multiple intercepts and coefficients for all but the first level of the dependent variable. These represent in a somewhat complicated way the differences between the first level of the dependent variable and each of the others; in a way, multinomial regressions are series of binary logistic regressions. We also get Wald statistics, which are, as usual, the coefficients divided by their standard errors.

Let us check the significance of the predictors. We can unfortunately

not use `drop1`, but we can do something that is pretty much equivalent to it: an anova comparison of `model.01` to a model without the interaction, plus we can use `Anova` in the by now familiar way. Both reveal that the interaction is not significant at all: $p > 0.9$.

```
> anova(model.01, multinom(ASSIGNMENT ~ REGION+WORKHOURS))¶
> options(contrasts=c("contr.sum", "contr.poly"))¶
>    Anova(model.01, type="III")¶
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

Now what do the coefficients mean? I am nearly tempted to say, "you don't want to know …" The explanations of the coefficients are even more evidence why trying to understand the results in terms of coefficients is often not the best/most intuitive strategy. You will find detailed explanations of them in the code file; suffice it to say here that, when you exponentiate them, you get ratios between different predicted probabilities. One visual representation we might use is the type exemplified by Figure 77 above and the code file shows you how you can generate that graph as well as two others. On the whole, the results are comparable to those of Figure 77: low numbers of work hours lead to oral exams, intermediate ones lead to lab reports, and high ones are more associated with theses, and these preferences are, with some small differences, obtained for all regions.

To determine the classification accuracy, we could proceed the usual way, or we can take take things to the next level. Again we use function `model.statistics` from Antti Arppe's nice package `polytomous`:

```
> model.statistics(ASSIGNMENT, predictions.cat,
   predictions.num)¶
```

This provides an immensely useful set of summary statistics: Log-likelihood statistics and deviances for our `model.01` (-329.5837 and 143.4688) and for a model consisting of just the intercept (-71.73441 and 143.4688), the classification accuracy (0.8733), and, as in the excurses before, Nagelkerke R^2 (0.9233), everything one would want to know …

Recommendation(s) for further study

- Gries (2009: Section 5.1) on (hierarchical) configural frequency analysis (and the script `hcfa` with which it can be computed interactively) and Field, Miles, and Field (2012: Sections 18.7-18.12) on loglinear analysis; also see the functions `loglin` and the function `loglm` (from the library `MASS`) to compute loglinear analyses

- the function `hmftest` from the library `mlogit` to compute the Hausman-McFadden test
- Agresti (2002: Ch. 7), Faraway (2006: Ch. 5), Fox and Weisberg (2011: Section 5.7), Field, Miles, and Field (2012: Section 8.9)

4.3. A Poisson regression with a categorical and a numeric predictor

In this section, I will discuss another type of generalized linear model, namely Poisson regression, which is used to model counts/frequencies. As discussed above on p. 294, just like binary logistic regression this approach also requires a link function – this time the exponential function – to make sure that a linear-model type of approach can be applied to a dependent variable that is never negative. As in the last two sections, I will only discuss one regression with a categorical and a numeric predictor here in the book and encourage you to then explore the other five examples in the code file. The example I will use to explain Poisson regression is concerned with factors that lead to in-/decreased numbers of disfluencies in conversations of bilingual and/or highly advanced non-native speakers and involves the following variables:

- a dependent variable DISFLUENCY, which represents the numbers of disfluencies 300 speakers each produced in 20 minutes of conversation;
- an independent binary variable SEX: *FEMALE* vs. SEX: *MALE*, the speaker's sex;
- a categorical independent variable MOVEDWHEN, which indicates when the speaker moved to the U.S.A.: as an *ADULT*, during *HIGH SCHOOL*, or during *PRIMARY SCHOOL*;
- an independent numeric variable REALITYTV, representing the numbers of hours/month the speakers self-reports to watch reality TV shows;
- an independent numeric variable SOCIALNETWORK, which represents the numbers of hours/week the speakers self-reports to spend time on social networks.

Here are the steps of a Poisson regression that we will follow:

Procedure

- Formulating the hypotheses

- Loading the data, preparing them for modeling, and exploring them
- Computing, selecting, and interpreting a Poisson regression model
  - obtaining *p*-values for all predictors and for the model as a whole
  - interpreting the regression coefficients/estimates on the basis of (i) predicted values and (ii) plots of predicted probabilities
- Testing the main assumption(s) of the test:
  - the usual suspects: independence of data points and residuals, no overly influential data points, no multicollinearity
  - the model does not suffer from overdispersion

First, the hypotheses, then we load some libraries (see the code file) and also the data (from <_inputfiles/05-4-3_disfluencies.csv>).

H_0: There is no correlation between DISFLUENCY and the predictors (independent variables and their interactions): $R^2 = 0$.

H_1: There is a correlation between DISFLUENCY and the predictors (independent variables and their interactions): $R^2 > 0$.

```
> DISFL<-read.delim(file=file.choose())¶
> str(DISFL); attach(DISFL)¶
```

The model we will discuss here tests the hypothesis that the frequency of disfluencies is correlated with the point of time when the speaker moved to the U.S. and the amount of time spent on social networks:

```
> summary(model.01<-glm(FREQDISFL ~ MOVEDWHEN*SOCNETWORK,
   data=DISFL, family=poisson))¶
```

The output of this model already indicates a first problem: overdispersion. The ratio of the residual deviance (3532.6) and the residual degrees of freedom (294) is much much larger than one and significant (`pchisq(3532.6, 294, lower.tail=FALSE)`¶), which is why we fit the model again with `family=quasipoisson`, which corrects the predictors' standard errors and, thus, the *p*-values, and we compute what has been proposed as an R^2:

```
> summary(model.01<-glm(FREQDISFL ~ MOVEDWHEN*SOCNETWORK,
   data=DISFL, family=quasipoisson))¶
> 1-(model.01$deviance/model.01$null.deviance)¶
[1] 0.2558909
```

Since we're using `glm`, much of the code for logistic regressions also applies here. For example, `drop1` and `Anova` get us *p*-values for predictors. Obviously, the interaction is not significant at all, so we would normally update the model by deleting it, and obviously MOVEDWHEN does not seem to play a role whereas SOCIALNETWORK does.

```
> drop1(model.01, test="LR")¶
Single term deletions
Model:
FREQDISFL ~ MOVEDWHEN * SOCNETWORK
                     Df Deviance scaled dev. Pr(>Chi)
<none>                    3532.6
MOVEDWHEN:SOCNETWORK  2   3538.0     0.46628    0.792

> options(contrasts=c("contr.sum", "contr.poly"))¶
>    Anova(model.01, type="III", test="LR")¶
Analysis of Deviance Table (Type III tests)
Response: FREQDISFL
                     LR Chisq Df Pr(>Chisq)
MOVEDWHEN              2.7684  2     0.2505
SOCNETWORK            19.3469  1   1.09e-05 ***
MOVEDWHEN:SOCNETWORK   0.4663  2     0.7920
                                                [...]
> options(contrasts=c("contr.treatment", "contr.poly"))¶
```

For expository purposes only, we continue with the interaction. Above, we used the function `effect` to obtain predicted probabilities – here, we're using it to obtain predicted frequencies and we can really re-use a lot of what we know about using `effect` from before. The only real difference is that, above we applied `ilogit` to `effect`'s output, because the binary logistic regression uses `logit` as a link function – since the Poisson regression uses `log` as a link function, we now apply `exp`. In the code file, I again explain the meanings of the coefficients and how they give rise to the predicted frequencies in much detail. We therefore proceed to the plot. Figure 78 plots DISFLUENCY against the SOCIALNETWORK and then adds three regression lines, one for each level of MOVEDWHEN. (I omitted the confidence bands here, which clutter up the graph unless one can use colors.)

It is plain to see why the interaction is not significant. The positive correlation between DISFLUENCY and SOCIALNETWORK is the same for each level of MOVEDWHEN. That positive correlation as a main effect is significant, but then the differences between the different levels of MOVEDWHEN – the intercepts – also do not reach standard levels of significance. Thus, since here we do not remove the interaction (again, just for expository reasons), we could wrap up the results: "On the whole, there is a highly significant (L.R. χ^2=1214.8; df = 5; $p < 0.001$) [see page 298 on how to compute

this] but not particularly strong correlation (R^2= 0.26). This correlation is due to the fact that the number of hours spent on social networks is significantly positively correlated with the numbers of disfluencies produced (L.R. χ^2=19.35; df = 1; $p < 0.001$) whereas the age of moving to the U.S.A. is not ($p > 0.25$), and neither is their interaction ($p > 0.79$)."

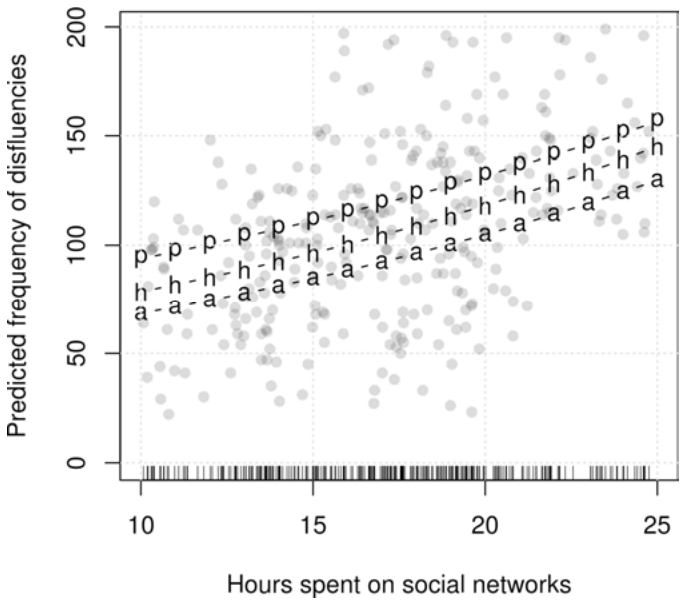

Figure 78. The interaction MOVEDWHEN:SOCIALNETWORK in `model.01`

Recommendation(s) for further study

- Gries (2009: Section 5.1) on (hierarchical) configural frequency analysis (and a script to compute this interactively) and Field, Miles, and Field (2012: Sections 18.7-18.12) on loglinear analysis; also see `loglin` and `loglm` (from the package `MASS`) for loglinear analyses
- Agresti (2002: Ch. 4, 8-9), Faraway (2006: Ch. 3-4), Zuur, Ieno, and Smith (2007: Section 6.1), Zuur et al. (2009: Ch. 8-9, 11), Hilbe (2009: Ch. 11), Fox and Weisberg (2011: Section 5.5-5.6)

5. Repeated measurements: a primer

The final section in this part on regression modeling is devoted to a type of scenario that differs from all previous ones. All models discussed so far

shared one and the same assumption, that the data points (and their residuals) are independent of each other. For instance, in the section on linear regressions, the average reaction time to each word was considered independent of the average reaction time of any of the other words. This scenario, while frequent, is not the only possible one – as you know from page 159 above, groups/samples can be dependent, which means that data points *are* related to each other. The most common ways in which data points are related to each other involve the following scenarios:

- in experimental settings, you obtain more than one response per subject (i.e., you do *repeated measurements* on each subject), which means that the characteristics of any one subject affect more than one data point;
- in experimental settings, you obtain more than one response per, say, a lexical item which you test in some stimulus, which means that the characteristics of any one lexical item affect more than one data point;
- in corpus data, you obtain more than one data point per speaker (often approximated by corpus file), which means …, you get the picture.

If your data involve such related data points but you ignore that in your statistical analysis, you run several risks. First, you run the risk of what is called "losing power", which means you may stick to H_0 although H_1 is true in the population (what is called a *type II error*, a *type I error* is to accept H_1 although H_0 is true in the popluation). Second, you risk obtaining inaccurate results because your statistical analysis doesn't take all the known structure in the data into consideration and will return – in the context of regression modeling – coefficients that are not as precise as they should be.

In this section, I will talk about methods that are used in such cases. However, this section will only be very brief because, while the methods that are used in such cases are quite important and powerful, they also involve considerable complexity and require much more space than I can devote to them here. (See below for some excellent references for follow-up study, in particular Girden (1992), which inspired some of the discussion here, and Field, Miles, and Field (2012).) Also, while the overall logic of repeated measurements applies to many different kinds and configurations of independent and dependent variables, I will only discuss cases that could be considered repeated-measures ANOVAs, i.e. cases in which the dependent variable is interval-/ratio-scaled (i.e., not categorical) and in which the independent variables involved are treated as categorical.

5.1. One independent variable nested into subjects/items

By way of introduction, I will begin my discussion here with a brief example of three different ways in which the simplest possible dependent-samples type of data can be analyzed. In Section 4.3.2.2, we dealt with such a case when we explored the question whether translations of 16 texts were longer than the originals. That scenario involved dependent samples because one could connect every original to its translation and we, therefore, computed a *t*-test for dependent samples. Let us clear memory, load the package `ez`, reload those data (now from <_inputfiles/05-5-1_textlengths.csv>) and then revisit this scenario, here for expository reasons as a two-tailed hypothesis (that the mean lengths from originals and translations differ). Also, before we attach the data frame, we convert the column `TEXT`, which simply numbers the texts, to a factor: this variable is really only categorical since the numbers do not do anything but identify which text a length belongs to – the sizes of the numbers do not matter.

```
> Texts<-read.delim(file.choose()¶
> Texts$TEXT<-factor(Texts$TEXT)¶
> str(Texts); attach(Texts)¶
```

We already know from above that the differences between the originals' and the translations' lengths are normally distributed so we immediately compute the *t*-test for dependent samples (again, here a two-tailed one) and obtain the familiar *t*- and *df*-values as well as a now only marginally significant *p*-value.

```
> t.test(LENGTH~TEXTSOURCE, paired=TRUE)¶
```

Above, we saw that a linear model with one binary predictor is essentially equivalent to a *t*-test for independent samples (recall p. 266f.). It may therefore not be a big surprise that a repeated-measures ANOVA with one binary predictor is essentially equivalent to a *t*-test for dependent samples. The two ANOVAs differ, however, in how the variability in the data is divvied up in the analysis. An independent-measures linear model with one binary or categorical predictor divides the variability in the data up into variability that is attributed to the levels of the independent variable and variability that is attributed to random variation (random noise, residual variability, or error). The effect of the independent variable is then assessed by comparing the two amounts of variability, and the more variability the independent variable accounts for compared to the residual variability, the

more likely it is the independent variable's effect will be significant.

In a repeated-measures ANOVA, the variability is divided up differently. First, there is variability between typically different subjects or here, different texts. But then there is also variability within different subjects (or here, different texts), and a part of that variability is due to the independent variable (here, TEXTSOURCE: *ORIGINAL* vs. TEXTSOURCE: *TRANSLATION*) and the remainder is random error / residual variation. Since in repeated-measures ANOVAs the effect of the independent variable is nested within subjects or, here, texts, we therefore compare the amount of within-subject/text variability that is attributed to the independent variable not to the overall remaining variability, but to the remaining amount of within-subject/text variability, and again the more within-subject variability is accounted for by the independent variable compared to residual within-subject variability, the more likely it is the result will be significant. And this is why dependent-samples / repeated-measurements studies can be more precise: the effects of independent variables are compared to a smaller amount of residual (within-subject/text) variability.

How do we do this in R? We use the function `aov` (for analysis of variance) and tell it (i) that we want a model in which `LENGTH` is modeled as a function of `TEXTSOURCE` (`LENGTH ~ TEXTSOURCE`, no surprises here) and (ii) what the relevant source of error/residual variability (`ERROR(…)`) is by stating that the independent variable `TEXTSOURCE` is nested into, i.e. repeated within, each element of `TEXT` (`TEXT/TEXTSOURCE`):

```
> model.01.aov<-aov(LENGTH ~ TEXTSOURCE +
   Error(TEXT/TEXTSOURCE))¶
> summary(model.01.aov)¶

Error: TEXT
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 15 210479   14032

Error: TEXT:TEXTSOURCE
           Df Sum Sq Mean Sq F value Pr(>F)
TEXTSOURCE  1  51040   51040   3.717  0.073 .
Residuals  15 205991   13733
```

As discussed above, the output divides the overall variability into that between subjects/texts (i.e., the upper part labeled `Errror: TEXT`) and the one within the subjects/texts, which in turn is either due to the independent variable `TEXTSOURCE` (mean square: 51,040) or random/residual noise (mean square: 13733). The *F*-value is then the ratio of the two mean squares at levels of the independent variable minus 1 and subjects/texts

minus 1 degrees of freedom. As you can see, this result is then identical to the *t*-test: The *F*-value is t^2, the *F*-value's residual *df* are the *t*-tests *df*, the *p*-values are identical, and, obviously, so is the conclusion you would write up: With a two-tailed hypothesis, the means (`model.tables( model.01.aov, "means")¶`) do not differ from each other significantly.

A very attractive alternative way to conduct a repeated-measures ANOVA involves the very useful function `ezANOVA` from the library `ez`. The first argument (`data`) is the data frame containing the data, the second (`dv`) specifies the dependent variable, the third (`wid`) specifies the subjects/text identifier, and the fourth (`within`) defines the independent variable nested within the identifier. You get an ANOVA table with the same *F*-value, its two *df*s, the same *p*-value, and a measure of effect size in the column labeled *ges*. (Plus, explore the code with `ezPlot` and look at `?ezStats¶`.)

```
> ezANOVA(data=Texts, dv=.(LENGTH), wid=.(TEXT),
   within=.(TEXTSOURCE))¶
```

5.2. Two independent variables nested into subjects/items

How do we extend the above approach to more complex data such as cases where two variables are nested into a subject or an item? Consider a hypothetical case where five subjects are asked to provide as many synonyms as they can to eight stimuli (different for each subject, so there is no repetition of items), which result from crossing words with positive or negative connotations (a variable called MEANING) and words from four different parts of speech (a variable called POS). Let's assume we wanted to know whether the numbers of synonyms subjects named in 30 seconds differed as a function of these independent variables (and for the sake of simplicity we are treating these frequencies as interval-/ratio-scaled data). We load the data from <_inputfiles/05-5-2_synonyms.csv>:

```
> Syns<-read.delim(file.choose())¶
> str(Syns); attach(Syns)¶
```

In this case, no monofactorial test is available for comparison so we immediately do the repeated-measures ANOVA. The logic is actually not different from above: we want to study the effects of both independent variables and their interaction but both these variables are nested into `SUBJECT`. Thus, we either use `aov` …

```
> model.01.aov<-aov(SYNONYMS ~ MEANING*POS +
   Error(SUBJECT/(MEANING*POS)))
> summary(model.01.aov)¶
```

… or ezANOVA:

```
> ezANOVA(data=Syns, dv=.(SYNONYMS), wid=.(SUBJECT),
   within=.(MEANING, POS))¶
```

As before, both return the same results: The only effect reaching standard levels of significance is POS, and the output of `model.tables` shows that nouns and verbs resulted in high numbers of synonyms, whereas adjectives and adverbs only yielded medium and lower numbers respectively. The output of ezANOVA also returns a test for sphericity, a very important assumption of repeated-measures ANOVAs (see the recommendations for further study). In this case, all the *p*-values are > 0.05 so sphericity is not violated and we can rely on the results of our *F*-tests.

5.3. Two independent variables, one between, one within subjects/items

One final example, in which I show how to handle cases where you have two independent variables, but only one of them is nested into subjects – the other varies between subjects. Imagine you had 10 non-native speaker subjects, each of whom participated in four proficiency tests, or tasks: an oral exam, an in-class grammar test, an essay written in class, and an essay written at home. This is the variable nested into the subjects. However, you also suspect that the sexes of the speakers play a role and, guess what, those are not nested into subjects … This is the between-subjects variable. Let's load the data from <_inputfiles/05-5-3_mistakes.csv>.

```
> Mistakes<-read.delim(file.choose())¶
> str(Mistakes); attach(Mistakes)¶
```

It should be clear what to do: for aov, you specify the formula with all independent variables and tell it that only TASK is nested into SUBJECTS.

```
> model.01.aov<-aov(MISTAKES ~ SEX*TASK +
   Error(SUBJECT/TASK))¶
> summary(model.01.aov)¶
```

For ezANOVA, you use the argument `between` to tell the function that the

independent variables SEX does not vary within, but between subjects:

```
> ezANOVA(data=Mistakes, dv=.(MISTAKES), wid=.(SUBJECT),
   within=.(TASK), between=.(SEX))¶
```

Unfortunately, while we get significant results for both TASK and its interaction with SEX – explore the means with `model.tables` again – this time around the sphericity tests are cause for alarm. `ezANOVA` suggests two corrections for violations of sphericity, both of which still return significant values for TASK and TASK:SEX, but this is beyond the scope of this book, see the recommendations for further study below and the next section.

5.4. Mixed-effects / multi-level models

The above has already indicated that repeated-measures ANOVAs are not always as straightforward to use as the above may have made you expect. First, repeated-measures ANOVAs as discussed above only involve categorical independent variables, but you may often have interval-/ratio-scaled variables and may not want to factorize them (given the loss of information and power that may come with that, see Baayen 2010). Second, many variables you may wish to include are not *fixed effects* (i.e., variables whose levels in the study cover all possible levels in the population) but are *random effects* (i.e., variables whose levels in the study do not cover all possible levels in the population, such as SUBJECT, ITEM, …, see Gelman and Hill 2007: 245f.). Third, repeated-measures ANOVA requires a balanced design and may therefore be problematic with missing data in experiments and unbalanced observational data. Finally, violations of sphericity are not always easy and uncontroversial to address; (see Baguley 2012: Section 18.2.2 for more discussion).

A strategy to handle data with dependent/related data points and random effects that is currently very hot in linguistics is the use of mixed-effects models, or multi-level models. With much simplification, these are regression models that can handle fixed and random effects as well as repeated measurements, unbalanced data, and hierarchical/nested data. They do this by simultaneously modeling different sources of variability by, for example, instead of simply fitting one regression line over many subjects through a point cloud in a coordinate system, they allow the analyst to model the dependent variable with a different regression line for each subject or item, where the different regression lines may have, say, subject-

specific or item-specific different intercepts (called *random intercepts*, because they are modeled as a normally-distributed random variable) and/or slopes (called *random slopes*). For reasons of space and others to be discussed below, I will not discuss these highly complex models here in detail, but I want to give one or two brief examples. For the first of these, I will return to the *t*-test for dependent samples again:

```
> rm(list=ls(all=TRUE)); library(effects); library(nlme)¶
> Texts<-read.delim(file.choose()¶
> Texts$TEXT<-factor(Texts$TEXT)¶
> str(Texts); attach(Texts)¶
```

The package `nlme` (as well as the newer package `lme4`) allows you to fit a large variety of mixed-effects models. This is one way of applying these to the *t*-test data. The function for linear mixed effects is `lme`, and here it takes two arguments: First, the argument `fixed`, which defines the fixed-effects structure of the model, and our only fixed-effect independent variable is `TEXTSOURCE`. Second, the argument `random` describes the random-effects structure of the model, and the notation means we want the intercept (`1`) to be able to vary by `TEXT` (`|TEXT`), which is just another way of capturing text-specific variability as we did in the repeated-measures ANOVA.

```
> model.01.lme<-lme(fixed = LENGTH~TEXTSOURCE, random= ~
   1|TEXT)¶
> summary(model.01.lme)¶
```

The output we get contains a lot of information but we will only focus on the random-effects and the fixed-effects output. The former (in the section "Random effects") contains an estimate of the variability of the 16 random intercepts for the 16 texts, namely a standard deviation of 12.23172. The latter (in the section "Fixed effects: …") contains the familiar kind of table of coefficients, standard errors, *t*-values, and *p*-values. The *t*-value (1.92787), its *df* (15), and the *p*-value (0.073) should look very familiar, since they correspond to the above results for the same data. And you can even create the familiar kind of effects plot for this result because the function `effect` does accept `lme` models as input:

```
> plot(effect("TEXTSOURCE", model.01.lme))¶
```

Some other applications of repeated-measures ANOVAs can be explored similarly. For example, the above data on the mistakes can be studied with this function call:

```
> summary(model.01.lme<-lme(fixed = MISTAKES ~ SEX*TASK,
   random= ~ 1|SUBJECT))¶
```

You can allow intercepts to vary across subjects in the same ways as above (you can also allow slopes to vary, but I will not discuss that here), you can plot the main effects or the interactions of such models with `effect`, and you can even apply `Anova(model)` to `lme` models to get *p*-values for the fixed-effect predictors.

Seems simple, doesn't it? Why isn't there a whole section on this, explaining and exemplifying it all in detail as for the other models in this chapter. Well, unfortunately, things are very far from being that simple. In fact, mixed-effects modeling is one of the most fascinating but also among the most complex statistical techniques I have seen. Right now, it actually seems to be seen as the best thing since sliced bread, and indeed the potential of this approach is immense and far-reaching. Having said that, I must admit that I sometimes think that some of the hype about this method is a bit premature simply because so many things are still unclear. Ask any two or three experts on how to do X with multi-level models, and you often get very different responses. Pick any two to three references on mixed-effects modeling and you will see that not only is there very little agreement on some seemingly central questions, but also that some types of problems are not even mentioned very widely. For example,

- it seems we're not even close to a somewhat widely accepted view on what a *model selection process* or even just a *maximal model* would look like. Some sources recommend a model selection process where we begin with no fixed effects but first explore random effects; others recommend starting with a full-fledged fixed-effects maximal model; some recommend beginning with a simple random-effects structure (just intercepts), others recommend beginning with a maximal random-effect structure with random intercepts and slopes for everything (and then simulations suggest that these models do no converge even if they are given the right model structure) …;
- it is not clear yet how *predictors should be selected* for retention in, or deletion from a model: some use *p*-values (based on *t*- or *F*-values, but then it's debated how to choose the residual *df*s), some use MCMC sampling (which is not easily available for some types of dependent variables); some use information criteria (such as *AIC* or *BIC* or even *DIC*) for the whole process; some use likelihood ratio tests, which require attention to whether the models have been fit with ML or REML, …;

- many references do not discuss how to handle the *intercorrelations of random intercepts and slopes*;
- many references say practically nothing about how to decide on a *covariance structure of the data*; I think I have seen only one reference discussing this in a somewhat accessible fashion;
- you have seen that *centering variables* can be useful in regression modeling but how to do this best in mixed-effects models is again often not discussed well – when do we center around an overall mean, when around group means? And the list goes on, boundary effects, how to compute R^2s, …

None of the above is to deny that mixed-effects modeling is very powerful and has the potential to help us very much in analyzing our data … once the field has developed a bit more of some common thoughts on how they should be applied to the various kinds of data out there. The fact that now some journals already *require* mixed-effects modeling for particular data sets seems a bit overeager, given how many open questions remain. However, once some standards regarding the many open questions begin to emerge and once some libraries and functions are developed that make tackling some of these questions more easily (Baayen's `pvals.fnc` is one case in point), then the discipline will benefit from mixed-effects models in innumerable ways. Till that happens, here are some, I think, very good references (of varying degrees of technicality) that will hopefully get you started beyond this little primer …

Recommendation(s) for further study

- for repeated-measures ANOVAs: Girden (1992), Johnson (2008: Sections 4.3-4.4), and especially Miles, Field, and Miles (2012: Ch. 13-14)
- for mixed-effects/multi-level models: Twisk (2006), Gelman and Hill (2007: Ch. 11-15), Zuur, Ieno, and Smith (2007: Ch. 8), Baayen (2008: Ch. 7), Baayen, Davidson, and Bates (2008), Johnson (2008: Sections 7.3, 7.4), Zuur et al. (2009, in particular Ch. 5), Miles, Field, and Miles (2012: Ch. 19), Baguley (2012: Ch. 18); also see Baayen (2011)

6. Hierarchical agglomerative cluster analysis

We have so far only concerned ourselves with methods in which independent and dependent variables were clearly separated and where we already

had at least an expectation and a hypothesis prior to the data collection. Such methods are sometimes referred to as *hypothesis-testing statistics*, and we used statistics and *p*-values to decide whether or not to reject a H_0. The method called hierarchical agglomerative cluster analysis that we deal with in this section is a so-called *exploratory*, or *hypothesis-generating*, method or, more precisely, a family of methods. It is normally used to divide a set of elements into clusters, or groups, such that the members of one group are very similar to each other and at the same time very dissimilar to members of other groups. An obvious reason to use cluster analyses to this end is that this method can handle larger amounts of data and be at the same time more objective than humans eyeballing huge tables.

To get a first understanding of what cluster analyses do, let us look at a fictitious example of a cluster analysis based on similarity judgments of English consonant phonemes. Let's assume you wanted to determine how English native speakers distinguish the following consonant phonemes: /b/, /d/, /f/, /g/, /l/, /m/, /n/, /p/, /s/, /t/, and /v/. You asked 20 subjects to rate the similarities of all $^{(11 \cdot 10)}/_{2} = 55$ pairs of consonants on a scale from 0 ('completely different') to 1 ('completely identical'). As a result, you obtained 20 similarity ratings for each pair and could compute an average rating for each pair. It would now be possible to compute a cluster analysis on the basis of these average similarity judgments to determine (i) which consonants and consonant groups the subjects appear to distinguish and (ii) how these groups can perhaps be explained. Figure 79 shows the result that such a cluster analysis might produce – how would you interpret it?

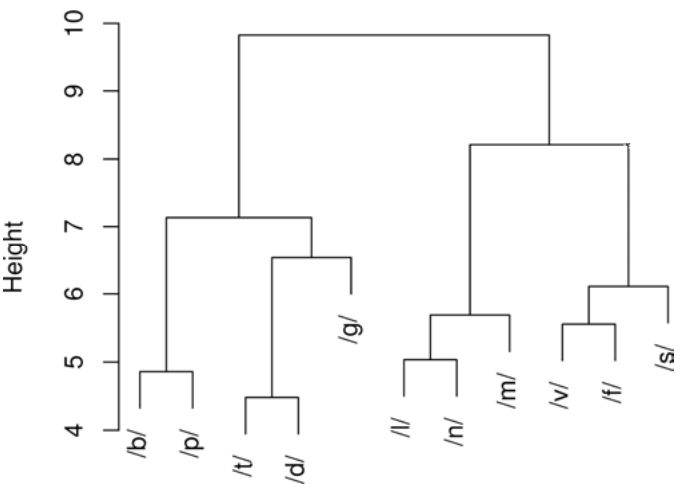

Figure 79. Fictitious results of a cluster analysis of English consonants

THINK
BREAK

The 'result' suggests that the subjects' judgments were probably strongly influenced by the consonants' manner of articulation: on a very general level, there are two clusters, one with /b/, /p/, /t/, /d/, and /g/, and one with /l/, /n/, /m/, /v/, /f/, and /s/. It is immediately obvious that the first cluster contains all and only all plosives (i.e., consonants whose production involves a momentary *complete* obstruction of the airflow) that were included whereas the second cluster contains all and only all nasals, liquids, and fricatives (i.e., consonants whose production involves only a momentary *partial* obstruction of the airflow).

There is more to the results, however. The first of these two clusters has a noteworthy internal structure of two 'subclusters'. The first subcluster, as it were, contains all and only all bilabial phonemes whereas the second subcluster groups both alveolars together followed by a velar sound.

The second of the two big clusters also has some internal structure with two subclusters. The first of these contains all and only all nasals and liquids (i.e., phonemes that are sometimes classified as between clearcut vowels and clearcut consonants), and again the phonemes with the same place of articulation are grouped together first (the two alveolar sounds). The same is true of the second subcluster, which contains all and only all fricatives and has the labiodental fricatives merged first.

The above comments were only concerned with which elements are members of which clusters. Further attempts at interpretation may focus on how many of the clusters in Figure 79 differ from each other strongly enough to be considered clusters in their own right. Such discussion is ideally based on follow-up tests which are too complex to be discussed here, but as a quick and dirty heuristic you can look at the lengths of the vertical lines in such a tree diagram, or dendrogram. Long vertical lines indicate more autonomous subclusters. For example, the subcluster {/b/ /p/} is rather different from the remaining plosives since the vertical line leading upwards from it to the merging with {{/t/ /d/} /g/} is rather long.[37]

Unfortunately, cluster analyses do not usually yield such a perfectly interpretable output but such dendrograms are often surprisingly interesting

37. For a similar but authentic example (based on data on vowel formants), cf. Kornai (1998).

and revealing. Cluster analyses are often used in semantic, cognitive-linguistic, psycholinguistic, and computational-linguistic studies (cf. Miller 1971, Sandra and Rice 1995, Rice 1996, and Manning and Schütze 1999: Ch. 14 for some examples) and are often an ideal means to detect patterns in large and seemingly noisy/chaotic data sets. You must realize, however, that even if cluster analyses as such allow for an objective identification of groups, the analyst must still make at least three potentially subjective decisions. The first two of these influence how exactly the dendrogram will look like; the third you have already seen: one must decide what it is the dendrogram reflects. In what follows, I will show you how to do such an analysis with R yourself. Hierarchical agglomerative cluster analyses typically involve the following steps:

Procedure

Tabulating the data
- Computing a similarity/dissimilarity matrix on the basis of a user-defined similarity/dissimilarity metric
- Computing a cluster structure on the basis of a user-defined amalgamation rule
- Representing the cluster structure in a dendrogram and interpreting it
- (Post-hoc exploration (such as average silhouette widths)

The example we are going to discuss is from the domain of corpus/computational linguistics. In both disciplines, the degree of semantic similarity of two words is often approximated on the basis of the number and frequency of shared collocates. A very loose definition of a 'collocates of a word *w*' are the words that occur frequently in *w*'s environment, where *environment* in turn is often defined as 'in the same sentence' or within a 4- or 5-word window around *w*. For example: if you find the word *car* in a text, then very often words such as *driver*, *motor*, *gas*, and/or *accident* are relatively nearby whereas words such as *flour*, *peace treaty*, *dictatorial*, and *cactus collection* are probably not particularly frequent. In other words, the more collocates two words *x* and *y* share, the more likely there is a semantic relationship between the two (cf. Oakes 1998: Ch. 3, Manning and Schütze 2000: Section 14.1 and 15.2 as well as Gries 2009a for how to obtain collocates in the first place).

In the present example, we look at the seven English words *bronze*, *gold*, *silver*, *bar*, *cafe*, *menu*, and *restaurant*. Of course, I did not choose these words at random – I chose them because they intuitively fall into two clusters with bar (and thus constitute a good test case). One cluster consists

of three co-hyponyms of the *metal*, the other consists of three co-hyponyms of *gastronomical establishment* as well as a word from the same semantic field. Let us assume you extracted from the British National Corpus (BNC) all occurrences of these words and their content word collocates (i.e., nouns, verbs, adjectives, and adverbs). For each collocate that occurred with at least one of the seven words, you determined how often it occurred with each of the seven words. Table 46 is a schematic representation of the first six rows of such a table. The first collocate, here referred to as *X*, co-occurred only with bar (three times); the second collocate, *Y*, co-occurred 11 times with *gold* and once with *restaurant*, etc.

Table 46. Schematic co-occurrence frequencies of seven English words in the BNC

Collocate	*bronze*	*gold*	*silver*	*bar*	*cafe*	*menu*	*restaurant*
X	0	0	0	3	0	0	0
Y	0	11	0	0	0	0	1
Z	0	1	1	0	0	0	1
A	0	0	0	1	0	2	0
B	1	0	0	1	0	0	0
C	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...

We are now asking the question which words are more similar to each other than to others. That is, just like in the example above, you want to group elements – above, phonemes, here, words – on the basis of properties – above, average similarity judgments, here, co-occurrence frequencies. First you need a data set such as Table 46, which you can load from the file <_inputfiles/05-6_collocates.RData>, which contains a large table of co-occurrence data – seven columns and approximately 31,000 rows.

```
> load(file.choose ()) # load the data frame¶
> ls() # check what was loaded¶
[1] "collocates"
> str(collocates)¶
'data.frame':   30936 obs. of  7 variables:
 $ bronze    : num  0 0 0 0 1 0 0 0 0 0 0 ...
 $ gold      : num  0 11 1 0 0 0 0 1 0 0 ...
 $ silver    : num  0 0 1 0 0 0 0 0 0 0 ...
 $ bar       : num  3 0 0 1 1 1 1 0 1 0 ...
 $ cafe      : num  0 0 0 0 0 0 0 0 0 1 ...
 $ menu      : num  0 0 0 2 0 0 0 0 0 0 ...
> attach(collocates)¶
 $ restaurant: num  0 1 0 0 0 0 0 0 0 0 ...
```

Alternatively, you could load those data with `read.table(…)` from the file <_inputfiles/05-6_collocates.csv>. If your data contain *missing data*, you should disregard those. There are no missing data, but the function is still useful to know (cf. the recommendation at the end of Chapter 2):

```
> collocates<-na.omit(collocates)¶
```

Next, you must generate a similarity/dissimilarity matrix for the seven words. Here, you have to make the first possibly subjective decision, deciding on a similarity/dissimilarity measure. You need to consider two aspects: the level of measurement of the variables in point and the definition of similarity to be used. With regard to the former, we will only distinguish between binary/nominal and ratio-scaled variables. I will discuss similarity/dissimilarity measures for both kinds of variables, but will then focus on ratio-scaled variables.

In the case of nominal variables, there are four possibilities how two elements can be similar or dissimilar to each other, which are represented in Table 47. On the basis of Table 47, the similarity of two elements is typically quantified using formula (66), in which w_1 and w_2 are defined by the analyst:

$$\frac{a + w_1 \cdot d}{(a + w_1 \cdot d) + (w_2 \cdot (b + c))} \tag{66}$$

Table 47. Feature combinations of two binary elements

	Element 2 exhibits characteristic *x*	Element 2 does not exhibit characteristic *x*
Element 1 exhibits characteristic *x*	*a*	*b*
Element 1 does not exhibit characteristic *x*	*c*	*d*

Three similarity measures are worth mentioning here:

- the Jaccard coefficient: $w_1 = 0$ and $w_2 = 1$;
- the Simple Matching coefficient: $w_1 = 1$ and $w_2 = 1$;
- the Dice coefficient: $w_1 = 0$ and $w_2 = 0.5$.

What are their pairwise similarity coefficients of these three vectors?

```
> aa<-c(1, 1, 1, 1, 0, 0, 1, 0, 0, 0)¶
> bb<-c(1, 1, 0, 1, 0, 1, 0, 1, 0, 1)¶
> cc<-c(1, 0, 1, 1, 1, 1, 1, 1, 1, 0)¶
```

THINK BREAK

- Jaccard coefficient: for `aa` and `bb`: 0.375, for `aa` and `cc` 0.444, for `bb` and `cc` 0.4;
- Simple Matching coefficient: for `aa` and `bb`: 0.5, for `aa` and `cc` 0.5, for `bb` and `cc` 0.4;
- Dice coefficient: for `aa` and `bb`: 0.545, for `aa` and `cc` 0.615, for `bb` and `cc` 0.571 (see the code file for a function that computes these).

But when do you use which of the three? One rule of thumb is that when the presence of a characteristic is as informative as its absence, then you should use the Simple Matching coefficient, otherwise choose the Jaccard coefficient or the Dice coefficient. The reason for that is that, as you can see in formula (66) and the measures' definitions above, only the Simple Matching coefficient fully includes the cases where both elements exhibit or do not exhibit the characteristic in questions.

For ratio-scaled variables, there are (many) other measures, not all of which I can discuss here. I will focus on (i) a set of distance or dissimilarity measures (i.e., measures where large values represent large degrees of dissimilarity) and (ii) a set of similarity measures (i.e., measures where large values represent large degrees of similarity). Many distance measures are again based on one formula and then differ in terms of parameter settings. This basic formula is the so-called Minkowski metric represented in (67).

$$(67) \qquad \left(\sum_{i-1}^{n} \left| x_{qi} - x_{ri} \right|^{y} \right)^{1/y}$$

When y is set to 2, you get the so-called Euclidean distance.[38] If you in-

38. The Euclidean distance of two vectors of length n is the direct spatial distance between two points within an n-dimensional space. This may sound complex, but for the simplest case of a two-dimensional coordinate system this is merely the distance you would measure with a ruler.

sert $y = 2$ into (67) to compute the Euclidean distance of the vectors `aa` and `bb`, you obtain:

```
> sqrt(sum((aa-bb)^2))¶
[1] 2.236068
```

When y is set to 1, you get the so-called Manhattan- or City-Block distance of the above vectors. For `aa` and `bb`, you obtain:

```
> sum(abs(aa-bb))¶
[1] 5
```

The similarity measures are correlational measures. One of these you know already: the Pearson product-moment correlation coefficient r. A similar measure often used in computational linguistics is the cosine (cf. Manning and Schütze 1999: 299–303). The cosine and all other measures for ratio-scaled are available from the function `Dist` from the library `amap`.[39] This function requires that (i) the data are available in the form of a matrix or a data frame and that (ii) the elements whose similarities you want are in the rows, not in the columns as usual. If the latter is not the case, you can often just transpose a data structure (with `t`):

```
> library(amap)¶
> collocates.t<-t(collocates)¶
```

You can then apply the function `Dist` to the transposed data structure. This function takes the following arguments:

- `x`: the matrix or the data frame for which you want your measures;
- `method="euclidean"` for the Euclidean distance; `method="manhattan"` for the City-Block metric; `method="correlation"` for the product-moment correlation r (but see below!); `method="pearson"` for the cosine (but see below!) (there are some more measures available which I won't discuss here);
- `diag=FALSE` (the default) or `diag=TRUE`, depending on whether the distance matrix should contain its main diagonal or not;
- `upper=FALSE` (the default) or `upper=TRUE`, depending on whether the distance matrix should contain only the lower left half or both halves.

39. The function `dist` from the standard installation of R also allows you to compute several similarity/dissimilarity measures, but fewer than `Dist` from the library `amap`.

Thus, if you want to generate a distance matrix based on Euclidean distances for our collocate dataset you simply enter this:

```
> Dist(collocates.t, method="euclidean", diag=TRUE,
       upper=TRUE)¶
```

As you can see, you get a (symmetric) distance matrix in which the distance of each word to itself is of course 0. This matrix now tells you which word is most similar to which other word. For example, the word *silver* is most similar to is *cafe* because the distance of *silver* to *cafe* (2385.566) is the smallest distance that *silver* has to any word other than itself.

The following computes a distance matrix using the City-Block metric:

```
> Dist(collocates.t, method="manhattan", diag=TRUE,
       upper=TRUE)¶
```

To get a similarity matrix with product-moment correlations or cosines, you must compute the difference 1 minus the values in the matrix. To get a similarity matrix with correlation coefficients, you therefore enter this:

```
> 1-Dist(collocates.t, method="correlation", diag=TRUE,
       upper=TRUE)¶
           bronze   gold silver    bar   cafe   menu restaurant
bronze     0.0000 0.1342 0.1706 0.0537 0.0570 0.0462     0.0531
gold       0.1342 0.0000 0.3103 0.0565 0.0542 0.0458     0.0522
silver     0.1706 0.3103 0.0000 0.0642 0.0599 0.0511     0.0578
bar        0.0537 0.0565 0.0642 0.0000 0.1474 0.1197     0.2254
cafe       0.0570 0.0542 0.0599 0.1474 0.0000 0.0811     0.1751
menu       0.0462 0.0458 0.0511 0.1197 0.0811 0.0000     0.1733
restaurant 0.0531 0.0522 0.0578 0.2254 0.1751 0.1733     0.0000
```

You can check the results by comparing this output with the one you get from `cor(collocates)¶`. For a similarity matrix with cosines, you enter:

```
> 1-Dist(collocates.t, method="pearson", diag=TRUE,
       upper=TRUE)¶
```

There are also statistics programs that use 1-*r* as a distance measure. They change the similarity measure *r* (values close to zero mean low similarity) into a distance measure (values close to zero mean high similarity).

If you compare the matrix with Euclidean distances with the matrix with *r*, you might notice something that strikes you as strange …

THINK
BREAK

In the distance matrix, small values indicate high similarity and the smallest value in the column *bronze* is in the row for *cafe* (1734.509). In the similarity matrix, large values indicate high similarity and the largest value in the column *bronze* is in the row for *silver* (ca. 0.1706). How can that be? This difference shows that even a cluster algorithmic approach is influenced by subjective though hopefully motivated decisions. The choice for a particular metric influences the results because there are different ways in which vectors can be similar to each other. Consider as an example the following data set, which is also represented graphically in Figure 80.

```
> y1<-1:10; y2<-11:20; y3<-c(6, 6, 6, 5, 5, 5, 4, 4, 4, 3)¶
> y<-t(data.frame(y1, y2, y3))¶
```

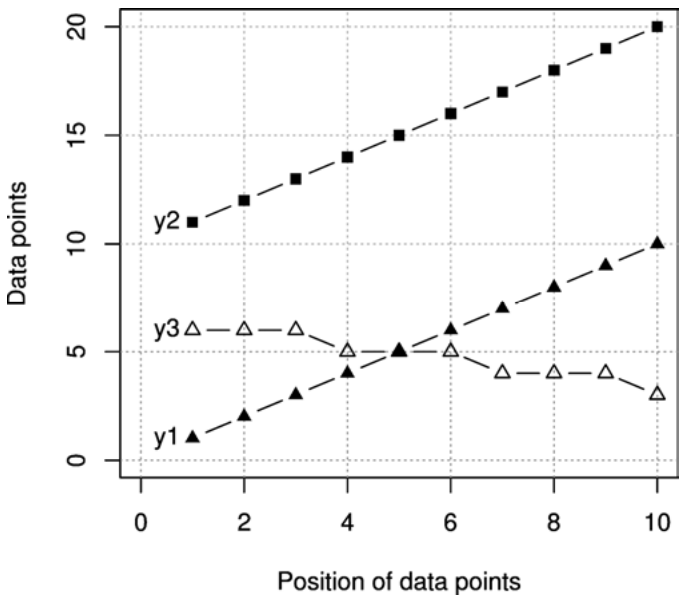

Figure 80. Three fictitious vectors

The question is, how similar is `y1` to `y2` and to `y3`? There are two obvious ways of considering similarity. On the one hand, `y1` and `y2` are perfectly parallel, but they are far away from each other (as much as one can say

that about a diagram whose dimensions are not defined). On the other hand, y1 and y3 are not parallel to each other at all, but they are close to each other. The two approaches I discussed above are based on these different perspectives. The distance measures I mentioned (such as the Euclidean distance) are based on the spatial distance between vectors, which is small between y1 and y3 but large between y1 and y2. The similarity measures I discussed (such as the cosine) are based on the similarity of the curvature of the vectors, which is small between y1 and y3, but large between y1 and y2. You can see this quickly from the actual numerical values:

```
> Dist(y, method="euclidean", diag=TRUE, upper=TRUE)¶
         y1       y2       y3
y1  0.00000 31.62278 12.28821
y2 31.62278  0.00000 35.93049
y3 12.28821 35.93049  0.00000
> 1-Dist(y, method="pearson", diag=TRUE, upper=TRUE)¶
           y1        y2        y3
y1 0.0000000 0.9559123 0.7796728
y2 0.9559123 0.0000000 0.9284325
y3 0.7796728 0.9284325 0.0000000
```

According to the Euclidean distance, y1 is more similar to y3 than to y2 – 12.288 < 31.623 – but the reverse is true for the cosine: y1 is more similar to y2 – 0.956 > 0.78. The two measures are based on different concepts of similarity. The analyst must decide what is more relevant: low spatial distances or similar curvatures. For now, we assume you want to adopt a curvature-based approach and use 1-*r* as a measure; in your own studies, you of course must state which similarity/distance measure you used, too.[40]

```
> dist.matrix<-Dist(collocates.t, method="correlation",
      diag=TRUE, upper=TRUE)¶
> round(dist.matrix, 4)¶
           bronze   gold silver    bar   cafe   menu restaurant
bronze     0.0000 0.8658 0.8294 0.9463 0.9430 0.9538     0.9469
gold       0.8658 0.0000 0.6897 0.9435 0.9458 0.9542     0.9478
silver     0.8294 0.6897 0.0000 0.9358 0.9401 0.9489     0.9422
bar        0.9463 0.9435 0.9358 0.0000 0.8526 0.8803     0.7746
cafe       0.9430 0.9458 0.9401 0.8526 0.0000 0.9189     0.8249
menu       0.9538 0.9542 0.9489 0.8803 0.9189 0.0000     0.8267
restaurant 0.9469 0.9478 0.9422 0.7746 0.8249 0.8267     0.0000
```

The next step is to compute a cluster structure from this similarity ma-

40. I am simplifying a lot here: the frequencies are neither normalized nor logged/dampened etc. (cf. above, Manning and Schütze 1999: Section 15.2.2, or Jurafsky and Martin 2008: Ch. 20).

trix. You do this with the function `hclust`, which can take up to three arguments of which I will discuss two. The first is a similarity/distance matrix, the second chooses an amalgamation rule that defines how the elements in that matrix get merged into clusters. This choice is the second potentially subjective decision and there are again several possibilities.

The choice `method="single"` uses the so-called *single-linkage-* or *nearest-neighbor* method. In this method, the similarity of elements *x* and *y* – where *x* and *y* may be elements such as individual consonants or subclusters such as {/b/, /p/} in Figure 79 – is defined as the *minimal* distance between any one element of *x* and any one element of *y*. In the present example this means that in the first amalgamation step *gold* and *silver* would be merged since their distance is the smallest in the whole matrix (1-*r* = 0.6897). Then, *bar* gets joined with *restaurant* (1-*r* = 0.7746). Then, and now comes the interesting part, {*bar restaurant*} gets joined with *cafe* because the smallest remaining distance is that which *restaurant* exhibits to *cafe*: 1-*r* = 0.8249. And so on. This amalgamation method is good at identifying outliers in data, but tends to produce long chains of clusters and is, therefore, often not particularly discriminatory.

The choice `method="complete"` uses the so-called *complete-linkage-* or *furthest-neighbor* method. Contrary to the single-linkage method, here the similarity of *x* and *y* is defined as the *maximal* distance between any one element of *x* and any one element of *y*. First, *gold* and *silver* are joined as before, then *bar* and *restaurant*. In the third step, {*bar restaurant*} gets joined with *cafe*, but the difference to the single linkage method is that the distance between the two is now 0.8526, not 0.8249, because this time the algorithm considers the maximal distances, of which the smallest is chosen for joining. This approach tends to form smaller homogeneous groups and is a good method if you suspect there are many smaller groups in your data.

Finally, the choice `method="ward"` uses a method whose logic is similar to that of ANOVAs because it joins those elements whose joining increases the error sum of squares least. For every possible amalgamation, the method computes the sums of squared differences/deviations from the mean of the potential cluster, and then the clustering with the smallest sum of squared deviations is chosen. This method is known to generate smaller clusters that are often similar in size and has proven to be quite useful in many applications. We will use it here, too, and again in your own studies, you must explicitly state which amalgamation rule you used. Now you can compute the cluster structure and plot it.

```
> clust.ana<-hclust(dist.matrix, method="ward")¶
```

```
> plot(clust.ana)¶
> rect.hclust(clust.ana, 2) # red boxes around clusters¶
```

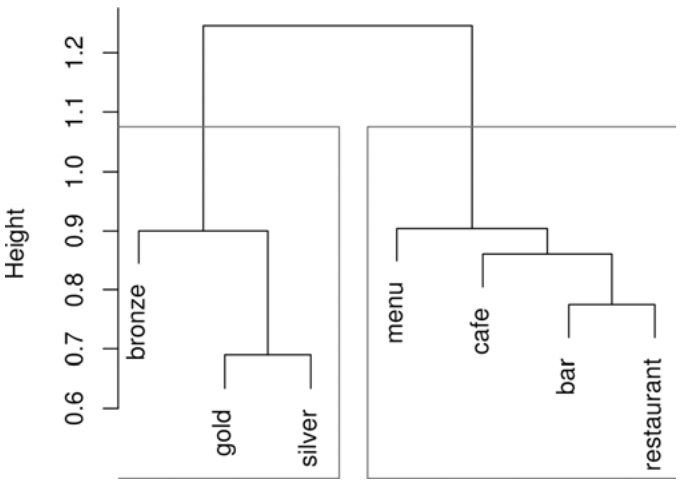

Figure 81. Dendrogram of seven English words

This is an uncharacteristically clearly interpretable result. As one would have hoped for, the seven words fall exactly into the two main expected clusters: one with the 'metals' and one with the gastronomy-related words. The former has a substructure in which *bronze* is somewhat less similar to the other two metals, and the latter very little substructure but groups the three co-hyponyms together before *menu* is added. With the following line you can have R show you for each element which cluster it belongs to when you assume two clusters.

```
> cutree(clust.ana, 2)¶
bronze     gold     silver      bar     cafe     menu restaurant
     1        1          1        2        2        2          2
```

While I can't discuss the method in detail, I want to briefly give you at least a glimpse of how more difficult cluster structures can be explored. As you will remember, Figure 79 was a much less clear-cut case in terms of how many clusters should be distinguished: any number between 2 and 5 seems defensible. The function cluster.stats from the library fpc offers a variety of validation statistics, which can help to narrow down the number of clusters best distinguished. One of these involves the notion of average silhouette widths, which quantifies how similar elements are to the clusters which they are in relative to how similar elements are to other clusters. It is

then possible to compute average silhouette widths for all possible cluster solutions and pick the one with the highest average silhouette widths. If we apply this logic to Figure 79, we get Figure 82. It shows why the decision for any one number of clusters is so difficult – many solutions fare nearly equally well – but why, if anything, four clusters sould be distinguished: with four clusters, the average silhouette width is highest: 0.14.

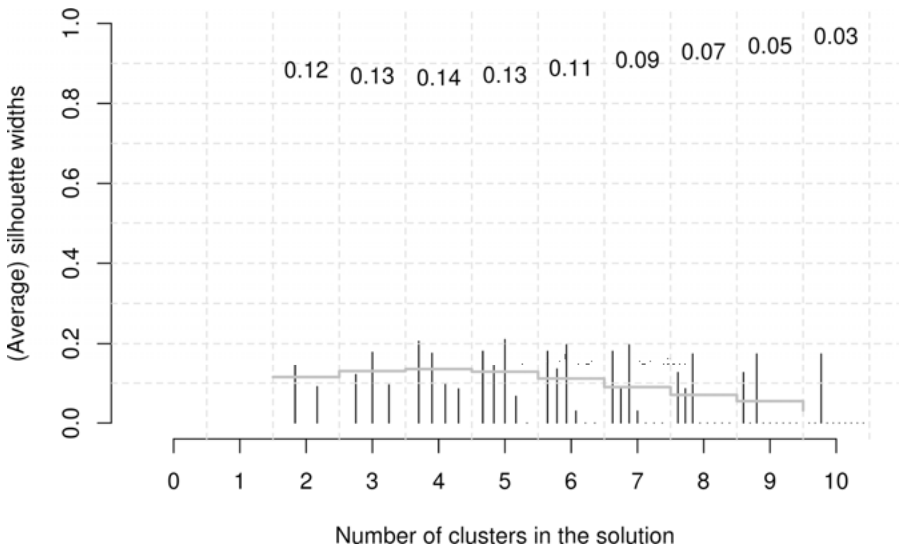

Figure 82. Average silhouette widths for all cluster solutions of Figure 79

Now you should do the exercises for Chapter 5 …

Recommendation(s) for further study

- the function `daisy` (from the library `cluster`) to compute distance matrices for dataset with variables from different levels of measurement
- the function `kmeans` to do cluster analyses where you provide the number of clusters beforehand
- the function `pvclust` (from the library `pvclust`) to obtain *p*-values for clusters based on resampling methods; cf. also `pvrect` and `pvpick` (from the same library)
- the function `varclus` (from the library `Hmisc`) to do variable clustering
- the function `nj` (from the library `ape`) to perform neighbor clustering and phylogenetic cluster analyses
- Crawley (2007: Ch. 23), Baayen (2008: Ch. 5), Johnson (2008: Ch. 6)

Chapter 6
Epilog

Now that you have nearly made it through the whole book, let me give you a little food for further thought and some additional ideas on the way. Ironically, some of these will probably shake up a bit what you have learnt so far, but I hope they will also stimulate some curiosity for what else is out there to discover and explore.

Let me first mention a few areas that you should begin to explore as you become more familiar with regression modeling. One issue I have only alluded to in passing in the code file is that of *(cross) validation*. Regressions often run the risk of what is called *overfitting*: they fit a particular data set rather well, but generalize badly to others, which of course jeopardizes the generalizability of the findings to the population as a whole. Very often, results can be validated by splitting up the existing sample into, often, 10 parts and then do 10 analyses, in each of which you obtain a regression equation from 90% of the data and apply it to the unseen 10%. Such methods can reveal a lot about the internal structure of a data set and there are several functions available in R for these methods. A related point is that, given the ever increasing power of computers, resampling and permutation approaches become more and more popular; examples include the *bootstrap*, the *jackknife* procedure, or *exhaustive permutation procedures*. These procedures are non-parametric methods you can use to estimate means, variances, but also correlations or regression parameters without major distributional assumptions. Such methods are not the solution to all statistical problems, but can still be interesting and powerful tools (cf. the libraries `boot` as well as `bootstrap`).

Recommendation(s) for further study
Good (2005), Rizzo (2008: Ch. 7, 8)

Also, the analysis of special data points in your sample(s) is very important, given the impact that *outliers* and *points with high leverage* can have on the data. In addition, learning more about what to do with missing data should be high on your list of things. On the one hand, it may be useful, for instance, to run a regression on missing data to see whether there is something in the data that allows you to predict well when, say, subjects do

respond to a stimulus. On the other hand, small proportions of missing data may be *imputed*, that is predicted from other data points (see Torgo: Section 2.5).

Then, there is a range of additional techniques you may wish to explore. This book focused on hypothesis-testing approaches, in particular regressions, but there are many interesting exploratory tools that, for reasons of space, I could not discuss: *principal components analysis* and *correspondence analysis* are two well-known cases in point, *association rules* or *naïve Bayes classifiers* are others.

It is also worth pointing out that R has many many more possibilities of graphical representation than I could mention here. I only used the traditional graphics system, but there are other more powerful tools, which are available from the libraries `lattice` and `ggplot2` (you should explore <http://www.yeroon.net/ggplot2/>). The website <http://gallery.r-enthusiasts.com/> provides many very interesting and impressive examples for R plots, and several good books illustrate many of the exciting possibilities for exploration (cf. Unwin, Theus, and Hofmann 2006, Cook and Swayne 2007, Sarkar 2008, Keen 2010, and of course Murrell 2011).

Finally, note that the *null hypothesis significance testing (NHST) paradigm* that is underlying most of the methods discussed here is not as uncontroversial as this textbook (and most others) may make you believe. While the computation of *p*-values is certainly still the standard approach, there are researchers who argue for a different perspective. Some of these argue that *p*-values are problematic because they do in fact not represent the conditional probability that one is really interested in. Recall, the above *p*-values answer the question "How likely is it to get the observed data when H_0 is true?" but what one actually wants to know "How likely is H_1 given the data I have?" Suggestions for improvement include:

- one should focus not on *p*-values but on effect sizes and/or confidence intervals (which is why I mentioned these above again and again);
- one should report so-called p_{rep}-values, which according to Killeen (2005) provide the probability to replicate an observed effect (but are not uncontroversial themselves);
- one should test reasonable H_0s rather than hypotheses that could never be true in the first place (there will always be some effect or difference).

Another interesting approach is the so-called *Bayesian approach* to statistics, which allows to include subjective prior knowledge or previous results with one's own data. All of these things are worth exploring.

Recommendation(s) for further study
- Cohen (1994), Loftus (1996), Denis (2003) for discussion of the NHST
- Killeen (2005) on p_{rep}-values
- Iversen (1984) on Bayes statistics

I hope you can use the techniques covered in this book for many different questions, and when this little epilog also makes you try and extend your knowledge and familiarize yourself with additional tools and methods – for example, there are many great web resources, <http://www.statmethods.net/index.html> and <http://www.r-bloggers.com/> are among my favorites – then this book has achieved one of his main objectives.

References

Agresti, Alan
2002 *Categorical Data Analysis*. 2nd ed. Hoboken, NJ: John Wiley and Sons.

Agresti, Alan
2010 *Analysis of Ordinal Categorical Data*. 2nd ed. Hoboken, NJ: John Wiley and Sons.

Anscombe, Francis J.
1973 Graphs in statistical analysis. *American Statistician* 27: 17–21.

Baayen, R. Harald
2008 *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*. Cambridge: Cambridge University Press.

Baayen, R. Harald, D.J. Davidson, and Douglas M. Bates.
2008 Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language* 59 (4): 390-412.

Baayen, R. Harald
2010 A real experiment is a factorial experiment? *The Mental Lexicon* 5 (1): 149-157.

Baayen, R. Harald
2011 Corpus linguistics and naïve discriminative learning. *Brazilian Journal of Applied Linguistics* 11 (2): 295-328.

Backhaus, Klaus, Bernd Erichson, Wulff Plinke, and Rolf Weiber
2003. *Multivariate Analysemethoden: eine anwendungsorientierte Einführung*. 10th ed. Berlin: Springer.

Baguley, Thom
2012 *Serious Stats: A Guide to Advanced Statistics for the Behavioral Sciences*. Houndmills, Basingstoke, Hampshire: Palgrave MacMillan.

Bencini, Giulia, and Adele E. Goldberg
2000 The contribution of argument structure constructions to sentence meaning. *Journal of Memory and Language* 43 (3): 640–651.

Berez, Andrea L., and Stefan Th. Gries
2010 Correlates to middle marking in Dena'ina iterative verbs. *International Journal of American Linguistics*.

Bortz, Jürgen
2005 *Statistik for Human- und Sozialwissenschaftler*. 6th ed. Heidelberg: Springer Medizin Verlag.

Bortz, Jürgen, and Nicola Döring
1995 *Forschungsmethoden und Evaluation*. 2nd ed. Berlin, Heidelberg, New York: Springer.

Bortz, Jürgen, Gustav A. Lienert, and Klaus Boehnke
1990 *Verteilungsfreie Methoden in der Biostatistik*. Berlin, Heidelberg, New York: Springer.
Braun, W. John, and Duncan J. Murdoch
2008 *A First Course in Statistical Programming with R*. Cambridge: Cambridge University Press.
Brew, Chris, and David McKelvie
1996 Word-pair extraction for lexicography. In *Proceedings of the 2nd International Conference on New Methods in Language Processing*, Kemal O. Oflazer and Harold Somers (eds.), 45–55. Ankara: Bilkent University.
Bretz, Frank, Torsten Hothorn, and Peter Westfall
2011 *Multiple Comparisons Using R*. Boca Raton, FL: Chapman and Hall/CRC.
Chambers, John M.
2008 *Software for Data Analysis: Programmming with R*. New York: Springer.
Chen, Ping
1986 Discourse and Particle Movement in English. *Studies in Language* 10 (1): 79–95.
Clauß, Günter, Falk Rüdiger Finze, and Lothar Partzsch
1995 *Statistik for Soziologen, Pädagogen, Psychologen und Mediziner*. Vol. 1. 2nd ed. Thun: Verlag Harri Deutsch
Cohen, Jacob
1994 The earth is round ($p < 0.05$). *American Psychologist* 49 (12): 997–1003.
Cook, Dianne, and Deborah F. Swayne
2007 *Interactive and Dynamic Graphics for Data Analysis*. New York: Springer.
Cowart, Wayne
1997 *Experimental Syntax: Applying Objective Methods to Sentence Judgments*. Thousand Oaks, CA: Sage.
Crawley, Michael J.
2002 *Statistical Computing: An Introduction to Data Analysis using S-Plus*. – Chichester: John Wiley.
Crawley, Michael J.
2005 *Statistics: An Introduction Using R*. – Chichester: John Wiley.
Crawley, Michael J.
2007 *The R book*. – Chichester: John Wiley.
Dalgaard, Peter
2002 *Introductory Statistics with R*. New York: Springer.

Denis, Daniel J.
2003 Alternatives to Null Hypothesis Significance Testing. *Theory and Science* 4.1. URL <http://theoryandscience.icaap.org/content/vol4.1/02_denis.html>

Divjak, Dagmar S., and Stefan Th. Gries
2006 Ways of trying in Russian: clustering behavioral profiles. *Corpus Linguistics and Linguistic Theory* 2 (1): 23–60.

Divjak, Dagmar S., and Stefan Th. Gries
2008 Clusters in the mind? Converging evidence from near synonymy in Russian. *The Mental Lexicon* 3 (2):188–213.

Everitt, Brian S., and Torsten Hothorn
2006 *A handbook of statistical analyses using R*. Boca Raton, FL: Chapman and Hall/CRC.

von Eye, Alexander
2002 *Configural frequency analysis: methods, models, and applications*. Mahwah, NJ: Lawrence Erlbaum.

Faraway, Julian J.
2005 *Linear models with R*. Boca Raton: Chapman and Hall/CRC.

Faraway, Julian J.
2006 *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression models*. Boca Raton: Chapman and Hall/CRC.

Field, Andy, Jeremy Miles, and Zoë Field
2012 *Discovering Statistics Using R*. Los Angeles and London: Sage Publications.

Frankenberg-Garcia, Ana
2004 Are translations longer than source texts? A corpus-based study of explicitation. Paper presented at Third International CULT (Corpus Use and Learning to Translate) Conference, Barcelona, 22–24. Januar 2004.

Fraser, Bruce
1966 Some remarks on the VPC in English. In *Problems in Semantics, History of Linguistics, Linguistics and English*, Francis P. Dinneen (ed.), p. 45–61. Washington, DC: Georgetown University Press.

Gaudio, Rudolf P.
1994 Sounding gay: pitch properties in the speech of gay and straight men. *American Speech* 69 (1): 30–57.

Gelman, Andrew, and Jennifer Hill
2007 *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge: Cambridge University Press.

Gentleman, Robert
2009 *R Programming for Bioinformatics*. Boca Raton, FL: Chapman and Hall/CRC.

Good, Philip I.
2005 *Introduction to Statistics through Resampling Methods and R/S-Plus*. Hoboken, NJ: John Wiley and Sons.
Good, Philip I., and James W. Hardin
2012 *Common Errors in Statistics (and How to Avoid Them)*. 4th ed. Hoboken, NJ: John Wiley and Sons.
Gorard, Stephen
2004 Revisiting a 90-year-old debate: the advantages of the mean deviation. Paper presented at the British Educational Research Association Annual Conference, University of Manchester. http://www.leeds.ac.uk/educol/documents/00003759.htm.
Gries, Stefan Th.
2003a *Multifactorial Analysis in Corpus Linguistics: A Study of Particle Placement*. London, New York: Continuum.
Gries, Stefan Th.
2003b Towards a corpus-based identification of prototypical instances of constructions. *Annual Review of Cognitive Linguistics* 1: 181–200.
Gries, Stefan Th.
2006 Cognitive determinants of subtractive word-formation processes: a corpus-based perspective. *Cognitive Linguistics* 17 (4): 535–558.
Gries, Stefan Th.
2009a *Quantitative Corpus Linguistics with R: A Practical Introduction*. London, New York: Taylor and Francis.
Gries, Stefan Th.
2009b *Statistics for Linguistics with R: A Practical Introduction*. Berlin, New York: Mouton de Gruyter.
Gries, Stefan Th.
forthc. Frequency tables: tests, effect sizes, and explorations.
Gries, Stefan Th., and Stefanie Wulff
2005 Do foreign language learners also have constructions? Evidence from priming, sorting, and corpora. *Annual Review of Cognitive Linguistics* 3: 182–200.
Harrell, Frank E. Jr.
2001 *Regression Modeling Strategies. With Applications to Linear Models, Logistic Regression, and Survival Analysis*. New York: Springer.
Hawkins, John A.
1994 *A Performance Theory of Order and Constituency*. Cambridge: Cambridge University Press.
Hilbe, Joseph M.
2009 *Logistic Regression Models*. Boca Raton, FL: Chapman and Hall/CRC.
Iversen, Gudmund R.
1984 *Bayesian Statistical Inference*. Beverly Hills, CA: Sage.

Jaeger, T. Florian
2008 Categorical data analysis: away from ANOVAs (transformation or not) and towards logit mixed models. *Journal of Memory and Language* 59 (4): 434–446
Jaccard, James
2001 *Interaction Effects in Logistic Regression*. Thousand Oaks, CA: Sage.
Johnson, Keith
2008 *Quantitative Methods in Linguistics*. Malden, MA: Blackwell.
Jurafsky, Daniel, and James H. Martin
2008 *Speech and Language Processing*. 2nd ed.. Upper Saddle River, NJ: Pearson Prentice Hall.
Keen, Kevin J.
2010 *Graphics for Statistics and Data Analysis with R*. Boca Raton, FL: Chapman and Hall/CRC.
Killeen, Peter R.
2005 An alternative to null-hypothesis significance tests. *Psychological Science* 16 (5): 345–353.
Kornai, Andras
1998 Analytic models in phonology. In *The Organization of Phonology: Constraints, Levels and Representations*, Jaques Durand and Bernard Laks (eds.), 395–418. Oxford: Oxford University Press.
Krauth, Joachim
1993 *Einführung in die Konfigurationsfrequenzanalyse*. Weinheim: Beltz.
Kučera, Henry, and W. Nelson Francis
1967 *Computational analysis of Present-Day American English*. Providence, RI: Brown University Press.
Larson-Hall, Jennifer
2010 *A guide to doing statistics in second language research using SPSS*. London and New York: Routledge.
Lautsch, Erwin, and Stefan von Weber
1995 *Methoden und Anwendungen der Konfigurationsfrequenzanalyse*. Weinheim: Beltz.
Ligges, Uwe
2005 *Programmieren mit R*. Berlin, Heidelberg, New York: Springer.
Loftus, Geoffrey R.
1996 Psychology will be a much better science when we change the way we analyze data. *Current Directions in Psychological Science* 5 (6): 161–171.
Maindonald, W. John, and John Braun
2003 *Data Analysis and Graphics Using R: An Example-based Approach*. Cambridge: Cambridge University Press.
Manning, Christopher D., and Hinrich K. Schütze
2000 *Foundations of Statistical Natural Language Processing*. Cambridge, MA: The MIT Press.

Marascuilo, Leonard A., and Maryellen McSweeney
1977 *Nonparametric and Distribution-free Methods for the Social Sciences*. Monterey, CA: Brooks/Cole.
Matt, Georg E., and Thomas D. Cook
1994 Threats to the validity of research synthesis. In *The Handbook of Research Synthesis*, H. Cooper and L.V. Hedges (eds.), 503–520. New York: Russell Sage Foundation.
Miller, George A.
1971 Empirical methods in the study of semantics. In *Semantics: An Interdisciplinary Reader*, Danny D. Steinberg and Leon A. Jakobovits (eds.), 569–585. London, New York: Cambridge University Press.
Murrell, Paul
2011 *R graphics*. 2nd ed. Boca Raton, FL: Chapman and Hall/CRC.
Nagata, Hiroshi
1987 Long-term effect of repetition on judgments of grammaticality. *Perceptual and Motor Skills* 65 (5): 295–299.
Nagata, Hiroshi
1989 Effect of repetition on grammaticality judgments under objective and subjective self-awareness conditions. *Journal of Psycholinguistic Research* 18 (3): 255–269.
Oakes, Michael P.
1998 *Statistics for Corpus Linguistics*. Edinburgh: Edinburgh University Press.
Pampel, Fred C.
2000 *Logistic Regression: A Primer*. Thousand Oaks, CA: Sage.
Peters, Julia
2001 Given vs. new information influencing constituent ordering in the VPC. In *LACUS Forum XXVII: Speaking and Comprehending*, Ruth Brend, Alan K. Melby, and Arle Lommel (eds.), 133–140. Fullerton, CA: LACUS.
Rice, Sally
1996 Prepositional prototypes. In *The Construal of Space in Language and Thought*, Martin Pütz and René Dirven (eds.), 35–65, Berlin, New York: Mouton de Gruyter.
Rietverld, Toni, and Roeland van Hout.
2005 *Statistics in Language Research: Analysis of Variance*. Berlin & New York: Springer.
Rizzo, Maria L.
2008 *Statistical Computing with R*. Boca Raton, FL: Chapman and Hall/CRC.
Sandra, Dominiek, and Sally Rice
1995 Network analyses of prepositional meaning: Mirroring whose mind – the linguist's or the language user's? *Cognitive Linguistics* 6 (1): 89–130.

Sarkar, Deepayan
2008 *Lattice: Multivariate Data Visualization with R*. New York: Springer.

Sheskin, David J.
2011 *Handbook of Parametric and Nonparametric Statistical Procedures*. Boca Raton, FL: Chapman and Hall/CRC.
Shirai, Yasuhiro, and Roger W. Andersen
1995 The acquisition of tense-aspect morphology: A prototype account. *Language* 71 (4): 743–762.
Spector, Phil
2008 *Data Manipulation with R*. New York: Springer.
Spencer, Nancy J.
1973 Differences between linguists and nonlinguists in intuitions of grammaticality-acceptability. *Journal of Psycholinguistic Research* 2 (2): 83–98.
Steinberg, Danny D.
1993 *An Introduction to Psycholinguistics*. London: Longman.
Stoll, Sabine, and Stefan Th. Gries
2009 How to characterize development in corpora: an association strength approach. *Journal of Child Language* 36 (5): 1075-1090.
Torgo, Luís
2011 *Data Mining with R: Learning with Case Studies*. Boca Raton, FL: Chapman and Hall/CRC.
Twisk, Jos W.R.
2006 *Applied Multilevel Analysis*. Cambridge: Cambridge University Press.
Unwin, Anthony, Martin Theus, and Heike Hofmann
2006 Graphics of Large Datasets: Visualizing a Million. New York: Springer.
Van Dongen. W. A. Sr.
1919 He Puts on His Hat & He Puts His Hat on. *Neophilologus* 4: 322–353.
Wright, Daniel B., and Kamala London
2009 *Modern Regression Techniques Using R*. Los Angeles, London: Sage.
Zar, Jerrold H.
1999 *Biostatistical Analysis*. 4th ed. Upper Saddle River, NJ: Prentice Hall.
Zuur, Alain F. Elena N. Ieno, and Graham. M. Smith.
2007 *Analysing Ecological Data*. Berlin & New York: Springer
Zuur, Alain F., Elena N. Ieno, Neil Walker and Anatoly A. Saveliev
2009 *Mixed Effects Models and Extensions in Ecology with R*. Berlin & New York: Springer