A hands-on tutorial on using R for (mostly) linguistics research

Çağrı Çöltekin

01-12-2015, 22:59

Note: This version of the exercises are under a reorganization/rewrite process. The older but more complete version can be found found at `http://coltekin.net/cagri/R.old/`.

This is a hands-on tutorial on R, a powerful statistical analysis software. The tutorial is prepared for the course Seminar in Methodology and Statistics taught by John Nerbonne at the University of Groningen.

The aim of the exercises is to provide a hands-on tutorial on some statistical analysis procedures that are common in various branches of linguistics. This tutorial assumes that you are familiar with basic statistical concepts. However, no initial knowledge of R is assumed.

Any suggestions and/or corrections are welcome.

1 Starting R and finding your way around

Depending on the environment or operating system you are using, starting R may be a bit different. Typically you will click on the relevant icon or menu item, or on UNIX-like systems you can run the command `R` on the shell prompt.

When you start R, it will print some default information, and wait for your commands.

First thing you need to get used to (if you are not already) is that R is controlled through a command line interface. After the initial information, you should see the cursor next to the *command prompt* '`>` '. R presents this prompt when it is ready to accept commands.

The command-line interface may feel awkward or old-fashioned at first, but once you get used to, you will see that it is not as scary as it may seem at first sight, and it has its advantages in many cases.

1.1 Getting help

The greeting message you see at the startup already gives you a few tips. Now type

```
> help()
```

including the parentheses but not the command prompt. In this tutorial we will follow this convention: the commands you should type will be displayed after the command prompt, '`> `'.

If you type the above command and press enter, R will present the built-in documentation about how to get help. Depending on the R configuration on your system, you may get the help text either on the same window, or R may present the help in another window. If you get help on the same window, you can scroll up and down using arrow keys or page up/down keys on your keyboard. Pressing '`q`' will quit help and give the command prompt back. As you were instructed at the greeting message, you could alternatively type

```
> help.start()
```

and get the documentation in an external browser. To get help on a particular command, for example `pnorm`, you can type

```
> help(pnorm)
```

but in case you do not remember the exact command, you can search a keyword in the documentation using `help.search()`. For example, if you were wondering what was the command that did Student's T test, you can try

```
> help.search('student')
```

R will list the help topics that match, and you can again use `help` to read the documentation. Two shortcuts you may appreciate if you use the help facility frequently are `?` and `??` which correspond to `help()` and `help.search()` respectively. When using `?` and `??`, you should just type the keyword(s) without parentheses. If the keyword contains white spaces, you need to use double or single quotes around it.

A tip that you may be happy to hear is that R remembers your previous commands. You can return to the previous commands using the *up* arrow key on your keyboard, navigate between them with *up* and *down* arrow keys, and you can modify and re-run them if you wish.

There are many small tips and tricks you will collect while working with R, one last tip to mention here is that R command line allows 'tab completion'. That is, if you type a unique initial segment of a command (or variable or file name in the right context), and press the 'tab' key on your keyboard, R will try to complete the rest for you. If there are more commands that match the initial string you typed, pressing 'tab' twice will list all matching commands.

Besides the documentation built in to R, the official web site of the R project contains the reference manual (R Core Team 2014), and many useful documents and pointers to other sources. If all else fail, you can ask your questions at one of the mailing lists (that you can find through the official R site), or sites like `http://stats.stackexchange.com/`. Before asking questions on online lists and groups, you should always make an effort to find the answer in obvious places.

1.2 Doing simple calculations with R

R can be used as a calculator. Try typing a few arithmetic expressions at R's prompt and check what happens. Listing 1 demonstrates some of the arithmetic operations.

The lines that do not start with a command prompt in Listing 1 are the outputs. In line 5, the multiplication operation takes precedence: it is calculated as `6 - 12`, not `3*4`. In line 7, to make sure that addition is done before division, we used parentheses. If you are familiar with usual operator precedence in programming languages, R will not surprise you. However, there is no harm in adding a couple of parentheses to make sure you get the result you want.

Listing 1: Simple calculations.

```
1 > 1 + 2
2 [1] 3
3 > 3 * 4
4 [1] 12
5 > 6 - 3*4
6 [1] -6
7 > 7/(6 + 2)
8 [1] 0.875
9 > 16^(1/2) # 0.5th power (sq. root) of 16
10 [1] 4
```

Another thing to note in this listing is that R regards any text after the hash sign (#) until the end of the line as a comment, and ignores it. Comments do not have much use during interactive use, but they come handy when you save command sequences (R scripts or programs) in files for future reference.

1.3 Variables

Under the hood, R provides a complete general purpose programming language (in fact R is an implementation of language *SPLUS*) which may be really handy if you have some programming background. In this set of exercises we will not go into programming. However, we will be using variables frequently.

Use of variables may save you from quite some typing, and R will save the values of variables on exit by default so that you can access the same values when you restart R.

To assign a value to a variable you can use the assignment operator, '=', (or, equivalently, <- as R experts do). And you can use the variables in calculations or if you type a variable name and press enter, R will report the value. Listing 2 demonstrates the basic use of variables.

Listing 2: Variable assignment.

```
1 > now = 2010
2 > birth.year <- 1988
3 > birth.month = "February"
4 > age = now - birth.year
5 > age
6 [1] 22
7 > now = now + 2
8 > Age = now - birth.year
9 > Age
10 [1] 24
11 > age
12 [1] 22
```

In line 1 we store the value 2010 in variable now (yes, now is relative). In line 2 we use the alternative assignment operator <-, this is equivalent to =. In this tutorial we use both somewhat randomly to remind you that you may see R code using both, and they are equivalent (see the answer of Exercise 1.3, for one more assignment operator).

In line 2 and 3 we use a dot '.' instead of space. R variable names cannot contain space characters, and dot is the conventional character instead of space in R community. There are more rules for variable names. For example, they cannot contain many other special characters (like -, +, /) and they cannot start with numbers.

Line 3 demonstrates use of character strings. Character strings must be enclosed in matching double (") or single (') quotes.

R supports a variety of operations on string type, and it may come quite handy while working with language data (e.g., corpora). Apart from numbers and strings there are other types that your variables can take. For example *boolean*s that take values TRUE or FALSE and *categorical* variables (or *factor* variables as R calls them) are interesting for many statistical tasks. We will return to discussion of these types later.

Line 4 subtracts value of birth.year from now and stores the result in a new variable age. As demonstrated in line 5, if we type the name of the variable R tells us the value stored in the variable.

Line 7 may be confusing for non-programmers. This line adds 2 to variable now, and re-assigns the new value to the same variable now. In other words, we increment now by 2.

In line 8, we (re)calculate the age, but beware: the case matters in variable names. Age is not the same as age. As a result we have two variables now, lowercase age still contains the previous calculation on line 4, and uppercase Age contains the calculation in line 8. The rest of the lines demonstrate this difference.

You should enter this command sequence in R to check if all works as in the listing.

If you'd like to see the user variables, you can use the function ls(), and if you want to get rid of one, for saving space, for keeping your environment clean and tidy or for any other reason, you can use rm().

1.4 Vectors in R

In statistics, we are generally interested in a sample, or a list of values. For that purpose, R offers a data structure called *vector*. Vectors in R are similar to arrays or lists in programming languages. The important thing to know is that a vector is a container of a set of values of the *same type*.

For the exercises in this section, we will use the following data. For a class, students are asked to submit a 3,500 to 4,000-word report. 10 students turned in the reports with the following word lengths:

```
3510,3508,3468,3520,3516,3525,3505,3519,3558,3487
```

To enter this data into a vector variable we type,

```
> nwords = c(3510,3508,3468,3520,3516,3525,
             3505,3519,3558,3487)
> nwords
 [1] 3510 3508 3468 3520 3516 3525 3505 3519
 [9] 3558 3487
```

This example demonstrates the primary way of assigning a vector to a variable. The function c (stands for concatenate), puts together its arguments into a vector. Like simple data types, if we type the name of the variable, we get its value displayed (in fact, the simple variables we have been working with are vectors containing single elements). Entering large datasets this way is, at best, cumbersome, and R provides other ways of entering data to which we will return later.

At this point you should type the above assignment command to create the vector nwords. We will use this data set in the next few sections.

R supports mathematical operations between vectors and the scalar values and vectors and vectors. Standard R functions that normally take a basic value can also take vectors as arguments, in which case the function is applied to all members of the vector.

Elements of a vector can be selected by specifying the position of the element(s) between square brackets after their name. For example, if we want to refer to the fourth element of vector `nwords`, `nwords[4]` (in fact, as we will see later, one can also select possibly discontinuous ranges of data with this notation).

Listing 3 demonstrates some of these operations.

Listing 3: Some vector operations.

```
> nwords2 = 2 * nwords
> nwords2
 [1] 7020 7016 6936 7040 7032 7050 7010 7038 7116
     6974
> nwords + nwords2
 [1] 10530 10524 10404 10560 10548 10575 10515
     10557
 [9] 10674 10461
> log(nwords + nwords2)
 [1] 9.261984 9.261414 9.249946 9.264829 9.263692
 [6] 9.266248 9.260558 9.264544 9.275566 9.255409
> nwords[1]
[1] 3510
> nwords[10]
[1] 3487
```

The first line multiplies a vector with a scalar value. In other words, all members of the vector is multiplied with `2`. Line 4, on the other hand, sums two vectors. Finally, in line 6, the function `log()` is applied to each member of the resulting vector.

Besides the arithmetic operations and scalar functions applied to vector elements, there are a set of functions that operate on vectors. Listing 4 demonstrates some of these functions. Note that the listing already includes a few statistical functions (finally we are getting closer to the point!).

Listing 4: More vector operations.

```
> length(nwords)
[1] 10
> sum(nwords)
[1] 35116
> min(nwords)
[1] 3468
> max(nwords)
[1] 3558
> head(nwords,2) # first two elements
[1] 3510 3508
> tail(nwords,3) # last three elements
[1] 3519 3558 3487
> sort(nwords)
 [1] 3468 3487 3505 3508 3510 3516 3519 3520 3525
     3558
> range(nwords)
[1] 3468 3558
> mean(nwords)
[1] 3511.6
> median(nwords)
[1] 3513
> summary(nwords)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3468    3506    3513    3512    3520    3558
```

Exercises

Exercise 1.1. You want to do a 'Multivariate ANOVA', but you do not know the exact command that does it. Use `help.search()` (or `??`) to find the name of the command. ▷

Exercise 1.2. The R function `shapiro.test()` implements well-known Shapiro-Wilk normality test.

1. How many of the initial characters you need type before R can complete the function name using tab completion?
2. How many R functions start with `sh`?

▷

Exercise 1.3. Perform the following actions in R:

1. store 20 in the variable `x`
2. store 10 in the variable `m`
3. store 5 in the variable `s`
4. subtract `m` from `x`, store the result in `t`
5. divide `t` by `s`, store the result in variable `z`

What is the value of the variable `z`? ▷

Exercise 1.4. Redo the calculations in Exercise 1.3 steps 4 and 5 without the use of the temporary variable `t`. Use parentheses if necessary to get the same result. ▷

Exercise 1.5. R supports a wide range of mathematical functions. A function that is often useful in statistical analysis is the *logarithm* function. Using the shortcut you learned in Section 1 search for the function name that returns logarithm of a given number, and use it to calculate logarithm of `2.7`. ▷

Exercise 1.6. You should have obtained a value close to `1` in Exercise 1.5. This is because of the fact that R calculates *natural logarithm* (base e=2.718282…) by default. Often we use base-`2` logarithm. The function you used above can be used to calculate base-`2` logarithm as well. Using the shortcut you learned in Section 1 read the help for the logarithm function to learn how to specify which base to use. Calculate base-`2` logarithm of `2.7`. ▷

Exercise 1.7. After completing the exercises in this section, you should have a set of variables that we will not use in the future. List the variables in your R session, and delete the variables `t` and `x` (If you wish, you can delete all variables except the vectors `nwords` and `nwords2`. We will not use the other variables in the rest of the tutorial.). List your variables again to see if you have achieved the desired result. ▷

Exercise 1.8. Remember that the *standard error of the mean* can be calculated using the formula

$$\frac{s}{\sqrt{n}}$$

where s is the (estimated) standard deviation, and n is the size of the sample. Calculate the standard error of the mean for the word count data stored in `nwords`. You can calculate the standard deviation using the function `sd()`. It is easy to just count the number of elements in `nwords`, but you can use the `length()` function to get the number of elements in a vector. ▷

Exercise 1.9. In Exercise 1.3 you calculated z-score for a single value using pre-specified mean and standard deviation. More formally, z-score is calculated with the following formula:

$$z = \frac{x - \mu}{\sigma}$$

Calculate z-scores of the values in the vector `nwords`, and assign it to a new vector variable named `znwords`. Display the resulting vector, its mean and the standard deviation. ▷

Exercise 1.10. In a (hypothetical) country, four political parties got 36,35,8 and 71 seats in the parliament with 150 seats. Sort the numbers of seats in reverse order, with the largest element first and the smallest as last. ▷

Exercise 1.11. Use the seat counts in Exercise 1.10 to calculate the percentages of seats for each party. Use a single expression, and do not hard code the number of seats in the parliament into your expression. You may want to store the data in a variable for convenience. ▷

Exercise 1.12. You realized that the word counts we used in this section included the essay title and the student's name by mistake. For each essay, we would like to discount word counts by `6,8,6,5,7,5,9,7,10,9`. Store these values in a new vector variable `wdiff`. ▷

Exercise 1.13. Create a backup copy of the data stored in the vector `nwords` in the vector `nwords2` (this may sound like serious work, but you can simply use the assignment operator). Subtract the vector `wdiff` from the vector `nwords` and store the result again in the vector `nwords`. Display the contents of `nwords` and `nwords2`. ▷

Exercise 1.14. Find the differences of *mean*s of the values stored in `nwords2` and `nwords`. Is it the same as the mean of the vector `wdiff`? (In other words is the difference of the means the mean of the differences?) ▷

2 Basic data exploration and inference

This section provides a brief glance at some of the basic summarization, visualization and inference techniques. We will return to most of the tools and topics introduced here in the rest of the tutorial.

In this section we will use data from a *hypothetical* experiment where the number of words spoken per day by 10 female and 10 male English speakers were measured. Note, again, that the data we will use is fictional, for a real investigating of the problem see Mehl et al. 2007.

The following R commands create four vector variables.

```
words.f <- c(17667, 15347, 14401, 5037, 20845,
             11211, 6008, 17140, 13284, 10930)
words.m <- c(5599, 19776, 13961, 10144, 6107,
             16776, 31955, 21140, 5482, 2152)
words <- c(words.f, words.m)
age <- c(24, 31, 28, 21, 29, 29, 25, 32, 30, 31,
         33, 26, 22, 24, 23, 23, 20, 21, 29, 27)
```

The vectors `words.f` and `word.m` hold the number of words measured from female and male participants, respectively. The vector `words` holds their combination, and `age` contains the ages of the participants. Organizing this data in separate vector variables is not what we normally do. We will later use *data frames* for storing a related set of vectors such as the ones above.

2.1 Summarizing and visualizing one-dimensional data

Even for a data set as simple/small as `words` above, it is difficult draw conclusions only by looking at the raw data. We generally want to summarize the data at hand to understand it better. In Listing 4, we have already seen how to get some useful summaries in R.

For one dimensional data, here are a few functions that produce common summaries:

- `mean()` mean of the given data set.
- `median()` median of the given data set.
- `min()` minimum value.
- `max()` maximum value.
- `summary()` So-called *5-point summary* (minimum, lower quartile, median, upper quartile and maximum) and mean.
- `sd()` standard deviation.
- `mad()` maximum absolute deviation from the median.

You are encouraged to try these functions (again) on `words` data set.

These summaries are often useful, indicating the center and spread of the data at hand. However, we often want to understand the data in more detail. In that case graphical summaries are more helpful.

One of the ways to visualize small data sets is stem-and-leaf plots. Stem-and-leaf plots can be produced with the function `stem()` in R.

Exercise 2.1. Produce a stem-and-leaf plot of `words`. Does the plot indicate an outlier? Can you say more about the distribution of the word counts? ▷

One of the better ways of inspecting your data is producing a *histogram*. You can display a histogram in R using `hist()` function.

Exercise 2.2. Produce a histogram of `words`. ▷

Yet another method of visualizing your data is plotting *box-and-whisker plots* (or box plots). Box plots display a summary similar to the five-point-summary in a graphical way. In a box plot, the box covers the range between first and third quartile (the interquartile range, IQR). The middle bar represents the median. The whiskers extend 1.5 interquartile range from the box, or up to the maximum or minimum values if they are within this range. The data points more extreme than the whiskers are considered outliers and plotted separately.

Exercise 2.3. Display a box plot of `words`. ▷

Exercise 2.4. Normally, box plots are more useful for comparing two or more samples, or groups. Plot box plots for `words.f`, and `words.m` side by side. Do women talk more than men? **TIP:** you can use `help()` if you do not know how to use `boxplot()` to display two groups instead of one. ▷

2.2 Summarizing and visualizing two-dimensional data

The basic measure of relatedness of two sets of continuous variables is their *correlation*. The correlation of two can be calculated using the function `cor()`.

Exercise 2.5. Find the correlation between `words` and `age`. Do people speak more as they get older? Is this a strong correlation? ▷

Exercise 2.6. Do you expect any correlation between the variables `words.f` and `words.m`? Calculate the correlation coefficient between these two variables. Can you explain your findings? ▷

The function `cor()` by default calculates the *Pearson product-moment correlation coefficient*, known as *Pearson's* r. We will return to the topic of correlation later, and discuss interpretation and inference of it in more detail.

To visualize the relationship between two numeric variables, we can use *scatter plots*. Given two vector variables of the same size, the function `plot()` creates a scatter plot.

Exercise 2.7. Create a scatter plot for visualizing relationship between `words` and `age`. Which variable should be plotted along the x-axis? ▷

The relationship between two variables can also be summarized and visualized by a straight line. The equation for a straight line is

$$y = a + bx$$

This equation forms the basis for nearly all statistical methods in modern science. y and x in this equation are the variables of interest, and a and b are called *intercept* and *slope*. The standard method of estimating such a line that fits the data is called *least squares regression*. To estimate a least squares regression line from related sets of data points, we use

```
lm(y ~ x)
```

You should note that unlike correlation, regression is asymmetric (`lm(y~x)` and `lm(x~y)` will produce different results). We put our response variable (outcome or dependent variable) before the tilde '`~`', and the predictor (explanatory or independent variable) on the right side. Similar to the sign of the correlation coefficient, the sign of the slope indicates the direction of the relationship. The magnitude of the slope indicates magnitude of the effect of the predictor on the response variable. Here is how we fit a regression line that reflects the effect of the age on number of words spoken per day:

```
> lm(words ~ age)
Call:
lm(formula = words ~ age)
Coefficients:
(Intercept)          age
    25610.1       -468.3
```

The output indicates that the intercept is 25610.1 and slope is −468.3. In other words the fitted regression line can be expressed as

$$\mathit{words} = 25610.1 - 468.3 \times \mathit{age}$$

Like in many other cases we will study, there is no meaningful interpretation of the intercept (according to this equation one is expected to speak 25610.1 words per day at age 0). The slope indicates that we expect 468.3 fewer words spoken per day with every year of age. To arrive at these conclusions we need to make sure that the 'model' above meets certain criteria.

Exercise 2.8. The command

```
abline(lm(words ~ age))
```

plots least-squares regression line over the existing graph. Plot the regression line. Does the regression line agree with the correlation coefficient you have calculated earlier? ▷

Exercise 2.9. Create a scatter plot of `words.f` against `words.m`, and also draw the corresponding regression line. ▷

Note that this section includes a rather quick and dirty introduction to correlation and regression as exploratory/descriptive tools. Both topics will be revisited in more detail later. Similarly, the above graphs are a very first introduction to making graphics in R. We will explore the graphical capabilities in R as we go, and dedicate a special section for producing informative and pretty graphics.

2.3 Simple inference

The summaries and graphs we discussed in this section so far helps us understand the data at hand better. Often, our questions are not about the particular sample we have at hand. We would like to know whether 'women talks more than men' *in general*, not only in this particular sample. We use our sample to estimate some quantities of the population that the sample comes from, e.g., mean number of words spoken per day for all humans. Naturally, we do not expect two samples taken from the same population to be exactly the same, and our estimation will include some uncertainty due to not having all the information about the population. Inferential statistics is about quantifying this uncertainty and making sure that the estimates we have reflects the population values within certain bounds.

It is worth to mention a very important aspect of statistical analysis here: the sample you took should be representative for the population you are interested to study. No statistical technique can fix the effects of wrong sampling. For example, for our 'words per day' example, you cannot generalize anything about the number of words spoken per day by very young and very old people, nor people speaking another language.

The simplest inference we can make is about the mean. The *standard error of the mean* (SE) we have calculated in Exercise 1.8 is an important quantity for assessing the uncertainty of the mean value estimated from a sample.

Exercise 2.10. Calculate the SE for the complete `words` data set. Store the result in a variable and display it. ▷

In practice, it is more common to report *confidence intervals*. 95% confidence intervals are the most commonly reported ones. A quick way of calculate approximate 95% confidence intervals that works fine for large samples is $\hat{\mu} \pm 2 \times \mathrm{SE}$ where $\hat{\mu}$ stands for the estimated mean.

Exercise 2.11. Calculate the 95% confidence interval for `words` data set with the above approximation. ▷

If the population standard deviation is known, or if the data set is large, we can use *normal distribution* to calculate the

confidence intervals. For 95% confidence intervals we need to know the lower 2.5% and upper 97.5% quantiles. These values can be looked up in tables that most statistics textbooks include. Or, easier, can be calculated using the `qnorm()` function in R. The number 2 used in the above approximation is based on the fact that these quantiles fall between ± 1.96 standard deviations away from the mean for the normal distribution. In most cases, we estimate the population standard deviation from the sample, to correct for the uncertainty introduced by this, we use *t distribution* with $n - 1$ degrees of freedom. Similarly, we can use the function `qt()` for this purpose. For example, `qt(0.025, 9)` will give you the value corresponding to the lower 2.5% quantile for the t distribution with 9 degrees of freedom.

Exercise 2.12. Calculate the 95% confidence interval for `words` data set using values obtained with `qnorm()` and `qt()`. Compare the results to each other and the values obtained with the approximate calculation above. ▷

Confidence intervals are related to the classical hypothesis testing. If a particular value does not fall into the 95% confidence interval, we can reject the null hypothesis that the mean is equal to this value. For example if we calculated a 95% confidence interval of [10000, 16000], we can reject the (null) hypothesis that the mean of the population is 20000, at the significance level of 0.05.

Exercise 2.13. Assume that it is known that average number of words spoken by a Dutch speaker per day is 16000. Using confidence intervals you have calculated above on `words`, test the alternative hypothesis that the number of words spoken per day is different for Dutch and English speakers. ▷

The standard way of testing hypothesis like the one above is to perform a *t test*, in this particular setting, a *one-sample t test*. The function `t.test()` in R performs one- and two-sample t tests.

Exercise 2.14. Perform the same test in Exercise 2.13 using the function `t.test()`. **TIP:** see built-in help for specification of the null hypothesis. ▷

Exercise 2.15. A popular book claims that, on average, women speak 20000 words per day. Can you reject this hypothesis using the data (`words.f`) above? ▷

Exercise 2.16. Finally: do women speak more than men?

Try to answer this question using the data sets in `words.f` and `words.m`. Note that our hypothesis is directional, hence, you should use a one-sided test.

What is your conclusion based on the R output? ▷

3 Linear regression: a first introduction

This section will introduce the simple linear regression briefly. We will return to the topic in almost all of the sections that follow.

The typical application of linear regression is when you have a continuous outcome with continuous predictor(s). In this section, we will consider the case with only one predictor. First, we describe the data that we will exercise with.

During language acquisition, it is claimed that caregivers adapt their language to the language abilities of children. We follow a particular child and record an hour conversation between the child and the mother once every month between the child's second and fourth birth dates. We calculate a well-known measure of language complexity/competency for children Mean Length of Utterance (MLU), for the child and her mother for each recording session. The data is real (from the CHILDES database). However, it is still a 'toy' data set, and you should be careful not to make generalizations based on this data.

Here is how to create our data set (you can copy & paste):

```
mlu <- data.frame(
    age=seq(25,48),
    chi=c(1.46, 1.41, 1.66, 1.74, 1.90, 1.91,
        1.85, 2.06,
          2.27, 2.43, 2.70, 2.81, 2.69, 2.72,
              2.64, 3.05,
          3.22, 3.42, 3.70, 3.90, 3.57, 3.49,
              3.66, 3.64),
    mot=c(5.42, 5.69, 6.27, 6.10, 6.06, 5.98,
        6.10, 6.09,
          6.10, 6.14, 6.42, 6.35, 6.21, 6.07,
              5.84, 6.17,
          5.74, 6.11, 6.41, 5.50, 6.00, 6.90,
              6.65, 6.40)
)
```

This time we have created three vectors, and wrapped it into a *data frame*. Data frames are the data structure we will use most of the time, although, we will rarely create them by hand as we did above. The way we specified the `chi` and `mot` variables (vectors) should be familiar. We have created `age` using `seq()` which returns a vector of integer values between 25 and 48 in this example (the age of the child in months). The resulting data frame will have corresponding values of each variable (or vector) on the same row.

The following listing gives some examples of how to access individual columns, rows and items in a data frame.

```
> mlu$age
 [1] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
[16] 40 41 42 43 44 45 46 47 48
> mlu[2,2]
 [1] 1.41
> mlu[,1]
 [1] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
[16] 40 41 42 43 44 45 46 47 48
> mlu[2,]
  age  chi  mot
2  26 1.41 5.69
> mlu$chi[2]
[1] 1.41
```

In a nutshell, you can extract certain variables (or columns) from a data frame using the `$` notation.In essence you can select any individual cell with the notation `mlu[r,c]`, where `r` refers to the row number and `c` refers to the column number. If you leave either column or the row number unspecified, you get the complete column, or the row respectively (note, you keep the comma '`,`' in both cases). You can also mix and match the dollar-notation and the vector indexing.

You should study each line, and make sure you understand what it means. We will exercise with more complex ways to access and manipulate data in data frames. However, the basics above will be used rather frequently.

3.1 Some preliminaries

The basic way of visualizing two related sets of numerical data is to plot them using a scatter plot.

Exercise 3.1. Plot a scatter plot for visualizing the effect of the child's MLU on the mother's MLU. Remember that we put the predictor (or independent variable) on the x-axis. ▷

In simple linear regression, our aim is to find the best linear equation that fits the data points. The general form of the equation is

$$y_i = a + bx_i + \epsilon_i$$

where y and x in this equation are the variables of interest, the index i ranges over all observations, and a and b are called *coefficients*, or individually they are called *intercept* and *slope* respectively. The term ϵ reflects the fact that our best estimate of a and b will not result in perfect prediction of y_i from x_i for every observation. In other words, ϵ_i is the error made by the model for observation i. Our best estimate is the one that makes the least error (for some definition of 'least error'). We will come back to the proper estimation of the regression equation, shortly after some exercises with drawing lines in R.

We already used the function `abline()` to draw the line estimated by `lm()`. If you give two numeric arguments to the `abline()` function (instead of the `lm()` result), it takes the first argument as the intercept (a), and the second one as the slope (b) and draws the corresponding line (hence, the name '$a$bline'). For example `abline(0,1)` will draw a line with intercept 0 and slope 1 (a line that passes from the origin with a 45 degrees of slope).

Exercise 3.2. Using the scatter plot you have produced in Exercise 3.1, try to draw the line that you think fits the data the best. **Do not** use `lm()` to estimate the line yet, draw multiple straight lines using `abline()` until you are convinced that you have the best line. ▷

Exercise 3.3. Many plotting commands in R accept a '`col`' argument for specifying the color of the objects drawn. Similarly, you can specify the width of a line using the argument '`lwd`'. Redraw the best-fitting line from Exercise 3.2. Make sure the line is 'red' and it is twice as thick as the standard lines. ▷

We already know that the function `lm()` in R finds the best line using the *least squares regression*. In fact, this function can estimate any *general linear model*, and we will use it in most of the sections that follow for performing different analyses. First here is a simple call to `lm()` (the output is slightly edited to save space):

```
> lm(mlu$mot ~ mlu$chi)
Call: lm(formula = mlu$mot ~ mlu$chi)
Coefficients:
(Intercept)      mlu$chi
     5.7133       0.1503
```

The estimated intercept is 5.7133 and the slope is 0.1503. The intercept represents expected value of the mother's MLU when the child's MLU is 0. Although this is a reasonable quantity to predict, our sample would not allow us to predict it reliably (why?). On the other hand the slope tells us that for every unit increase in child's MLU, we expect mother's MLU to increase by 0.15.

Exercise 3.4. Draw the estimated regression line in color blue over the ones you have drawn in Exercise 3.2 and Exercise 3.3. Compare it with the line you estimated. ▷

The estimated regression line tells what we found in our data. However, it does not tell anything about generalizability of these results outside our sample. The `lm()` function does more than what we see when we just run it. It returns an R object that we can investigate further, and inferential question we have just raised can be answered by asking for a `summary()`, as shown in Listing 5.

Listing 5: Summary of a linear regression fit.

```
1 > m <- lm(mlu$mot ~ mlu$chi)
2 > summary(m)
3 Call: lm(formula = mlu$mot ~ mlu$chi)
4 Residuals:
5      Min       1Q   Median       3Q      Max
6 -0.79928 -0.14665  0.06142  0.14003  0.66232
7 Coefficients:
8             Estimate Std. Error t value Pr(>|t|)
9 (Intercept)   5.7133     0.2326  24.559   <2e-16
10 mlu$chi      0.1503     0.0839   1.791   0.0871
11
12 Residual standard error: 0.3182 on 22 degrees of
       freedom
13 Multiple R-squared:  0.1272,    Adjusted
       R-squared:  0.08757
14 F-statistic: 3.207 on 1 and 22 DF,  p-value:
       0.08708
```

First, instead of using the `lm()` output directly, we save the 'model object' returned in a variable. In line 2, we get the summary of the model using the variable '`m`'. The same could be achieved with the command '`summary(lm(mlu$mot ~ mlu$chi))`' without storing the intermediate result. The first line in the output above reminds us the way we run `lm()`. The lines 4–6 present the 5-point-summary for the residuals, the ϵ in our formula above. For now, we skip these, but it will soon be clear why this is important for interpreting linear regression results. Next, lines 8–10 present the estimated coefficients (intercept and the slope) along with some inferential statistics about them. The standard error presented is similar to the standard error of the mean we have discussed earlier. It represents the standard deviation of the coefficient estimates that would be obtained from similar samples. The t-tests presented have the null hypothesis that the coefficient tested is 0. For the intercept, this test is not quite useful. The other problems regarding estimation and interpretation of the intercept in this problem aside, we do not really think that mothers start speaking to their children only when their children start talking. The inference for the slope is generally something we are interested. Remember that the slope represents the unit increase in the mother's MLU, given the child's MLU. In other words, this is the effect of the child's MLU on the mother's MLU. If we cannot reject the null hypothesis that the slope is 0 (=no effect), we cannot be certain that the effect we estimated is not a chance effect.

The lines 12–14 present further statistics that are useful in our interpretation of the linear regression results. We will return to these later. For now, we note that the reported '`R-squared`' value is the standardized effect size for linear regression, and interpreted as the 'amount of variance in the response variable that is explained by the predictor(s)'.

Exercise 3.5. Find the Pearson correlation coefficient between

the mother's and the child's MLU. Compare square of the correlation coefficient with the `R-squared` reported in Listing 5. ▷

3.2 Some model diagnostics

The `lm()` function in R will find the regression equation with the minimum sum of squared errors ($\sum_i \epsilon_i^2$). However, the results may be irrelevant if the following modeling assumptions are not checked.

- Observations, or equivalently residuals (ϵ) are independent.
- Residuals should be normally distributed with 0 mean.
- The residual variance is constant.

Besides these, the least squares regression estimation is sensitive to extreme values or outliers.

We will return to all these assumptions, and how to check them later. For now, we will only check the residuals for normality, as we already know how to do it. Note that almost all assumptions of the linear regression is about residuals. Now you know why the summary presented in Listing 5 includes a five-point-summary of residuals. To extract residuals from a model you can use the `resid()` function.

Exercise 3.6. Extract residuals from the model fitted in Listing 5,

- visualize the residuals using a histogram,
- Check normality of the residuals using a normal quantile-quantile plot. A normal Q-Q plot can be used to visually check whether a given sample follows the normal distribution or not. The R function `qqnorm()` produces a normal Q-Q plot, and `qqline()` plots the theoretical normal line.

▷

Exercise 3.7. We wonder whether MLU is a good measure of a child's language ability. For a normally developing child, we expect the age of the child to be a good predictor of his/her language ability. Using `age` as a proxy to child's language ability, investigate the relation between the MLU and the language ability.

1. What is the best choice of predictor and response variables for this problem?
2. What is the estimates of intercept and the slope, and how do you interpret them?
3. Produce a scatter plot of the data, and draw the regression line.
4. Is the slope estimated statistically significant at level 0.05?
5. Do you observe any clear outliers in the scatter plot?
6. Extract residuals form the model, and check whether they are normally distributed or not.

▷

4 Linear models with categorical predictors

In this set of exercises we will study methods to analyze data where the response variable is continuous (e.g., pitch, duration, reaction time), the predictor is categorical (e.g., gender, language, part-of-speech tag). Crucially, all methods we discuss here works with independent samples (e.g., no multiple measurements from the same individual). Analyzing 'repeated measures' or non-independent data will be addressed in the next section.

In Section 2, we have already seen an example where we tried to see whether 'talkativeness' can be predicted by gender, and analyzed the data using the t test.

For the first part of this section, we revisit the t-test exercises from Section 2, so you will need the `words` data set used there. If you do not have the variables from Section 2 in your R environment (remember, you can check this using `ls()`), use the commands given at the beginning of the Section 2 to create it.

Now that we know that the related sets of variables should be stored in data frames instead of individual vector variables, we would like to put these variables together in a data frame.

Exercise 4.1. Create a data frame containing only the variables `words` and `age`. Name your data frame '`talk`'. ▷

The `talk` data frame does not yet contain the information on the genders of the participants. From the way we constructed the `words` vector earlier, we know that the first 10 values correspond to the female participants, and the last 10 values correspond to the male participants. In a properly constructed data frame, we typically do not want unrelated values on a row. So, a data frame that contains word counts for male and female users in different columns is not a good option (although this data format, called 'wide format', is used by some other statistical software by default). Instead, we will keep the data frame as created in Exercise 4.1, and add a new column with the categorical variable `gender`.

Like numeric data types, R supports a 'factor' data type that is suitable for categorical or nominal data (which can optionally be ordinal, but this is not a concern for us in this section). Here is how to add the `gender` column to the `talk` data frame.

```
> talk$gender <- factor(c(rep('F', 10), rep('M',
    10)))
> talk$gender
 [1] F F F F F F F F F F M M M M M M M M M M
Levels: F M
```

The first line, has two new R functions that we haven't seen yet. `rep()` returns a vector where its first argument repeated as many times as its second argument. As we have seen before, `c()` concatenates the results. And, `factor()` makes sure that the resulting vector contains *factors*. If we display it as in the second line of the listing above, R will also give you the possible values 'levels' the factor variable can take.

We will return to some of the details regarding the factor variables. For now, it suffices to understand that we have a factor variable `gender` with values `F` for the values in `words` that were collected from the female participants, and `M` for male participants.

Besides the `talk` data we reorganized, we will use another *hypothetical* set of data where we investigate early language development of children, given their parents' socio economic status (SES). Our fake data set include *mean length utterance*

(MLU) values from 10 girls and 10 boys for each three levels of SES we consider (low, mid, high). You can load the data with the following command.

```
> mlu.ses <- read.csv(
   'http://coltekin.net/cagri/R/data/mlu-ses.csv')
> mlu.ses$ses <- factor(
    mlu.ses$ses, levels=c('low', 'medium', 'high'))
```

If you are working offline, you should use the local file name instead of the URL in the first command. Note that in R strings backslash '`\`' has a special function. When using local path names on Windows, you should either use double backslash (like '`C:\\My files\\R\\mlu-ses.csv`') in path names, or use forward slash '`/`' instead. The second command in the above listing corrects the ordering of the SES levels (and yes, we will return to this one too).

If all goes fine, you will have a *data frame* with four variables (`subject`, `ses`, `gender`, `mlu`) and 60 rows (observations, cases). Note that the `talk` data frame we have created earlier also have variables with the same name. For each exercises that follows, we will tell explicitly which data to use.

Exercise 4.2. The function `str()` prints a readable summary of a variable (an R object). Inspect the data frame using `str()`. Does the summary match what you expect from the description above? ▷

Exercise 4.3. You should have seen in Exercise 4.2 that R took the variable `subject` (which corresponds to the arbitrary ID of participant in the study) as a numeric value. Convert `subject` variable in `mlu.ses` to a factor variable. ▷

Earlier, we saw how to extract the value of a particular cell, column or row from a data frame. Often, we want to extract values that meet a certain condition. The row and column and indices that we used earlier can contain arbitrary conditional expressions. For example, the following listing demonstrates how to extract all rows where `mlu` is greater than four and `ses` is `high`.

```
> mlu.ses[mlu.ses$mlu > 4 & mlu.ses$ses == 'high',]
   subject  ses gender  mlu
54      54 high female 4.20
59      59 high female 4.62
```

Note that equality in a comparison is expressed with double equal signs '`==`' and the sign '`&`' means boolean 'and' operation. Similarly some common operators you can use are,

- `!=` not equal to
- `>=` greater than or equal to
- `<=` less than or equal to
- `|` boolean 'or'

Exercise 4.4. Choose only the `mlu` values from `mlu.ses` where the `gender` is `male`. ▷

4.1 Comparing two means

The classical method for comparing two means is the *Student's t test* that we have experimented with in the previous section. Here we will repeat some of the earlier t-test exercises in a slightly differently way. An alternative notation for running the t test in Exercise 2.16 is The output will be the same. The

Listing 6: T test with formula notation.

```
> t.test(talk$words ~ talk$gender,
    alternative='greater')
```

reason for this repetition is to familiarize you with the categorical variables and the formula notation. Both of these concepts will be used throughout this section, and for the rest of the tutorial. Note that you need to prefix the variable names with '`talk$`' to specify which data you are referring to. If you do not what to type the data frame name you can use the command

```
> attach(talk)
```

after which, the columns in `talk` data frame will be accessible from your default R environment without referring to the data frame name. You can `detach()` if you want to go back to the state before the `attach()` command.

The notation `words ~ gender` can be translated into plain words as '`words` is explained by `gender`', or more technically '`words` is the response variable, and `gender` is the predictor'. We will use this notation frequently, and Appendix B explains the formula notation in some detail.

4.2 Checking assumptions of t test and ANOVA

All statistical methods (or models) come with a set of assumptions. For independent-samples t test and the ANOVA that we will work on shortly, there are three important assumptions:

- Observations are independent.
- The data in each group follow an (approximately) normal distribution.
- The variances for each group is (approximately) the same.

The first assumption above is related to how you collect your data (or conduct your experiments). We will discuss methods for analyzing non-independent data later. For now we will work with a few graphical and analytic tools for investigating the last two assumptions.

Exercise 4.5. One of the ways of visualizing the distribution of a set of data points is to plot a histogram. Plot necessary histograms to inspect whether the `talk` data meets the normality assumption required by the t test in Listing 6. Note that you will need to check both subsets. ▷

Exercise 4.6. Use normal Q-Q plots to inspect whether the data meets the normality requirement. ▷

A non-graphical method of testing for normality is 'Shapiro-Wilk test of normality'. The R function `shapiro.test()` performs this test.

Exercise 4.7. Verify whether Shapiro-Wilk test of normality agrees with your earlier judgments from the histograms and Q-Q plots. ▷

By default, `t.test()` in R will apply a correction for the unequal variances. This is the reason why earlier t tests we run were reported as 'Welch Two Sample t-test'. If you want to override this behavior you can pass the option `var.equal=TRUE` to `t.test()`.

Exercise 4.8. Run an independent samples t test to test the alternative hypothesis that women talk more than man, using the `words` data set as before, but *instruct `t.test()` to assume equal variances, and use the formula notation*.

Do the t-test results with or without corrections differ substantially?

Can you reason about the change in the 'degrees of freedom' reported by the t tests with and without the Welch correction? ▷

Exercise 4.9. A good way of visualizing whether variances of two (or more) samples are equal is to use box plots. We have already produced the plot necessary in Exercise 2.4. Produce the same box plot, but use the formula notation this time. Do the variances of male and female subsets look equal? ▷

Exercise 4.10. Another (non-visual) way of checking equality of variances is to use `var.test()` (which basically performs an F-test). Do the variances of male and female subsets differ according to `var.test()`? ▷

We have already seen that `t.test()` includes a correction for the non-homogeneity of variances by default. On the other hand, if divergence from normality is a concern, then we prefer a non-parametric test. The non-parametric alternative of independent two-samples t test is called *Mann-Whitney test* (equivalently, *Wilcoxon rank sum test*).

Exercise 4.11. In R, the non-parametric alternatives of all flavors of the t test are performed using the function `wilcox.test()`. Perform a Mann-Whitney test, and compare the results with the results from the t tests performed earlier. ▷

4.3 Single ANOVA

If we have more than two levels or 'groups' in our predictor variable, the classical test to perform is the single, or one-way, ANOVA (remember that in this section we only work with independent samples). Traditionally ANOVA and linear regression would be considered very different analyses. In this subsection and the next, we will follow this approach and study these methods as 'hypothesis testing' methods. However, single and multiple ANOVA we discuss in this section, and the linear regression are, in fact, part of a general framework called *general linear models*. We will make a first attempt to demonstrate this connection in Section 4.6.

To demonstrate ANOVA in R, we try to investigate the effect of *socio-economic status* (SES) on children's *mean length utterance* (MLU) using the hypothetical data set introduced at the beginning of this section. Before doing the actual analysis we will inspect our data and check whether the data meets the requirements of ANOVA or not.

Exercise 4.12. Generate side-by-side box plots of MLU for each level of SES.

Remember that you can access the individual columns (variables) in a data frame using the dollar '`$`' operator. For example `mlu.ses$mlu` will give you the `mlu` column of the data frame. Alternatively, you can give the option `data=mlu.ses` to `boxplot()` to and use the column names directly.

Based on the box plots,

1. Do the variances look similar?
2. Can you see any clear divergences from normality?
3. Do you expect any significant differences in MLU due to SES?

▷

Exercise 4.13. Check normality of each group using graphical methods, and `shapiro.test()`. ▷

All (independent samples) linear models can be fit using the function `lm()` that was introduced briefly in Section 3. Our use here will differ in two ways. First, our predictor is a categorical variable, and second, we summarize the model fit by `lm()` in a different way. Listing 7 shows how to do a single ANOVA investigating the effect of `ses` on `mlu`. We use the `lm()` function to fit a general linear model, and summarize it using the function `summary.aov()` which prints out the relevant information for a classical ANOVA.

Listing 7: Single ANOVA.

```
1 > summary.aov(lm(mlu~ses, data=mlu.ses))
2             Df Sum Sq Mean Sq F value   Pr(>F)
3 ses          2  20.71  10.357   21.07 1.41e-07 ***
4 Residuals   57  28.02   0.492
```

If you remember ANOVA from your basic statistics course, the output requires little explanation. The sums of squared differences, and 'mean squared difference' you see on the third line represent the variation between `ses` groups. Degrees of freedom are 2 since we have three groups. The quantities in the fourth line are due to 'within group' or 'error' variation. Degrees of freedom value is 57 since we have 60 data points and three groups. The F-value is the result of dividing 'Mean Sq' `ses` to 'Mean Sq' `Residuals`, which results in a very small p value, more precisely $1.41 \times 10^{-7} = 0.000000141$. Note that R typically uses the scientific notation when presenting very small numbers. If you are convinced that ANOVA is the correct analysis to run with this data, the result tells us that SES plays a role on MLU of a three-year-old child, and this role is (very) unlikely to be due to chance. Note, however, whether this effect has a practical importance is another question that needs some further information than the p-value, and also dependent on the particular problem studied.

Exercise 4.14. When you have only two groups, ANOVA is equivalent to the t test. Using the data frame `talk`, analyze the effect of `gender` on `words` using ANOVA. Compare your ANOVA results with the equivalent t test results from Section 4.1,.

Can you test 'directional' hypotheses with ANOVA as we did earlier with the t-test? ▷

At this point, you may wonder why do ANOVA at all, or why not perform multiple t-tests instead? One of the answers to this question is that you may not have a specific hypothesis about differences in individual groups (e.g., 'low' SES and 'medium' SES differ), but you may expect that the categorical predictor has an overall effect. This is probably reasonable for our example analysis of the effect of SES on MLU.

The second reason have to do with the logic of hypothesis testing in general. In an exploratory study looking for differences,

every pairwise difference you test will increase the probability of finding a difference by chance. As the number of groups increases, the number of possible (pairwise) comparisons increase. So, your chances of getting a significant difference where there isn't one increases. That said, if you have a set of specific hypotheses, you should go ahead and test them. The concern is valid when you would take 'any' pairwise difference interesting.

For an amusing story of multiple comparisons causing false positives, see `http://www.wired.com/wiredscience/2009/09/fmrisalmon/`, which reports on an fMRI study that finds "significant evidence" that a dead salmon shows emotional responses. For another fun but less 'scientific' demonstration, see `http://xkcd.com/882/`.

If you want to explore the data for potential pairwise differences and also want to perform valid hypothesis tests, then you should apply appropriate corrections (typically by lowering your significance level).

There are a number of different corrections that differ in their assumptions. Some of these corrections are implemented by the R function `pairwise.t.test()`.

Exercise 4.15. Performs all pairwise SES comparisons of MLU values in the data set `mlu.ses` using `pairwise.t.test()`. Note that `pairwise.t.test()` uses a different correction method than well-known Bonferroni correction. However, it can do Bonferroni correction if the appropriate options are given. Also perform pairwise comparisons with Bonferroni correction, and compare the results with the R default.

Which correction is more conservative? ▷

4.4 Factorial ANOVA

Often there are multiple categorical predictors that may affect the response variable of interest. In such cases, it is more economical (in terms of effort spent on experimentation or data collection) to perform a combined analysis. At least as importantly, analyzing multiple effects together also allows investigating their interactions. The extension of ANOVA with multiple groupings of the response variable is called *factorial*, or *n-way*, ANOVA.

The way to run factorial ANOVA is not much different than the single ANOVA example in Listing 7. Only difference is in the model specification, or the way we write the model formula. We will first start with an example where we ignore possible interactions. To specify a factorial ANOVA to investigate the independent effects of `ses` and `gender` on `mlu`, the formula to be used is '`mlu ~ ses + gender`'. Note that you should not read the sign '`+`' here as an arithmetic operation.

Exercise 4.16. Perform a factorial ANOVA that investigates the effects of `ses` and `gender` on `mlu`. Compare your results with the single ANOVA results in Listing 7. ▷

If we also want to include the interactions our formula becomes '`mlu ~ ses * gender`' (with a `*` instead of `+`. Again, you should refrain yourself from reading this as arithmetic multiplication).

Exercise 4.17. Perform a factorial ANOVA to investigate the effects of `ses` and `gender` and their interactions on `mlu`.
▷

The conventional visualization of interactions between two categorical variables can be displayed using the function `interaction.plot()`. Interaction plot puts one of the factors (categorical variables) on x-axis and the response (numeric) variable on the y-axis. It plots a dot for the mean of the response variable for all combinations of both factor variables, and draws a line between all dots that have the same level of the second factor. As a result, you need to give `interaction.plot()` two factor variables, and the response variable as arguments. It will calculate means of the each 'cell' from the data plot according tot the description above.

Exercise 4.18. Plot the interactions of `ses` and `gender`, and interpret the resulting graph with reference to the findings in Exercise 4.17. ▷

4.5 If ANOVA assumptions are not met

If normality and homogeneity of variances assumptions of ANOVA is violated with the data at hand, you typically have two sensible ways to go

- transform your data.
- use the non-parametric alternative, the Kruskal-Wallis rank sum test (for single ANOVA).

Finding correct transformation is not always easy. We will work with some sensible transformations later. For complicated methods, including most configurations of factorial ANOVA, there aren't straightforward non-parametric alternatives. As a result, trying to tame the data (e.g., by transformations) might be the only option at hand.

The function to perform Kruskal-Wallis rank sum test in R is called `kruskal.test()`. Kruskal-Wallis test does not assume normality, however, you should be aware that it requires the samples to come from identically-shaped distributions, which is not true if the variances are not similar.

Exercise 4.19. Perform the alternative non-parametric test for the ANOVA presented in Listing 7. Is the test more conservative or less sensitive than ANOVA? ▷

4.6 T-test as a linear model

You have been hinted along the way that there is a strong connection between the t-test (and also ANOVA) and the linear regression. In this subsection, we will try to demonstrate this connection with the t test, and extend it to ANOVA(s) later.

Exercise 4.20. Perform (yet another) the t-test testing the differences in number of words spoken per day between male and female speakers. Force `t.test()` to assume equal variances, do *not* use a directional alternative hypothesis. ▷

Exercise 4.21. Fit a least squares regression model using `lm()`, where the response is `words` and the predictor is `gender`. Produce a summary of the model with the function `summary()` (**not** with `summary.aov()`).

Which numbers match with the t-test results form Exercise 4.20. Can you explain what is happening? ▷

The trick is in the way the categorical variables are used in regression. Remember the usual equation for a linear line:

$$y = a + bx$$

Now, consider x is a categorical variable, e.g., gender, and we set it to 0 for women, and 1 for men. This means that expected value of y is equal to a for women (put 0 instead of x above formula and calculate), and expected value of y is $a + b$ for men. In other words, the difference between expected values of y for men and women is b. Hence, the t test performed for the slope means testing whether the mean difference between women and men is 0.

The above description is only a special case of what is called *indicator*, or *dummy*, coding. The coding turns analyses like t-test or ANOVA into linear regression. The concept is also directly related to the so-called 'contrasts' used in ANOVA. The particular way the variable x above is coded is called *treatment contrast*, which is the default in R.

We will return to this topic after we work on multiple regression.

5 Repeated measures

The analysis methods we have studied so far assume that the observations are independent. This assumption is often wrong, and it is intentionally violated in some experimental designs to increase the sensitivity of the tests. In this section we will exercise with a well known procedure *repeated-measures ANOVA* for analyzing (experimental) data where same subjects are measured more than once (hence, observations are not independent).

We will use two (again hypothetical) new data sets. In the first data set we assume that we investigate whether newborns distinguish their mother's native language from another language. We recruit 30 newborns, and when we find them awake during their first day in life, we let them listen to two comparable short stories, one in their mother's language and another in a foreign language. While they are listening to these stories, they are equipped with a pacifier through which we can measure their sucking rate. Crucially, each infant is tested on both conditions (the order of languages are randomized). Since our hypothetical newborns never fall asleep, start crying or spit out the pacifier in the middle of the story, we have 60 measurements from 30 infants. You can load the data set using,

```
> print(load(
  url('http://coltekin.net/cagri/R/data/newborn.rda
      ')
  ))
```

Notice that this time the file extension is different, and we used `load()` function. This data file is in R's native binary format. The advantage is that you retain all the information in the original data (including the variable name). The disadvantage is in its portability. The CSV files we used earlier can be read virtually in any environment by many applications, while the `.rda` files can only be read by R (but you do not need to worry about the platform or the operating system incompatibilities). The reason we wrapped the `load()` within a print is because of the fact that `load()` silently loads the variables in the data file. If you `print()` the return value, you will know which variables are loaded. Otherwise, you need to inspect your R environment to figure out the changes introduced by `load()`. If all went fine, you should have a new data frame named `newborn`. Note again that the data is fake, but the method, *high-amplitude sucking paradigm* described here without many details is a well-known technique for studying infants. For a real study of the sort described here see Nazzi et al. 1998.

The second data set we will use comes from another hypothetical language acquisition study. This time we are interested in children who are raised bilingually, where one of the languages they speak is 'home only' and the other language is also used in their school. The data set can be found at `http://coltekin.net/cagri/R/data/bilingual.txt` as a 'tab-separated file'.

Exercise 5.1. Read the tab-separated file `http://coltekin.net/cagri/R/data/bilingual.txt` into a new data frame with name `bilingual` (we will refer to this data set with this name throughout this section, but you can use a shorter name if you prefer).

You can use `read.delim()` for reading this file. Note that generic function for reading 'tabular files' is `read.table()`. ▷

Exercise 5.2. Inspect the data frame `bilingual`, and make sure that all variables in the data frame have correct data type. Particularly, we would like to make sure that all categorical variables are identified as `factor`s.

Optionally you can repeat Exercise 5.1, but supply the `read.delim()` (or `read.table()`) command with a `colClasses` option (see help on these functions for detailed use of this option). ▷

5.1 Paired t-test

As we did with independent-measures ANOVA, we will start with the simplest case: when we have only two conditions. If we only have two conditions measured repeatedly over a sample of individuals (people, animals, countries, hospitals, …) the we typically use a *paired t test*. Our data set `newborns` provides a textbook case application of *paired t test*. But for the sake of comparison, we will first take a detour, and revisit the independent samples t test.

Exercise 5.3. Assuming the `newborn` data came from two independent groups of babies (no baby is tested twice), test whether the babies respond differently to their native language and the foreign language.

Check whether t-test assumptions (except independence) are violated or not. ▷

In R, The paired t test can be performed using the same function, `t.test()`. All you need to do is pass the argument `paired=TRUE`.

Exercise 5.4. Perform a paired t test. Compare your results and to the independent-samples t test you have just performed. ▷

Exercise 5.5. Check whether the normality assumption of paired t tests holds for the `newborn` data with a normal Q-Q plot. What quantity has to be distributed normally?

Do you also need to check whether variances across the groups are approximately equal? ▷

Exercise 5.6. Plot side-by-side box plots of sucking rates for the native and the foreign language. Is the difference we are interested in clearly observable in the box plots? ▷

5.2 Repeated-measures ANOVA

Repeated measures ANOVA can be performed in R using a few different ways. In this tutorial, we will exercise with the function `aov()` that comes with the base R installation ('stats' package). `aov()` can handle only standard cases—no violation of the assumptions, no missing data— and only displays minimal information—no effect sizes. For more complex designs, one can use utilities found in additional packages or libraries, such as `Anova()` (note the capital 'A') from the `car` package and `ezANOVA()` (read 'easy ANOVA') from the package `ez`. We will conclude the section with an example run on `ezANOVA()`. However, we note that if your design is not ideal for repeated measures ANOVA you should probably use the 'multi-level' or 'mixed-effect' linear regression that we will see later in this tutorial.

As before, we will start with the simplest case. Remember that when we have two groups, the independent-measures ANOVA is equivalent to two-samples independent measures t test. Similarly, when we have only two groups, the repeated-measures ANOVA gives you the same results as the paired t test. Here is how we do a repeated-measures ANOVA using `aov()` on the data set `newborn`.

```
> m <- aov(rate ~ language +
    Error(participant/language),
          data=newborn)
> summary(m)
Error: participant
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 29   5792   199.7

Error: participant:language
          Df Sum Sq Mean Sq F value   Pr(>F)
language   1  306.9  306.95   28.24 1.06e-05 ***
Residuals 29  315.2   10.87
```

Not surprisingly, the p-value matches to the p-value found in Exercise 5.4. As in earlier ANOVA models, we specified a model predicting the `rate` from the `language` using the formula notation. Crucial difference is in the specification of the `Error()` term. This term specifies the replication in the experiment design. In this case, we tell `aov()` that the response corresponding to every level of `language` is measured for each `participant`. The specification of the error term could be confusing at first sight. Within the `Error()` term, the part before the slash '`/`' specifies the 'case' or 'subject' variable. The part after the slash specifies the 'within subject' variable(s). Note that we used `summary()` instead of `summary.aov()`, since the default summary method for an `aov()` object is ANOVA-like summary.

Without the `Error()` term, the function `aov()` is equivalent to the `lm()` function we used for independent measures ANOVA.

Exercise 5.7. Perform an independent measures ANOVA using `aov()` on the `newborn` data set. Compare your results with the repeated measures ANOVA results presented in the listing above. ▷

Exercise 5.8. Perform a repeated measures ANOVA that tests the effect of `age` on `mlu` in the `bilingual` data set. ▷

Extending the repeated measures ANOVA in Exercise 5.8 to include more predictors is easy. We just add all predictors to the formula notation as we do for the factorial ANOVA. You only need to be careful to include all the 'within-subject' variables in the error term. For example, for two-way within-subject factorial ANOVA with interaction term where age and language are the predictors, the error term becomes `Error(subj/(language*age))`.

Exercise 5.9. Perform a repeated measures ANOVA that tests the effect of `age` and `language` on `mlu` in the `bilingual` data set. Also include the interaction term in your analysis. ▷

Exercise 5.10. Use `interaction.plot()` to visualize the interaction between the variables `age` and `language` in the `bilingual` data. ▷

Often, we want to include predictors that are not or cannot be replicated in a repeated measures design. Such a variable in our `bilingual` data set is `gender`, which is a good example of a variable that can hardly be measured within-subjects. In this cases we use so-called mixed-design ANOVA analysis. There is nothing interesting specifying a mixed-design ANOVA in R. We just add the between-subject variable(s) to the model formula, but exclude it from the error term.

Exercise 5.11. Perform a mixed ANOVA with `age` and `language` as within-subject predictors, and `gender` as a between-subjects predictor. ▷

The above exercises exemplify a variety ANOVA designs that can be fit using `aov()`. However, `aov()`

- does not run any diagnostics or present corrected results in case of violation of the assumptions,
- does not report any effect size,
- cannot handle unbalanced designs or missing data.

In such cases a few packages in R provide solutions that are similar to other statistical software. We will only present an example using `ezANOVA()` from the package `ez`. However, when things are not perfectly balanced, and neat, the repeated-measures ANOVA becomes difficult to interpret. One reasonable course of action when ANOVA design becomes too complicated, or assumptions are violated at some level is to switch to so-called mixed-effect models which also offer some other benefits. We will discuss mixed-effect linear models later in this tutorial.

Listing 8 repeats Exercise 5.11 using `ezANOVA()`. The listing is slightly edited for clarity (you still need to exercise your skills in reading scientific notation).

The first thing to note in Listing 8 is the command `library()` on line 1. This includes the functions defined in library `ez`. There are a large number of (mostly free/open-source) libraries for R for various (statistical) tasks. In this tutorial, we try to stick to the bare-bones, but you should check existing libraries for the tasks that are not possible or difficult to do with the basic packages of R. The central repository for all R packages is at `http://cran.r-project.org/`. If the package you need is not installed on your computer, you can install any package from CRAN with the command `install.packages()`.

Returning to the ANOVA results Listing 8, the main findings should be the same as what you found with `aov()` in

Listing 8: An example with `ezANOVA()`.

```
> library('ez')
> ezANOVA(data=bilingual,
          dv=mlu,
          wid=.(subj),
          within=.(language, age),
          between=sex)
$ANOVA
        Effect DFn DFd        F        p      ges
2          sex   1  18 2.92e-04 9.87e-01 1.02e-05
3         lang   1  18 6.78e+00 1.80e-02 3.38e-02
5          age   2  36 1.74e+01 5.07e-06 1.47e-01
4     sex:lang   1  18 1.98e-01 6.62e-01 1.02e-03
6      sex:age   2  36 6.48e-01 5.29e-01 6.36e-03
7     lang:age   2  36 3.07e+00 5.87e-02 1.66e-02
8 sex:lang:age   2  36 1.42e+00 2.54e-01 7.76e-03

$`Mauchly's Test for Sphericity`
               Effect         W         p
5                 age 0.9937147 0.9478173
6          gender:age 0.9937147 0.9478173
7        language:age 0.9905139 0.9221786
8 gender:language:age 0.9905139 0.9221786

$`Sphericity Corrections`
               Effect  GGe    p[GG]  HFe    p[HF]
5                 age 0.99 5.38e-06 1.12 5.07e-06
6          gender:age 0.99 5.28e-01 1.12 5.29e-01
7        language:age 0.99 5.93e-02 1.11 5.87e-02
8 gender:language:age 0.99 2.54e-01 1.11 2.54e-01
```

Exercise 5.11. The additional information presented here include the effect size on the column labeled with `ges` (generalized η^2); the results of Mauchly's Test; and in case we could not maintain the sphericity assumption two common corrections used in the literature: Greenhouse-Geisser ϵ (GGe), and Huynh-Feldt ϵ (HFe) and their corresponding p-values. Furthermore, ezANOVA() includes options for cases when the data is not balanced (so called type I, type II and type III sums of squares—`aov()` only calculates type I sums of squares.). However, you should better read and understand these before using it in a real analysis.

We will stop the discussion of repeated measures ANOVA here, but we will revisit some of the concepts and exercises when we discuss the mixed-effect models.

6 Graphics

Graphs are important tools for making sense of our data and communicating our results. R provides many graphical routines to produce graphs that visualize data in useful ways. We have already worked with a few basic graphs in R. In this section, we will work with graphics in some detail.

As usual, first we will introduce some new data to work with. The new data set comes from a corpus study of child language acquisition. In a nutshell, the study is about how children may be extracting words out of a continuous data stream. Particularly, we are interested whether a set of statistics is helpful for identifying the word boundaries. The data can be found at `http://coltekin.net/cagri/R/data/seg.csv`. It contains four different statistics: (*pointwise mutual information* (PMI), *successor variety* (SV), *boundary entropy* (H) and *reverse boundary entropy* (RH)) calculated on all potential boundary locations for three child-directed utterances. The details are not that important for our purposes here, but if you like to know more about it, the data comes from Çöltekin 2011.

Exercise 6.1. Load the CSV file `http://coltekin.net/cagri/R/data/seg.csv` into a data frame named `seg` in your R environment.

Make sure that all columns in the data frame has sensible data types. Do appropriate conversions if necessary.

▷

6.1 Basic graphics

We used the `plot()` function before for scatter plots. This function behaves differently depending of the type of object to be plotted. In the simplest case, you can give a simple list of numbers, and `plot()` will plot them against their index, i.e., integers starting with one up to the number of elements in the data provided.

Exercise 6.2. Plot the `h` values in the order given in the data (against thier index value) for only the first utterance in the data frame `seg`. ▷

Exercise 6.3. Using the `seg` data set, display the relationship between `pmi` and `h` using a scatter plot. Make sure that `pmi` is placed on the x-axis, and the color of the points is red.

Plot the linear regression line over the scatter plot in blue.

▷

If you plot a data frame with `plot()`, it will result in a matrix of scatter plots where each variable is plotted against the other.

Exercise 6.4. Plot scatter plots of `pmi`, `h`, `rh` and `sv` against each other on a single graph.

TIP: remember that you can extract the relevant columns of the data frame with the syntax `seg[,4:7]`, or using symbolic names like `seg[,c('pmi', 'h', 'rh', 'sv')]`. ▷

We will see more examples of plotting different types of objects, and prettifying the plots like the one in the exercises above. For now, we will first exercise with some of the basic graphics we have seen before, and build on them slowly towards more advanced and nice-looking graphs.

Exercise 6.5. Plot the histograms of `pmi` and `h` values. Do the distributions look similar? ▷

Exercise 6.6. Using normal Q-Q plots, check whether `pmi` and `h` are distributed normally. ▷

Exercise 6.7. Plot side-by-side box plots of `pmi` for boundary (where `boundary == TRUE`) and non-boundary (where `boundary == FALSE`) locations in the first utterance in the `seg` data. ▷

So far, we have used `plot()` for plotting relations between two samples. We can also plot mathematical functions. For example, the following plots the well-known bell curve of the Gaussian function.

```
> x <- seq(-4,4,by=0.1)
> plot(x, dnorm(x))
```

We first create a vector variable that holds numbers between −4 to 4, and plot these number against `dnorm()`, which returns the value of the *normal density function* (more on density functions later). If you run the above command, you will see a set of dots tracing the standard Gaussian curve. If you want to see lines connecting these points instead, you can pass the option `type='l'` to the `plot()` function. Similarly, the option `type='b'` plots both (lines and points).

Exercise 6.8. Another interesting distribution is the Student's t distribution. The density function for the t distribution in R is `dt()`. Plot t distribution with degrees of freedom 5 using lines instead of individual points. Make sure that the curve is plotted in green, and the line is three times as thick as the default line width. ▷

It would be interesting to see both the normal and t distributions on the same graph. However, every time we run a `plot()` command R clears the earlier plot, and initializes a new 'canvas'. We have already seen the `abline()` function which allowed us to plot on an existing plot. There are more functions that draw over an existing graph. Most commonly used ones include,

- `points()` for plotting points.
- `lines()` for plotting lines.
- `linespoints()` for plotting lines.
- `text()` for plotting arbitrary text.

Exercise 6.9. Plot the density curves of the standard normal distribution and the t distribution with degrees of freedom 5 on the same graph. Make sure both are drawn with lines, and use a distinct color for each curve. ▷

The colors are useful for distinguishing different lines in a graph. However, the color distinction will be unreliable in black-and-white print. A common practice for identifying different lines on the same graph is to use different line types, or patterns. The commands that draw lines allow you to specify a different pattern using the option `lty`. For example, setting `lty=2` in `plot()` or `lines()` will draw a dashed line (the default is 1, which draws a `'solid'` line). Alternatively, you can use symbolic names like `'dotted'`, `'dashed'` etc.

Exercise 6.10. Plot the standard normal distribution and t distributions with degrees of freedom 1, 5, and 20 on the same graph. Use different colors and line types for each curve. ▷

Similar to `lty` that sets the line-pattern, you can also customize the type of the points drawn by `plot()` or `points()`. The parameter that decides the shape of the points drawn is `pch`. If you provide a single-character text string to `pch` it will use this character instead of the default. Alternatively, you can provide a numeric value to obtain a number of predefined shapes. For example `pch=22` will plot a filled square (see help text for `points()` for other symbols).

Exercise 6.11. Repeat Exercise 6.3, but use 'small solid circles' instead of the default hollow circle. ▷

With the `text()` command, you can place an arbitrary text on any point in the x-y plane. In its typical use, it is used like `text(x, y, labels)`, where all arguments are vectors of equal size. Furthermore, you can adjust the position of the labels with `pos` and `offset` options (see help text for more information).

Exercise 6.12. Repeat Exercise 6.9. Place the text strings *standard normal* and *t(5)* on appropriate places on the graph to identify the curves. Use the same colors for the text as the corresponding curve. ▷

Exercise 6.13. Repeat Exercise 6.2. However, use dotted lines instead of plotting individual points, and place the corresponding `phoneme` value above each point.

** Use red for the phonemes that correspond to boundaries and blue for word-internal locations.

This exercise, especially the last part, is rather tricky, but you have all the tools at hand to achieve this.

▷

6.2 Labels, axes, legends ...

In graphs like the ones in exercises 6.9 and 6.10, we typically include a *legend* to explain what colors or patterns mean. The command `legend()` in R adds a legend to an existing plot.

Exercise 6.14. Add a legend for the graph you produced in Exercise 6.10. Make sure that both line type and color matches with the lines on the graph. ▷

We improved the graph in Exercise 6.14 quite a bit, it is almost ready to be printed. However, the y-axis is labeled as 'dnorm(x)' which definitely is not the typical axis label found in printed material. You can specify the axis labels using `xlab` and `ylab` options, and a title on top with the option `main`.

Exercise 6.15. Repeat Exercise 6.10 but this time set the main title as 'normal and t distributions', set the y-axis label to 'density', and remove the axis label of the x-axis. ▷

By default R determines for a reasonable x-y region for your plots when you use `plot()` and the other functions that initialize a new graph. Sometimes, you may want to change the range of the values on the x- or the y-axis. Often you need to do this to make sure that the subsequent `points()` and `lines()` fit into the canvas prepared by a plotting function, sometimes you may want to extend one of the axes to leave some space for your legend, and sometimes you may want to include a particular reference point, for example, the origin (coordinates 0, 0) in the graph no matter what data to be plotted. To set the ranges that will be visible on a graph we use `xlim` and `ylim` parameters to the plotting functions. For example, `xlim=c(0, 10)` will result in the x-axis to cover the range between 0 and 10. Note that any graphics drawn outside the region specified by `xlim` and `ylim` will be clipped out, they will not be visible.

Exercise 6.16. Repeat Exercise 6.10 but this make sure that the origin, the point $(0, 0)$, is included in the graph. ▷

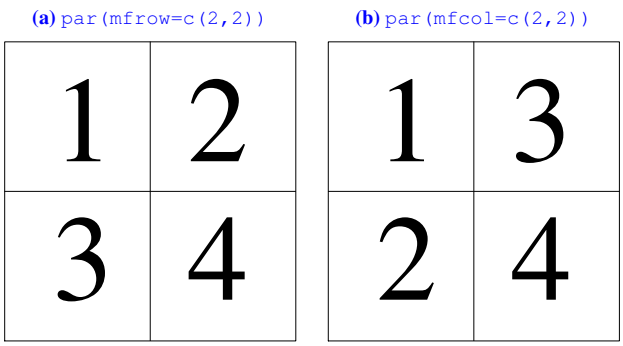


Figure 1: Order of graphs in 2x2 plots set up by (a) `mfrow` and (b) `mfcol`.

Exercise 6.17. Repeat the scatter plot in Exercise 6.3 in two steps. First, plot only the points that correspond to the boundary locations using a plus '`+`' sign instead of a circle. Next, plot the points that correspond to the non-boundary locations on the same graph using a minus '`-`' sign. Use different colors in each step. Include a main title, e.g., 'PMI vs. H', and make sure that the axis labels are printed in all capital letters. Place an appropriate legend indicating meanings of the symbols used.

Make sure all points fit into the graph. ▷

6.3 More than one graph on the same canvas

Often, we would like to display more than one graph on the same figure in a publication or presentation. One way to achieve this is to set one of the graphical parameters `mfrow` or `mfcol`. The graphical parameters in R are set using the command `par()`. Once set, these parameters will be effective for all graphics related commands. For `mfrow` and `mfcol` we specify a two-element vector, where the first element specify the number of rows, and the second one specifies the number of columns. For example, the command `par(mfrow=c(4,5))` creates a grid of four rows and five columns where next 20 `plot()` (or others like `hist()`, `boxplot()`) commands will place their output. The difference between `mfrow` and `mfcol` is the order of the plots. `mfrow` fills the specified grid following a row order, while `mfcol` fills the columns first. Figure 1 shows the order of graphs produced with `mfrow` and `mfcol` options.

Exercise 6.18. In Exercise 6.7, we have created box plots of boundaries and non-boundaries for the `pmi` values.

Plot four graphs on a 2x2 grid on the same canvas each displaying side-by-side box plots for boundary and non-boundary positions for `pmi`, `h`, `rh` and `sv` values. Set the main title accordingly to identify the graphs.

▷

The `par()` command sets many other graphical parameters. Some of these parameters, e.g., `pch` or `lwd`, serve as defaults to the later graphical commands, and can be overridden by the later commands like `plot()`. Some others, like `mfrow`, affect behavior that you cannot set through the individual commands. You are encouraged to skim through the help text for `par()` to get an impression of the parts of the R graphics that you can customize.

6.4 Writing your graphs to external files

Once you are happy with your graph, you will want to include it in your presentations and/or publications. You can, of course, get a screenshot, but in most cases, this method produces less than optimal graphics, especially for publishing. R supports a number of formats that produces 'publication quality' graphs. The possible file formats include Postscript, PDF, PNG and JPEG. Typically, if you want to have a bitmap file (for the web and possibly for your presentations), you should use *PNG* graphics. If it is for a publication, you should pick a *vector* file format, such as *PDF* (If you are a LaTeX user, you should definitely check `tikzDevice`, though).

To plot your graphics to an external file, you first need to use appropriate function to initialize the output 'device'. The initialization functions are typically the (lowercase) name of the graphics format you are interested in. For example, `pdf()`, `postscript()`, `png()` or `tiff()`. These functions somewhat differ depending on the file type you want to produce, but in almost all cases you need to specify a filename and the width and height of the resulting graphics. For bitmap graphics, the width and height are specified in pixels, for vector graphics it is specified in physical dimensions, e.g., in inches. You should consult the documentation of the functions you want to use. In general it is important to specify the correct size since some properties of the resulting graph, such as font sizes and line thickness, will be determined based on the size of the graphics. Once you have initialized the output, the commands you use for producing graphs are the same. When you are done with plotting your graph(s), you should type `dev.off()`. The resulting graphics will be written to the file you specified during the initialization.

Exercise 6.19. Plot the histogram and Q-Q plot (including the theoretical line) of the `pmi` values in `seg` data set on the same canvas next to each other (one row, two columns). Make sure that your graphs have sensible titles and axis labels. Use filled triangles for the Q-Q plot instead of the default circle. Write the results to a PDF file suitable for printing on A4 paper with one-inch margins on both sides. The width of an A4 paper is 8.27 inches, and you probably do not want to fill the whole paper, so you should use an image height about half of the image width. ▷

Exercise 6.20. Repeat Exercise 6.19 two times for producing PNG graphics of different sizes, 1024x512 (width x height) and 640x320. Display and compare the quality of the resulting graphics. ▷

6.5 Additional exercises

Exercise 6.21. Plot line segments passing through the following X-Y coordinates: (0,0), (1,1), (2,3) and (4,4). ▷

Exercise 6.22. In R, you can draw a *pie chart* with the function `pie()`. Plot a pie chart for the data used in Exercise 1.10. Use capital letters 'A' to 'D' as labels. ▷

Exercise 6.23. The function `barplot()` in R produces a *bar plot*. Repeat Exercise 6.22, but use a bar plot instead of a pie chart. ▷

Exercise 6.24. Draw *sine*, `sin()`, and *cosine*, `cos()`, functions in the range [-π, π]. Use a different color for each curve. **TIP**: for smoother curves, you need to use `seq()` to obtain data points with an interval smaller than one, for example `0.1`. **TIP2**: R defines a standard variable `pi` with the value of π. ▷

Exercise 6.25. We already know that `abline(a, b)` draws a straight line whose *intercept* is `a` and *slope* is `b`.

Using `abline()`, add horizontal and vertical lines that pass from the origin (0,0) to the graph you produced in Exercise 6.24. ▷

Exercise 6.26. Replicate the graph in Figure 1.

▷

7 Regression again

Linear regression is one of the basic tools in data analysis. In its classical application, linear regression is used when the response variable and the predictor(s) are both numeric (interval or ratio scale) variables. However, as exemplified in Section 4.6, and as we will demonstrate further, many of the statistical analysis techniques can be cast as variations (or extensions) of linear regression. In this section we will revisit the correlation and regression with single predictor variable, and continue with the multiple regression and model selection. First, we will again prepare the data we will use.

The new data set we will use in this part contains geographical distance and aggregate pronunciation difference of 613 locations in the Netherlands. It has two variables, linguistic distance measured by average string edit distance of a set of words, and geographic distance in kilometers. The data is a subset of the Dutch date set analyzed by Nerbonne 2010. Note that (to save myself from some difficult questions) the distances corresponding to the sites very close to each other are not included in the data set we will use here. The data can be found at `http://coltekin.net/cagri/R/data/ling-geo.rda`.

Exercise 7.1. Load the R data file from `http://coltekin.net/cagri/R/data/ling-geo.rda` into your R environment. If all goes right, you should have a new data frame named `lg`. ▷

We will also make use of the data set `seg` from Section 6, and the `mlu` data set from Section 3. If you do not have the data in your R environment, you should return to Exercise 6.1 and to the beginning of Section 3 to load or create these data sets.

7.1 Correlation

We have introduced how to calculate some of the correlation coefficients earlier in Section 2.2. Here, we will revisit this, and introduce the way to make inferences regarding correlation. We will not dwell on the assumptions of correlation. The Pearson's correlation coefficient shares all its assumption with least-squares regression with a single predictor, and we will check these assumptions in the next subsection.

Exercise 7.2. Using the `seg` data set, find the following correlation coefficients between `pmi` and `h`.

- Pearson's r
- Kendall's τ
- Spearman's ρ

Remember that all can be calculated using the same function `cor()`. ▷

The correlation coefficient calculated by `cor()` gives you the expected correlation in the population. However, it does not include any inferential information. The function `cor.test()` gives you the confidence interval and a hypothesis against with the null hypothesis 'there is no correlation (the coefficient is equal to 0)'.

Exercise 7.3. Perform `cor.test()` for the values you have calculated in Exercise 7.2. ▷

If you pass a data frame to `cor()`, it will return a matrix of all pairwise correlations between the variables in the data frame. This is handy if you have a large number of variables, and you'd like to inspect correlations between them.

Exercise 7.4. Calculate pairwise r values between `pmi`, `h`, `rh`, `sv` and `boundary` columns of the `seg` data frame.

TIP: In this exercise, we want to take a subset of 5 columns from a 7-column data frame. It is easier to exclude the remaining two columns instead of listing all 5 we need. In R a negative index value results in exclusion of the data point. For example, a index value of -1 will exclude the first item, and likewise, a vector index of the form $-\mathtt{c}(1, 2)$ will exclude the first two items. ▷

One thing to note in Exercise 7.4 is that one of the variables we have included in the correlation analysis (`boundary`) is not a numeric variable. Correlation coefficient between the numeric and categorical variables can be calculated using so-called *Point-biserial correlation coefficient*, which gives yet another hint that linear models can handle both numeric an categorical data types.

7.2 Least-squares linear regression

In Section 3, we already did some exercises with linear regression. In summary, we fit a linear model using `lm()`, and extract the information we need in most cases using `summary()`. We worked on these steps earlier in Listing 5 for predicting mother's MLU from the child's MLU (repeated here as Listing 9 for convenience).

You should already be able to interpret most of the information presented in Listing 9. If not, you are encouraged to revise Section 3.

Exercise 7.5. Using the `lg` data set, fit a linear regression model predicting the linguistic difference from geographic distance. Produce the summary of linear regression fit.

Assuming the modeling assumptions are sound (we'll check these shortly),

1. Is geographic distance reliably reflected in linguistic difference, e.g., is the slope of the model statistically significant?
2. What does the intercept represent in this model?
3. How much of the linguistic difference is explained by the geographic distance?

▷

Listing 9: Summary of a linear regression analsysis (repeated from Listing 5).

```
> m <- lm(mlu$mot ~ mlu$chi)
> summary(m)
Call: lm(formula = mlu$mot ~ mlu$chi)
Residuals:
      Min        1Q    Median        3Q       Max
-0.79928 -0.14665  0.06142  0.14003  0.66232
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    5.7133     0.2326  24.559   <2e-16
mlu$chi        0.1503     0.0839   1.791   0.0871

Residual standard error: 0.3182 on 22 degrees of
      freedom
Multiple R-squared:  0.1272,    Adjusted
      R-squared:  0.08757
F-statistic: 3.207 on 1 and 22 DF,  p-value:
      0.08708
```

7.3 Model diagnostics

Here, we will first focus on model diagnostics, checking whether the least-regression assumptions hold or not. Previously we extract residuals and tried to check whether they are normally distributed or not. R provides an easier way of checking various assumptions of linear regression. All you need to do is to type `plot(m)` (assuming you saved your model as `m`). This produces four graphs presented in Figure 2.

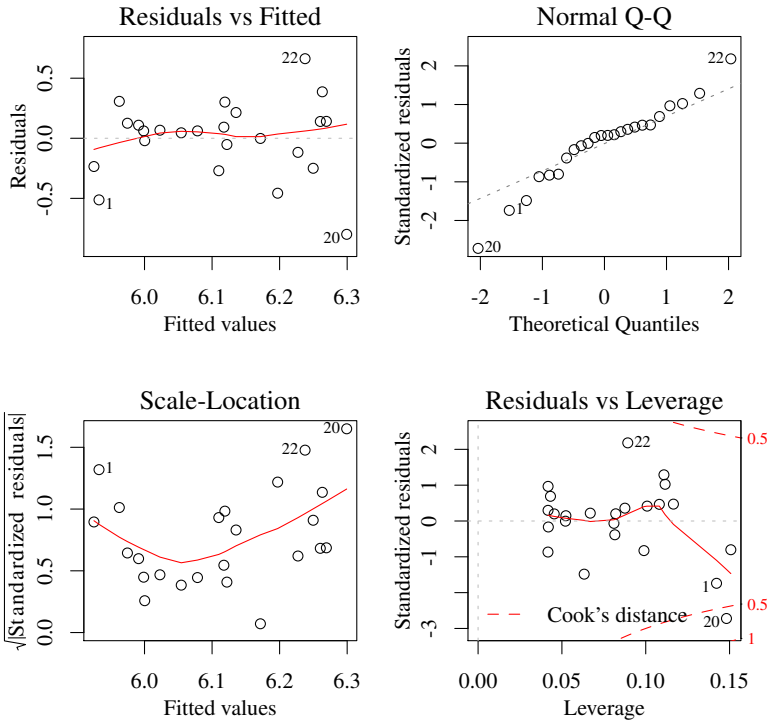


Figure 2: Diagnostic plots for linear regression.

All graphs in Figure 2 expose possible violations of modeling assumptions. The top-left graph, *residuals vs. fitted* gives an indication of whether your residuals are correlated or not (independent). What you do *not* want to see here is to have any sort of pattern. The graph at the bottom left, *fitted vs.* $\sqrt{|\ standardized\ residuals\ |}$ is basically the same graph with a different scale. It helps seeing some patterns that are difficult to see in the first graph. In both graphs, you could detect nonlinearities and if variance is not constant across the predicted values. For a regression analysis with single predictor, you can see these values from a simple scatter plot (you may need to tilt your head a bit to see the resemblance). However, as the number of predictors increase, scatter plots will not be as useful, but *fitted vs. residuals* graphs will still be a useful diagnostic.

In both graphs, although it is difficult to decide due to small number of data points, it seems variance decreases around the region that correspond to fitted values of 6.0 to 6.1 of the mother's MLU. The *fitted vs.* $\sqrt{|\ standardized\ residuals\ |}$ graph also indicates some mild non-linearity. The cases that may be causing the problems are labeled (by their row numbers in the data frame). These graphs identify data points 1, 20 and 22 as potential causes of poor model fit.

The top-right graph is the familiar Q-Q plot. For least-squares regression, we want our residuals to be normally distributed. Again, it is difficult to judge due to small number of cases, but our example seems to diverge slightly from the ideal line in both tails.

The last, bottom-right graph gives an indication of influential data points. *Leverage* of a data point is the distance of a data point from an average data point on the x-axis (or on the multidimensional space defined by all predictors). The points that have high leverage and/or large residuals are likely to be very influential. *Cook's distance* is a measure of influence of a particular data point. You can think of Cook's distance as measuring the effect of removing a data point from the regression analysis. If resulting parameter estimates change drastically, the point will have larger Cook's distance. The contour lines drawn (at Cook's distance 0.5, and 1.0 if you have such points) indicate influential data points. The points that appear outside the contour lines need particular attention. The data point 20 is obviously the most influential case, and 1 and 22 may also needs some attention. Although opinions differ, a rough guideline is that a Cook's distance above 1 should be a cause of concern.

Once you are convinced that a particular observation is an outlier, you may want to remove it from the model fit. `lm()` and similar functions provide a handy notation to select or exclude certain observations. The argument `subset` may be used to specify the row index for data in use. Alternatively, you can specify the `data` option with already selected rows, e.g., `data=mlu[c(1:19,21:24),]`, or simpler `data=mlu[-20,]`.

Exercise 7.6. Repeat the analysis in Listing 9 twice. First, without the most influential observation according to Figure 2. And second, without an observation (approximately) at the center with respect to the child's MLU values (such an observation is unlikely to be influential, why?).

Compare the change in coefficient estimates, and r^2 value with respect to the summary displayed in Listing 9. ▷

Exercise 7.7. Produce the diagnostic plots for the model you fit in Exercise 7.5. Make sure all four diagnostic plots are plotted on the same graph, and use gray 'dots' for the points plotted to reduce the clutter you get due to large number of points (this will also make the lines without a specified color gray, but that's fine).

1. Do you see any non-linearity?
2. Do the residual look normally distributed?
3. Is the variance of the residuals constant?
4. Are there very influential data points?

▷

Exercise 7.8. Repeat the plot you have produced in Exercise 7.7, this time save the result into a PNG file. ▷

7.4 An example transformation

If you were careful enough in Exercise 7.7, you should not be content with the assumptions of the model we fit in Exercise 7.5. When regression assumptions especially the linearity assumption, are violated, one of the ways to fix it is to transform the predictor or the response variable. Normally, inspecting your data is helpful in detecting non-linearities, and deciding for possibly useful transformations. Here, we will first do the transformation first, and then justify it.

Exercise 7.9. Add a new variable `log.geo` to `lg` data frame which contains the log-transformed geographic distance. ▷

Exercise 7.10. Fit a linear regression model using `log.geo` as the predictor of the linguistic differences.

1. Compare the output with the earlier output without log transformation.
2. Plot the diagnostic graphs, and compare them with the one you saved in Exercise 7.8.

▷

7.5 Predictions of a linear model

The inference for the slope indicates a reliable relationship between the outcome variable and the predictor. In other words, if modeling assumptions are correct, and the t-test performed for the slope is statistically significant, we confirm that predictor has an affect on the response variable within the chosen level of significance. Besides this 'hypothesis testing' interpretation, a linear regression fit can also be used for predicting the outcome variable for new values of the predictor. Given a predictor value x, and the coefficients a and b, the value of the outcome variable, $\hat{y}$, can simply be calculated using the formula

$$\hat{y} = a + bx$$

Note that this prediction does not include the uncertainty associated with the estimate.

In R, you can use `predict()` to calculate model predictions (and confidence) about unseen data points. For example, assuming the variable `mlog` contains the model that predict linguistic difference from the logarithm of the geographic distance (from Exercise 7.10), Listing 10 demonstrates the use of `predict()` function to obtain the predicted linguistic differences between sites whose distances are 10, 20, 50 and 100km, along with the lower and upper values of the 95% confidence interval for the estimated regression line.

Listing 10: Example use of `predict()`.

```
> predict(mlog,
          newdata=data.frame(
                log.geo=log(c(10, 20, 50, 100))),
          interval='confidence')
        fit       lwr       upr
1 0.1053994 0.1045929 0.1062059
2 0.1462061 0.1456081 0.1468042
3 0.2001497 0.1998106 0.2004888
4 0.2409565 0.2407551 0.2411578
```

Note that the `newdata` option of `predict()` requires a data frame where the predictor variable(s) are named the same as the original data frame used for fitting the model. This may seem unnecessary for simple regression, but it will be clear why a data frame is required when we work with multiple regression. If you do not provide the `newdata` option, `predict` will use the data used for fitting the model.

The option `interval='confidence'` in Listing 10 results in the confidence interval for the estimated regression line: the expected value of the response variable at a given predictor value. If you specify `interval='prediction'`, instead, you will get an interval where 95% of the future observations are expected to fall. You should think about the former as your estimate of the mean in the population, and the latter as the expected range of individual observations. If the number of observations used for fitting the model is large, the former (the confidence interval of the estimated mean) will be tight. However, the latter (the individual values sampled from the population) is unlikely to be affected by the increased number of observations during the model fit. If you skip the `interval` option, `predict()` returns only the predicted values without any interval.

Exercise 7.11. Using the `mlu` data set (see the beginning of Section 3) perform the following:

1. Plot a scatter plot of the mother's MLU vs. the child's MLU.
2. Fit a linear regression model predicting the mother's MLU from the child's MLU, assign the result to a variable.
3. Draw the linear regression line over the scatter plot using a red solid line.
4. Draw lines for the lower and upper bounds of the 95% confidence interval for the linear regression estimate over the scatter plot and the regression line. Use a different color and line type than the regression line.

   TIP: You can create a number of, e.g., 20, equally spaced values that span the x axis by
   `seq(min(mlu$chi), max(mlu$chi), length.out=20)`,
   and use it with predict to obtain the lower and upper bounds of the confidence intervals.

▷

Exercise 7.12. Plot the scatter plot of geographic distance vs. linguistic difference. Use gray dots for the points plotted.

Plot `geo` vs. predicted values from the linear model *without* log transformation (the model in Exercise 7.5) using a red solid line.

Plot `geo` vs. predicted values from the linear model *with* log transformation (the model in Exercise 7.10) using a blue dashed line.

Use sensible axis titles, and include an appropriate legend in the resulting figure.

▷

Exercise 7.13. We did not plot our confidence bands using the `lg` data set since due to large number of data points the confidence intervals is very small and not visible on the graph.

For the sake of demonstration, draw the confidence bands as in Exercise 7.11 for the model predicting linguistic differences from the logarithm of geographic distances. While plotting the confidence bands, use `newdata` option of `predict()` to obtain 20 equally spaced points over the x axis.

▷

8 Multiple regression

In our many passes over the regression so far, we have only worked with a single explanatory variable. In this part extend it to multiple predictors. Although it does not change the model fitting drastically, there are a number of issues that come with multiple predictors.

The data we will use in this part is another hypothetical study on child language acquisition. This time we want to investigate the effects of amount of time spend in front of TV to two-year-old children's language development. The data can be found as an R data file at `http://coltekin.net/cagri/R/data/tv.rda`. The response variable in this data set, `cdi`, is a standard measure of children's language abilities based on parental reports. The predictor we are mainly interested in is `tv.hours`, which is the weekly hours of TV time for each child. We have some other predictors that we will explain in the relevant exercises below. As in most cases in this tutorial, the problem/design is simplified and the data is randomly generated.

Exercise 8.1. Read the data from `http://coltekin.net/cagri/R/data/tv.rda`. If all goes fine, you should have a new data frame called `tv`. ▷

8.1 Revisiting single regression (for the last time)

Multiple regression is not very different than simple regression we studied piece-by-piece in multiple steps so far. In this subsection, we will fit two simple regression models for later comparison with the multiple regression.

Exercise 8.2. Fit a simple regression model where `tv.hours` is the only predictor of the `cdi` score. Store the model object in variable `m1`.

1. Summarize the results, what does the model predict about the effect of watching TV?
2. Produce the diagnostic plots. Make sure that all four diagnostic plots produced are shown on the same graph.
3. Plot a scatter plot of `cdi` against `tv.hours`, and the regression line over it.
4. Plot 95% 'confidence' and 'prediction' bands around the regression line. Use distinct colors and line types.

▷

Exercise 8.3. Fit a simple regression model where `mot.education` is the only predictor of the `cdi` score. Store the model object in variable `m2`. Summarize the results, and compare it with `m1`.

You are encouraged to repeat Exercise 8.2 fully for `m2`. ▷

8.2 Multiple regression

To fit a regression model with multiple predictors, we add each predictor to the right side of the tilde '`~`' in our model specification (formula). Note that for numeric predictors, the interaction terms are difficult to interpret. As a result we almost exclusively use `+` between the predictors.[1]

Exercise 8.4. Fit a multiple regression model predicting `cdi` from `tv.hours` and `mot.education` using the `tv` data set.

Save the new model object as `m3`.

Summarize the new model, and compare it with the earlier models `m1` (from Exercise 8.2) and `m2` (from Exercise 8.2).

In particular, you should be checking the changes in the p-values, and the r^2. Can you explain the differences that you observe? ▷

The scatter plots that were so useful in single regression becomes rather useless in the context of multiple regression. For two predictors, you can plot so-called 3D scatter plots. A 3D scatter plot is hardly digestible except for simple/demonstrative data sets. We will not experiment with them here, but note that a number of R packages, including `scatterplot3d` and `rgl` and `Rcmdr` have functions to plot 3D graphs.

Exercise 8.5. Plots two side-by-side scatter plots using the `tv` data set. One `cdi` against `tv.hours` and another `cdi` against `mot.education`.

On both plots, draw the single regression line, and the resulting line from the multiple regression `m3` (the intersection of the regression plane and the plane defined by the relevant axes).

TIP: You can use `coef()` to extract coefficients from `m3`. For example, `coef(m3)[1]` will give you the estimated intercept.
▷

Exercise 8.6. A typical visualization trick for representing high dimensional data is to use colors (or shades) to represent one of the dimensions. In this (somewhat tricky) exercise, we want to see for which values of the predictors `tv.hours` and `mot.education` we have high `cdi` values.

Plot squares filled with shades of gray that represent the predicted `cdi` scores for each pair of `tv.hours` and `mot.education` values within the range of data.

The maximum CDI value should be plotted in black, and the minimum should be plotted in white. Put `tv.hours` to x axis,

[1] If you include interactions with `:` or `*`, R will include new predictors that are products of the original variables. Technically, there is nothing wrong with this, but it rarely has a straightforward interpretation.

and `mot.education` to the y axis. For both, use the integer values within the range in the `tv` data set.

TIP1: you can initialize a new graph, but do not plot anything on it (except axes, labels etc.) using the option `type='n'`.

TIP2: the function `gray()` returns color values that can be used with plotting functions. Alternatively, you can use `rgb()` produce a colorful graph instead. ▷

8.3 Multicollinearity

One of the problems that affect the parameter estimates and interpretation of a multiple regression analysis is multicollinearity. If two predictors are collinear, part of the variance in the response variable is explained by both. Multicollinearity does not affect the predictive power of the model. However, the coefficient estimates will become less certain, and difficult to understand. If a predictor which is collinear with the existing predictors is added to a regression model, the earlier estimates may change. A sign of multicollinearity is the high linear correlation between the predictors.

Exercise 8.7. Check the correlation between the predictors of `m3` from Exercise 8.4, using `cor()`. ▷

An extreme case of multicollinearity is when one of the predictors is a linear combination of one or more other predictors. If this is the case, the least-squares estimation will not be possible.

Exercise 8.8. Fit another model, where as well as `tv.hours` and `mot.education`, their sum is also a predictor. Note that to make sure that `+` works as an arithmetic operator in a model formula, you need the function `I()`. In this case the new predictor should be specified as `I(tv.hours + mot.education)`.

Summarize the result.

▷

A well known measure of collinearity is *variance inflation factor* (VIF). The VIF represents the increased uncertainty of a coefficient (slope) estimate of a predictor due to the predictors in the model. The details of the calculation is, as usual, not we are interested here, but the following formula may give you some insights. Variance inflation factor for the j^{th} predictor, x_j, is

$$\mathrm{VIF}_j = \frac{1}{1 - r_j^2}$$

where r_j^2 is the r^2 value obtained by fitting a model predicting x_j from the rest of the predictors. For example, if our aim is to check VIF of x1 in the model specified by `y ~ x1 + x2 + x3`, we use the r^2 from the regression `x1 ~ x2 + x3`. Clearly, the higher the VIF, the higher the variability on estimates of coefficients. Although there is no clear-cut rule, a common suggestion to consider values above 5 to high VIF. You should have already realized that each predictor has an associated VIF.

You already know how to calculate VIF without any additional help. However, you can also use the function `vif()` from `car` package (Fox and Weisberg 2011). The (Fox and Weisberg 2011) package contains quite a few useful/convenient functions for regression and ANOVA.

Exercise 8.9. Calculate VIF values for the predictors in the largest model we studied so far, `m3`.

Reminder: to be able use the functions from the library `car`, you need to first run

```
> library('car')
```

If your installation does not have the `'car'` library, you can install it (with proper rights on a computer connected to Internet) using the command

```
> install.packages('car')
```

▷

8.4 r^2 and adjusted r^2

We already know that r^2 is the square of the Pearson's correlation coefficient, and interpreted as the 'the ratio or percentage of the variance in the outcome variable explained by the predictor'. With multiple regression, there is no straightforward correlation coefficient anymore. However, the 'Multiple R-squared' reported in the regression summary still has the interpretation of 'the ratio of the variance in the outcome variable explained by the predictors'.

Despite this nice interpretation, r^2 is inflated by adding more predictors, even if the new predictors have no explanatory power. The adjusted version, noted as $\bar{r}^2$, corrects for this behavior. The following formulation of the $\bar{r}^2$ helps relating it to r^2 (it looks crowded but is is quite easy to digest if you make the attempt).

$$\bar{r}^2 = 1 - \left[\frac{n-1}{n-k-1} \times (1 - r^2)\right]$$

where n is the number of observations, and k is the number of predictors (not including the intercept term). You should realize that $\bar{r}^2$ is always smaller than r^2, and as k gets larger, the difference will be bigger. For large data sets, on the other hand, the difference will be smaller (k will have little effect on the adjustment). One can also view $\bar{r}^2$ as the estimate of r^2 for the population (this will be more apparent if you rearrange the formula in terms of the 'model sums of squares' and the 'total sums of squares').

Exercise 8.10. The function `runif()` returns a random sample from a uniform distribution. For example, `runif(80, 0, 1)` returns a vector of 80 random numbers between 0 and 1, which are distributed uniformly.

Create two additional columns in `tv` data set, and fill it with random numbers from a uniform distribution (the range of the numbers does not matter).

Fit a new model where these two 'random' columns are added as predictors besides `tv.hours` and `mot.education`.

Compare the 'R-squared' and 'Adjusted R-squared' values with and without the new irrelevant (random) predictors. ▷

8.5 Model selection

In fitting models with multiple predictors, we typically experiment with including or excluding predictors. Deciding for an adequate model depends on many factors, such as the predictors that are measured/collected and aim of the model. For example the choice of a model over another can be very different if the aim of the modeling is to make predictions as opposed

to understand the coefficients (effects). The ground rule is, all else being equal, we prefer simpler models. As well as the *Occam's razor*, for the purpose of understandable coefficient estimates, simplicity is also motivated by the fact that with many predictors, the parameter estimates become too variable to be interpretable. For the purposes of prediction, large number of predictors will typically be costly to be measured, and they will cause *overfitting*: the model will perform well for the know data, but will perform poorly on unseen data. Here, we will not make any strong suggestions about model selection, but try to introduce some common tools that help comparing models.

The function `update()` in R allows you to modify an existing model gradually. Once we have a base model, instead of specifying a new model from scratch, we can define a model that adds or removes variables to or from the base model. For example,

```
> update(m, . ~ . + x1)
```

adds a new predictor `x1` to a model `m` that is fitted earlier, for example, with `lm()`. Similarly,

```
> update(m, . ~ . - x2)
```

returns a new model where the predictor `x2` is removed from the original model `m`. The dot '`.`' in these formulas represents the set of values from the original model (`m`). In both cases, the result is a new model object.

Exercise 8.11. The `tv` data set includes another predictor `daycare.hours` which specifies the number of hours per week the child spends in a daycare.

Using `update()` create a new model that adds this predictor to the model `m3` from Exercise 8.4. Name the new variable `m4`.

Does the new predictor affect the model fit? Is the effect statistically significant? ▷

Any additional predictor, even a completely random one, will result in a better fit: the residual variance will be lower (and r^2 will be higher). The question we often ask is whether the reduced residual variance can be a chance effect or not. This leads us to compare residual variances of two models. You should already be familiar with the concept of comparing two variances: this is what we do in ANOVA. If you run `anova()` on two model objects, it will perform an F-test, and tell you whether the residual differences are statistically significant.

If a model (with all the predictors) is doing something useful, it should be doing better than predicting the mean of the response variable for any predictor value. Such a test is automatically done when we summarize linear regression results. The F-test result on the last line of a linear regression summary is from such a test.

Exercise 8.12. The formula '`response ~ 1`' specifies a model with no predictor, or more correctly a model with a constant prediction.

Using the `tv` data set, fit such a model predicting `cdi`. Assign the resulting model to variable `m0`.

Check whether the estimated intercept is the same as mean of the response variable.

What does the standard error of the estimated intercept correspond to? ▷

Exercise 8.13. Perform an ANOVA comparing `m0` (Exercise 8.12) and `m1` (Exercise 8.2).

Do the improvement (reduction) in residual variation in `m1` statistically significant (at level 0.05)?

Compare your result with the F-test reported in the summary of model `m1`.

▷

Exercise 8.14. The ANOVA we have performed in Exercise 8.13 serves only as a demonstration. The same test is already included in the model summary. It is more interesting to compare two different models with predictors.

Compare `m1` vs. `m3` and `m3` vs. `m4` using `anova()`.

Note: conventionally, the smaller model is listed first.

What conclusions do you get out of these comparisons? ▷

If you run `anova()` on a single model object, it will perform F-tests incrementally for each predictor. The order of tests performed depend on the order the predictors in the regression formula. `anova()` first tests the model with the first predictor against the model of the mean, then the model with first two predictors against the model only with the first predictor, and so on. As a result, changing the order of predictors will result in different results.

Exercise 8.15. Repeat the tests performed in Exercise 8.14 by a single call to `anova()`. ▷

Akaike information criterion (AIC) is a measure used for model selection. The AIC is

$$\mathrm{AIC} = 2k - 2\log(L)$$

Where k is the number of predictors, and the L is the likelihood (of the data given the estimated model), which is another measure of model's fit to the data (we will return to discussion of likelihood).

Given two models, the model with *lower* AIC value is preferred. Notice that the AIC rewards good fit (higher L), [2] and penalizes the modes with large number of predictors (lower k).

The function `AIC()` calculates the AIC value for a given model.

Exercise 8.16. Compare the models you compared with `anova()` in Exercise 8.14 with `AIC()`. Which model is the best according to AIC? ▷

When there are many predictors, one may want to employ an automatic method for model selection. Note that the number of models to compare when you have k predictors is 2^k. For example, for $k = 10$ this amounts to 1024 comparisons, and for $k = 20$ the number of comparisons required is over a million. With increasing k, comparing all possible models becomes intractable.

Even if comparing all possible combinations of coefficients is possible, you should not expect any mechanical procedure to find you the best model. Normally your model selection should be guided by the knowledge relevant to the problem at

[2] Hint: logarithm of a number between 0 and 1 is between $-\infty$ and 0 respectively

hand. However, a few search procedures that look for an optimal model are commonly used in the literature. The function `step()` provides the well-known *backward* (starting with the full model, and eliminating variables that are deemed not useful), *forward* (starting with no or a few predictors, and adding ones that improve the model most) procedures, or a combination of `both`. If you want larger models to be explored, you will need to use the `scope` option to specify which predictors `step` should consider adding.

Exercise 8.17. Apply `step()` to the full-model (`m4` from Exercise 8.11) on `tv` data set. ▷

9 General Linear Models

The main distinction we made between the models (or analysis methods) we discussed so far has been based on the type of the predictors. This follows the historical distinction made in the literature. If the predictors are categorical, we use ANOVA. If they are numeric, we use regression. These distinction turns out to be superficial. Both type of analyses can be carried out a generalization of the linear regression, *general linear models*. We have already made an excursion into general linear models in Section 4.6, by analyzing a categorical predictor with `lm()`. In this part, we will built on this, working with categorical variables with more than two levels, and also mixing the categorical and numeric predictors.

The new data set we use comes from another hypothetical study where we try to assess the factors that affect school children's success in learning a second language (L2). We assume that all our participants are kids at the same age and schooling but from different backgrounds. The response variable is a test score for second language learners (`l2score`), and we will use a variety of predictors including the amount of L2 exposure outside the class (`l2exposure`), a standardized score of overall achievement in school (`gpa`), proficiency in their first language (`l1score`), socio-economic status of the parents (`ses`) and the gender (`gender`). The data set is available as a CSV file at `http://coltekin.net/cagri/R/data/l2.csv`.

Exercise 9.1. Load the CSV file from `http://coltekin.net/cagri/R/data/l2.csv`. Name the resulting data frame `l2`. Make sure that the variables `gender`, `ses` and `subject` are factor variables. ▷

Exercise 9.2. Using the `l2` data set, test whether gender affects the school success (`gpa`) using a linear model fit by `lm()`.

What does the intercept and slope represent in this model?

Verify your conclusions with an equivalent t-test. ▷

9.1 Categorical variables in regression

A categorical variable with k levels (groups) is represented with $k-1$ separate variables in linear regression. The most common way of representation is called *indicator*, *treatment* or *dummy* coding. In this type of coding, one of the levels are taken to be the base or the reference level (e.g., control group in a experimental study). For the reference level, all $k-1$ variables are set to 0. For the other levels, one of the $k-1$ variables are set to 1, the rest are set to 0.

In our gender example above, since we have only two levels, we have only one indicator variable (should be named `genderM` in Exercise 9.2). The indicator variable is set to 0 for female subject, and it is set to 1 for male subjects. If we write the linear predictor as

$$gpa = a + b \times genderM$$

it should be clear that for girls, we prediction of the GPA to be $a + b \times 0 = a$, the intercept. For male subjects, the prediction will be $a + b \times 1$. As a result the slope, b, will represent the difference between the means of male and female participants. Then, the t-test performed for the slope indicates whether there is a difference between the means of male and female participants.

Extending this idea to a variable with more than two levels is easy. For example, if we perform a similar analysis predicting `gpa` from `ses`, the estimated linear predictor is (assuming `low` is the base level)

$$gpa = a + b_1 \times ses.mid + b_2 \times ses.high$$

where `ses.mid` and `ses.high` are two indicator variables, that are set to 1 only for the corresponding level of the `ses`. Table 1 shows the relationship between the levels of the `ses` and the indicator variables.

Table 1: Indicator variables for three-level factor `ses`.

	indicator variable	
	ses.mid	ses.high
low	0	0
mid	1	0
high	0	1

Exercise 9.3. Fit a linear regression model predicting `l2score` from `ses` using the `l2` data set.

How do you interpret each coefficient and the corresponding significance value? ▷

Exercise 9.4. We already know that we can get a classical ANOVA summary using `summary.aov()`. Summarize the model in Exercise 9.3 using `summary.aov()`. Which quantities are the same on both summaries? ▷

The interactions of categorical variables are represented in a straightforward way in indicator coding. What we do is simply include the product of each of the indicator variables. For example, for the `gender` and `ses` factors we were working with, the resulting predictor would be like

In this case, the intercept represents the case where both variables are in their reference level (`gender = F` and `ses = low` in our example). The slope values should now be interpreted with care. For example, if `gender = M` and `ses = mid`, the expected response is $a + b_1 + b_2 + b_4$.

Exercise 9.5. Fit a linear model predicting `l2score` from `ses`, `gender` and their interactions.

Try to interpret each coefficient. ▷

Specification of intercept is implicit in R formulas. We can force intercept to be 0 by adding `-1` to the right side of a formula. For linear models with continuous predictors this does

not make much sense. For linear models with categorical predictors, on the other hand, forcing intercept to be 0 makes the coefficient estimates the estimates of means for each level. In combination with the standard errors of each estimate, this parametrization of a linear model may be more insightful in some cases.

Exercise 9.6. Fit a linear model predicting `l2score` from `ses` with no intercept. ▷

9.2 Other ways of coding categorical variables: contrasts

The indicator coding of categorical variables we worked with so far is easy to interpret. However, there is nothing special about this 0-1 coding scheme. A categorical variable with k levels can be coded as k−1 variables many different and useful ways. We will not go into details of contrast coding in this tutorial, but just mention a few ways to do it.

The default contrasts for a factor variable, as we saw above, is so-called *treatment contrasts*. However, if you create a factor variable as an *ordered* factor (either using the `ordered()` function or `ordered=T` option of `factor()`),[3] the default contrasts become so-called *polynomial* contrasts. Instead of differences between means, the polynomial contrasts test whether there is a polynomial trend of some sort along the ordered levels.

Exercise 9.7. Add a new variable named `ses.ordered` to the `l2`, which is an ordered copy of the `ses` variable.

First plot side-by-side box plots of `l2score` for each `ses.ordered` group. Do you observe of trend based on SES?

Now, fit a linear regression model predicting `l2score` from `ses.ordered`.

Compare the summary of the model with the one from Exercise 9.3.

▷

Exercise 9.8. Change the order of levels of the `ses.ordered` created in Exercise 9.7 as `"high"<"low"<"mid"`, repeat the model fit, and compare your results with Exercise 9.7.

▷

So, far we worked with the default contrasts for different types of factor variables (ordered and unordered). You can display and set contrasts for a factor variables using the function `contrasts()`.

Exercise 9.9. Inspect the contrast matrices of variables `gender`, `ses` and `ses.ordered` in the `l2` data set. ▷

You can create well-known contrasts using a set of functions that start with `contr.`. For example, `contr.poly(5)` will create the contrast matrix (similar to the one presented in Table 1) for a factor with five levels. These matrices then can be assigned to a factor variable through the `contrast()` function. For example,

```
> contrast(ses) <- contr.poly(3)
```

sets the contrast for the factor `ses` to polynomial contrasts.

In fact, you can use any k by k − 1 matrix as contrasts. Of course, it will be useful only if the resulting comparisons makes sense. As said at the beginning, we will not cover contrasts in much detail here. However, before concluding, we will experiment with another well-known contrasts, *Helmert contrasts*, which are used to compare each level of a factor to the previous one. Helmert contrasts in R can be obtained with the function `contr.helmert()`.

Exercise 9.10. Re-order the `ses.ordered` factor from `low` to `high`.

Set the contrasts for the `ses.ordered` to *Helmert* contrasts. Fit a linear regression model predicting `l2score` from `ses.ordered` with the new contrasts.

Compare your results with the model with the treatment contrasts.

▷

In closing, we note that in all the contrast experiments we did, the main (ANOVA) results are the same. The reason for setting contrasts is to facilitate the hypothesis tests performed with a linear model for different purposes. Contrasts that obey certain criteria are called *orthogonal* contrasts. Orthogonal contrasts are safe against the multiple comparisons problem we discussed in Section 4.

For the rest of this tutorial we stick to the default (treatment) contrasts.

9.3 Mixing categorical and numeric predictors

As we just saw, we can convert a categorical variable with k levels to k − 1 indicator variables, and use multiple linear regression to estimate a linear predictor. This also opens up the possibility of using both categorical and numeric predictors in the same model. Traditionally this sort of analyses would be called *analysis of covariance* (ANCOVA).

Given our formula notation, there is nothing interesting for fitting mixed-predictor models. We just specify all predictors, categorical or numeric, on the right hand side of the formula.

Exercise 9.11. Fit a linear model predicting `l2score` from `gender` and `gpa` (without interaction). Summarize and interpret your results.

Compare the results with a model where `gender` is the only predictor. How do you explain the change in the coefficient estimates of `gender` between these two models? ▷

We can conceptualize the model we fit in Exercise 9.11 above as having two intercepts, one for girls and one for boys, resulting in two regression lines with the same slope.

Exercise 9.12. Plot `l2score` against `gpa`, using the gender symbols ♀ and ♂, and colors red and blue for points corresponding to girls and boys respectively. (use other symbols if your R environment does not support Unicode characters).

Plot two lines over the scatter plot with the same slope and appropriate intercepts for girls and boys based on the estimations from Exercise 9.11. ▷

If we specify interaction between a categorical and a numeric variable, we can conceptualize it as fitting separate intercepts and slopes for all levels of the factor variable.

[3] Note that this is different than specifying the order the levels of an ordinary factor variable. The levels of an unordered factor variable can only be checked for (un)equality. On the other hand, the `ordered` factor variable has a defined ordering. So, one can compare ordered factors with operators like `>`, `<`, `<=` etc.

Exercise 9.13. Fit a linear model predicting `l2score` from `gender` and `gpa` without interaction. Summarize and interpret your results.

▷

Exercise 9.14. Repeat the Exercise 9.12 with varying intercepts and slopes estimated in Exercise 9.13. ▷

Exercise 9.15. Fit two separate linear models predicting `l2score` from `gender`, one only using the data corresponding to `gender=F`, and the other one for `gender=M`.

Compare the coefficient estimates with the ones from Exercise 9.13. ▷

Exercise 9.16. Fit a linear model predicting `l2score` from all other variables in the `l2` data set, except `subject`. Include all interaction terms (also interactions between the numeric predictors). We will refer to this model for the rest of this section as `m.full`.

1. Does the model fit to the data well?
2. Which coefficient estimates are statistically significant?
3. Is the overall model fit statistically significant? How do you interpret that the inference for the overall model, considering the inference for the individual coefficients?

▷

Exercise 9.17. Starting from the `m.full` from Exercise 9.16, do a stepwise elimination of the variables.

What does `step()` return as the best model?

How many variables are attempted for removal at the first step? (and why?) ▷

Exercise 9.18. Do another stepwise analysis, but this time start from a model with no predictors (the model of the mean), and allow addition and removal of the variables at each step The `scope` should be equal to the predictors of the `m.full` defined in Exercise 9.16.

Compare the AIC value with the one suggested by the `step()` in Exercise 9.17. Which procedure returns you the best model? Is this the best possible model with these predictors?

▷

Exercise 9.19. Produce the standard diagnostic plots for the best model identified by `step()` in Exercise 9.18.

1. Do you see any sign of non-linearity?
2. Is there any evidence for non-constant variance?
3. Are the residuals distributed approximately normally?
4. Are there any influential observations?

▷

The following exercise is useless in practice. However, understanding the model fit, and the consequences will help you understand the models we will discuss in later sections.

Exercise 9.20. Fit a model predicting `l2score` using `subject` as the only predictor. What does the coefficient estimates mean in this model? Can you explain the r^2 and the inference about the coefficient estimates and the overall model? ▷

10 Probability distributions

This section walks you through a set of utilities R provides for working with probability distributions. Probability distributions underlie all statistical analyses we perform. In most cases you will not use these utilities directly, R provides functions to do the analysis you are interested in without requiring to work with the distribution functions. Nevertheless, it is important to understand the concepts behind the analyses you are performing. If you read this section and do the exercises here, you will refresh your memory about probability distributions, and get a few additional tips and tricks about using R.

The probability distributions we have already mentioned directly or indirectly include *normal* (or *Gaussian*) distribution, *Student's t* distribution, *F* distribution. There are many other theoretical distribution that are interesting for statistical analysis. Most data in real life comes in the form of one distribution or another, and in statistics we often assume that the data comes from a certain distribution, typically the normal distribution. Even in non-parametric tests, where we do not assume that the data is distributed according to a theoretical distribution, we use the fact that a relevant statistic is distributed (roughly) according to a well-known probability distribution.

For each distribution it knows about, R provides four functions that may come handy at times.

d The functions that start with `d` are the *probability density* functions. The value of these functions indicate the likelihood of observing the given data point.

For example `dnorm(1.96)` will give you value of the density function at 1.96. You should remember that the values of density functions of continuous distributions, like `dnorm()` or `dt()`, are not probability values.

p The functions that start with `p` are the *cumulative distribution* functions (CDF). Cumulative distribution functions return the *probability* of observing a value lower than the given data point. The value of the CDF correspond to the area under the distribution function up to the given value. For example, `pnorm(-1.96)` will give you the total probability of observing a value less than or equal to -1.96, given the data is distributed according to standard normal distribution.

The CDFs are the way of obtaining the *p-values*. You can use these functions instead of the p-value tables that decorate inner covers or appendixes of statistics textbooks.

q The functions that start with `q` are the *quantile* functions. The quantile function of a distribution is the inverse of its CDF. In other words, given a probability p, the quantile function returns x where probability of observing a value less than or equal x is p.

If you are lost with the explanation don't worry (yet), the following example may help. Assume that we want a p-value of 0.025, for values distributed with standard normal distribution. If you run the command `qnorm(0.025)` in R, you will get the value of the variable for which you would obtain a p-value of 0.05 (in a two-tailed test).

r The functions that start with `r` are sampling functions. They return a vector of specified size that is sampled randomly from a given distribution. For example, `rnorm(10)` will

produce 10 random numbers that are distributed according to standard normal distribution. The sampling functions are particularly handy for doing simulations.

A probability distribution is specified using a number of *parameter*s. For example normal distribution is typically parametrized by its *mean* and *standard deviation*, and the t-distribution has a single *degrees of freedom* parameter. For example, `rnorm(10, mean=10, sd=5)` will produce 10 random number from normal distribution with mean 10 and standard deviation 5. If a distribution has *standard* parameter values, R will use the standard values if you do not specify the parameters. For normal distribution this is `mean=0` and `sd=1`.

Exercise 10.1. For the normal distribution with $\mu = 3500$ and $\sigma = 200$,

- find the probability that a value from this distribution is less than or equal to 3000.
- find the probability that a value from this distribution is greater than or equal to 4000.
- what is the probability that a value from this distribution is between 3000 and 4000.
- find the value of the density function at 3400. Is this value a probability?
- find the maximum value for which p-value is less than 0.01.
- find the upper and lower bounds for which $p < 0.005$.

▷

Exercise 10.2. Plot the cumulative distribution functions of the standard normal distribution and t-distribution with degrees of freedom 1, 5 and 20 on the same graph. Make sure all functions are plotted in the range -4 to 4 as smooth curves (not points), and choose different colors for each function. Use sensible axis labels, and include a legend to indicate which line belongs to which distribution.

Note that this is similar to the Exercise 6.10 but this time we plot the CDFs instead of the density functions. ▷

The *binomial distribution* characterizes *n* trials of an event with one of two outcomes. One of the outcomes occurs with probability *p*. For example, the binomial distribution with *n=10* and *p=0.5* characterizes number of heads (or tails) you get for 10 flips of a fair coin. Every 10 flips you perform will produce a number between 0 and 10 (more likely 5 than 1 or 9 though). The binomial distribution is not only for coin flips. Many interesting quantities are binomially distributed. Just name a few: whether a sentence is judged 'grammatical' or 'ungrammatical', whether a student passes the exam or not, whether one is diagnosed with dyslexia or not…

Exercise 10.3. Produce a random sample of 200 values from a binomial distribution with $n = 100$ and $p = 0.55$, and plot its histogram. Make sure that the histogram is 'normalized' such that the area under the histogram sums to 1, and modify the axes ranges to contain all possible values. Plot the theoretical density (or more correctly mass) function over the histogram.
▷

Exercise 10.4.

For large samples, it is said that the binomial distribution can be approximated by the normal distribution.

Plot histograms of increasing numbers of samples from from the binomial distribution with parameters $p = 0.5$ and $size = 20$, determine visually what sample size looks like the normal distribution.

Note that in this exercise you are simulating multiple runs of an experiment with fair coin where we count number of heads (or tails) in 20 coin flips. ▷

Exercise 10.5. Repeat Exercise 10.4 with `p=0.9`. Is the number of samples similar to what you have decided in Exercise 10.4?

Once you are convinced that the number gives you an approximately normal distribution, draw the normal distribution with the same mean and standard deviation over the histogram.
TIP: specifying `probability=TRUE` option to `hist()` will produce a histogram with 'relative frequencies' making it comparable to probability the density function.

▷

Another interesting probability distribution that is sometimes used when the data are counts of occurrence of and event (in a fixed time period or location) is called *Poisson distribution*. It has a single parameter λ (lambda) which corresponds to the rate of occurrence of the event.

Exercise 10.6. Draw probability density function for the Poisson distribution with rate parameters 3, 10, and 30 for the range 0 to 50 on the same graph.

▷

Exercise 10.7. Create samples of 10000 items from each of the following distributions:

- standard normal distribution
- t-distribution with 3 degrees of freedom
- F-distribution with degrees of freedom 3 and 10
- log-normal distribution with standard parameters
- Poisson distribution with rate parameter 3
- binomial distribution with `size=10`, `p=0.1`

Plot normal Q-Q plots for each distribution on separate graphs on the same canvas.

Repeat the exercise for only 20 (instead of 10000) samples from each distribution.

This exercise will give you a better idea of how non-normally distributed data looks like on a Q-Q plot.

▷

11 Logistic Regression

Logistic regression is a member of the *generalized linear models* (GLMs),[4] a generalization of linear regression to case

[4] Not to be confused with *general* linear models of Section 9.

where residuals (error) is not normally distributed. Furthermore, the linear model is related to the actual response through a *link function*. In *logistic regression* we will study here, the link function is the *logit* function, and the residuals are distributed binomially.

We will use two data sets in this part. First, a larger version of a familiar data set, the `seg` data set from Section 6. This data can be found at `http://coltekin.net/cagri/R/data/seg-large.csv`.

Second, we will use another data set related to language acquisition. The data contains information on 'overregularized' past tense verb forms in CHILDES recordings. The observations (lines in the data) correspond to recording sessions. For each session, the data contains the age of the child in months (`age`), the number of times the child utters irregular verbs in past form in (`n.past`) and number of times that form is overregularized (`n.or`). We are interested in predicting the probability (or rate) of overregularizations from a child's age. You can get the full data set at `http://coltekin.net/cagri/R/data/past-tense-or.csv`.

Again, the problem is real, but the data is fake (The counts of past tense forms are from CHILDES, but error rate does not have any empirical basis.). You should not take the conclusions out of this analysis seriously.

Exercise 11.1. Read the CSV file `http://coltekin.net/cagri/R/data/seg-large.csv` into your R environment. Name the resulting data set `seg` (overwriting the earlier one if exists).

Make sure that the variables `utterance` and `phoneme` are `factor` variables.

▷

Exercise 11.2. Read the CSV file `http://coltekin.net/cagri/R/data/past-tense-or.csv` into your R environment. Name the resulting data set `or`.

Add a new variable `correct` to the `or` data frame that contains the rate of correct (not overregularized) production of the past tense forms.

▷

11.1 Regression and binomial response variables

Exercise 11.3. Fit an ordinary least squares regression model predicting the rate of correct past tense production from the age of the child using the data set `or`.

Check the model assumptions through diagnostic plots, and refit a model excluding the most influential data point. Save the resulting model as `or.ols`.

Repeat the diagnostic plots. Do you see any other problems with the model diagnostics? ▷

Exercise 11.4. What is the predicted correct past-tense rate for children at ages 1.5 and 8? Note that `age` in our data is in months. ▷

If you were careful in Exercise 11.4, you should have realized that the model predicts a probability value greater than one. One of the first problems with using ordinary regression to predict probability values is that the probability values are bounded between 0 and 1, while our linear model is happy to predict any value between $-\infty$ and $+\infty$. The first step in logistic regression to relate the regression prediction to our response variable through a function. Instead of predicting the probability value p, we predict $\texttt{logit}(p)$ which is

$$\log\left(\frac{p}{1-p}\right)$$

Since probabilities are between 0 and 1, the quantity in the parentheses above, the *odds*, transform it between 0 and $-\infty$, and taking logarithm of the value expands the range from $-\infty$ and $+\infty$.

Exercise 11.5. Plot the curve of the logit function defined above. ▷

Transforming the response variable with logit is just part of the solution, and we do not normally do the transformation manually. However, just for the demonstration, we will transform the data and fit a ordinary regression model in the next exercise.

Exercise 11.6. Add a new variable, `log.odds`, to the `or` data frame which contains the log odds for the probability of correct responses.

Fit a linear regression model predicting the transformed variable `log.odds` from `age`. Exclude the outlier we identified earlier. Save the model as `or.logit`.

Produce the diagnostic plots and inspect the results. How do you interpret the coefficients? ▷

Exercise 11.7. What are the expected rate of overregularization for children at ages 1 to 10 (years) according to the model fit in Exercise 11.6?

▷

11.2 Binomial data and generalized linear models

The exercise above aims to give you a sense of how logistic regression proceeds. Specifying a non-linear function as our response variable (logit transform) is only part of the solution. The model fit in Exercise 11.6 still assumes that the residuals are normally distributed, which is incorrect for binary or binomial response data. We cannot fix this with `lm()`. The `lm()` function in R fits models with normal residuals. To fit a proper logistic regression, as well as other GLMs, we use the `glm()` function.

The syntax of `glm()` is similar to the syntax of `lm()`. The differences are the specification of the response variable, and the type of GLM (e.g., logistic or count regression) with the `family` parameter. The value of the `famliy` parameter we use with the logistic regression is `binomial`. If no family parameter is specified, the default `gauissian` is used, in which case the `glm()` is equivalent to `lm()`. In logistic regression, we specify our response variable either as binary 0 and 1 values, or as a two column matrix of 'success' and 'failures' for each observation. Since the `or` data fits the second format, Listing 11 presents specification and summary of a logistic regression model using `glm()` using the success/failure notation.

The output should mostly be familiar. The first important difference to keep in mind is that the coefficient estimates are not the estimates of the probabilities, but the logit of the probability values. The hypothesis test performed is a z-test (the

Listing 11: Logistic regression example. The output is edited for clarity.

```
> or.glm <- glm(cbind(n.past-n.or, n.or) ~ age,
            family='binomial', data=or, subset-68)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.93872  -0.54742   0.01848   0.75018   2.31268
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.558637   0.153005   3.651 0.000261
age         0.037836   0.003878   9.756  < 2e-16
---
(Dispersion parameter for binomial family taken
    to be 1)
    Null deviance: 190.765  on 90  degrees of
        freedom
Residual deviance:  93.036  on 89  degrees of
    freedom
AIC: 385.24
Number of Fisher Scoring iterations: 4
```

z-value here is also called Wald's statistic), but the interpretation is the same. The other differences you should have noticed include the frequent use of the term *deviance*, and no sign of r^2 in the output. These are because of the fact that the logistic regression model is estimated using *maximum likelihood estimation* (MLE) rather than *least squares regression* used in ordinary linear regression. The deviance measures the difference between the model of interest, as measured by log likelihood, and a model that fits the data perfectly.[5] The deviance in GLMs is similar to the residual sums of squares in ordinary regression. Smaller deviance indicating better model fit. The AIC we introduced in Section 8 is also reported here. Remember that this is a combination of the model fit (likelihood) and number of parameters in the model, and lower values of AIC indicate better models.

A problem with logistic regression (as well as other GLMs) is *overdispersion*. Overdispersion occurs when error (residuals) are more variable than expected from the theorized distribution. In case of logistic regression, the theorized error distribution is the binomial distribution. The variance of binomial distribution is a function of its mean (or the parameter p). If there is overdispersion, the coefficient estimates will be more confident (smaller standard error values) than they should be. One can detect overdispersion by comparing the residual deviance with the degrees of freedom. If these two numbers are close, there is no overdispersion. Residual variation much larger than degree of freedom indicates overdispersion. Underdispersion, detected by lower residual deviance than the degrees of freedom, is also possible but more rare. In our example in Listing 11, there is no indication of overdispersion.

The last line in Listing 11 indicates the number of iterations used in the MLE estimation. The MLE, unlike least squares estimation, is not an analytic procedure but an iterative optimization procedure. All else being equal, large number of iterations means that the model was difficult to fit, and in some cases `glm()` will give up and announce that the model could not be fit within the maximum number of iterations defined. Sometimes, you will get a result (probably accompanied by a warning) but the estimates be will far from optimum. This is generally evidenced by very large standard errors in comparison to the coefficient estimates.

We will first deal with the overdispersion problem. A simple solution for overdispersion is to estimate an additional parameter indicating the amount of the oversidpersion. With `glm()`, this is done so-called 'quasi' families, i.e., in logistic regression we specify `family=quasibinomial` instead of `binomial`.

Exercise 11.8. Although we concluded that there is no overdispersion with the model presented in Listing 11, fit the same model with `quasibinomial` family, and compare the summary with the model `or.glm` above.

▷

The `predict()` function for GLMs can present you either the value of the linear outcome (logit value in this case), or the actual response value (the probabilities). The fist one is the default, if you want to get the response values, you should specify `type='response'` option of the `predict()` function.

Exercise 11.9. What are the expected rate of overregularization for children at ages 1 to 10 (years) according to the model `or.glm` from Listing 11? ▷

Exercise 11.10. Plot the scatter plot of observed correct past tense rates against age in the `or` data. Plot the curve representing the expected correct past tense rate against the age range in the data using the model `or.glm`. ▷

11.3 Binary data

Another case of binary or binomial response is when we have binary 'success/failure' information for each observation. In the `or` data set we worked with above, the unit of observation has been a recoding session. Alternatively, we could consider every single past tense verb uttered by any child as the unit of observation, and indicate whether it was correct or overregularized (why is this a bad idea?). As a tricky exercise, you can try to expand the `or` data set as binary responses, and fit a GLM and observe the differences. However, we will switch to the `seg` data set for experimenting with binary-response data.

To fit a GLM with a binary response, we simply use our binary response variable (the response variable it can be in different data types, but it should have only two values), and use the `binomial` family with `glm`.

Exercise 11.11. Using the `seg` data set, fit a logistic regression model predicting whether there is a boundary or not for a given `pmi` value. Save the model as `seg.pmi`.

Do you observe overdispersion?

Plot the observed data, and the prediction curve on the same graph. ▷

A measure of fit for a logistic regression model is to check how accurately the model predicts the data. Note that since the predictions of the model are probabilities rather than direct success/failure indications, we need to set a threshold value. Using a threshold of 0.5, i.e., assuming probabilities over 0.5 indicate success, is common in practice. However, there may be cases where a different threshold may be applicable, or in some cases we are not interested in converting probabilities to decisions at all, as in the `or` data set above,

[5] This is called a *saturated* model, it typically includes as many parameters as the number of data points.

Exercise 11.12. Calculate the accuracy (ratio of successful predictions divided by total number of predictions) of boundary predictions by the model fitted in Exercise 11.11.

TIP: You can compare `seg$boundary` with `predict(seg.pmi, type='response')> 0.5`. Note that `type='response'` is important here.

▷

Exercise 11.13. We can view logistic regression as a classification method. For a classifier, we often want to know whether it achieves better than a trivial *baseline*. In most cases the trivial baseline is not just choosing the response randomly, but guessing the majority class. In our segmentation problem, majority class is non-boundary, that is there are more word-internal phoneme pairs than word boundaries.

What is the majority-class baseline accuracy for the `seg` data set?

▷

11.4 Further exercises in model selection

In the exercises above, we have worked with only one predictor for simplicity. The logistic regression, and the GLMs in general, is an extension of the general linear models we studied earlier. As a result, we can use multiple numeric or categorical predictors with the logistic regression as well.

Exercise 11.14. Fit a logistic regression model predicting boundaries from all variables in the `seg` data frame. Make sure that the variables `utterance` and `phoneme` are processed as categorical variables. Name the resulting model `seg.full`. Summarize the results. ▷

Exercise 11.15. Compare the models `glm.pmi` and `glm.full` using `anova()`. Note that if you like to perform hypothesis testing, you should specify `test="Chisq"`, since the difference between the deviances for the two models follows an approximate χ^2 distribution (as opposed to F-distribution we use with `lm()` residuals). ▷

Exercise 11.16. You should see in Exercise 11.15 that the difference between the models `seg.pmi` and `seg.full` is statistically significant, but you should have also realized that the smaller model has much fewer degrees of freedom.

The AIC values (in the model summaries) also indicate that the big model is better. However, some of the variables in the big model may not be necessary.

Perform a stepwise selection starting from the full model `model.full`. Save the model identified by `step()` as `seg.model`. ▷

Exercise 11.17. Calculate the accuracy of the `seg.model` from Exercise 11.15. Compare it with the accuracy of the model `seg.pmi` you calculated in Exercise 11.12.

Is the difference statistically significant? **TIP:** You could either use *Fisher's exact test*, or *Chi-Square Test for Independence* for this purpose. The names of the functions are `fisher.test()` and `chisq.test()` respectively.

▷

Exercise 11.18. A model's predictions are most useful outside the data used to fit the model. Fit another logistic regression model that predicts the boundaries from `pmi`, `h` and `rh` using the first half of the data (utterance numbers 0 through 49). Calculate the accuracy of the model on the first and second half of the data, and compare the results. **TIP:** converting `utterance` back to numeric may be useful here. ▷

Exercise 11.19. Repeat Exercise 11.18 with only a single predictor, `pmi`.

▷

11.5 More on logistic regression and GLMs

In this tutorial we only discuss the binary (or binomial) case. Logistic regression can also be used with a multinomial response, where you have more than two categories, e.g., noun, adjective or adverb. If the response is ordered, multinomial logistic regression can be fit using `polr()` from the `MASS` package. Unordered multinomial logistic regression can be fit using the `mnp` package.

The `lrm()` function from the `rms` package also provides functions for fitting logistic regression models, with some additional options and output with additional statistics (e.g., pseudo r^2 values). The package also contains other useful utilities for fitting and diagnosing GLMs.

12 Multilevel / mixed-effect models

Except for repeated-measures ANOVA, all methods we have studied so far assume independent observations. Often, this is not correct for the data at hand. And sometimes, we do break the independence assumption in data collection, because some questions are answered more directly or naturally by data with some systematic dependence between the observations. In this section, we will study so-called *mixed-effect* or *multilevel* models. The first name implies that some of the effects, the coefficients, which we always assumed to be fixed quantities so far, can be *random*, distributed according to a probability distribution. The second name emphasizes that the (some of the) coefficients are also 'modeled'. That is, coefficients vary according to a model at a different 'level'. The term 'level' here does not necessarily include a strict hierarchy. The use of the first name *mixed-effect models* is more common in (psycho)linguistics literature. However, there is also a large number of valuable resources that refer to this type of models as *multilevel models*. Both, at the end, refer to the same sort of modeling practice we will work on in this section.

We will use three different data sets in this section. Besides two familiar data sets, `bilingual` and `newborn` from Section 5, we will also introduce a new data set.

In our somewhat hypothetical investigation here, would like to know whether parenthetical expressions, *expressions like this one*, behave similar to *intonational phrases* (IP), or *phonological phrases* (PP). The proper definitions of these phrase types and the theoretical motivation behind this investigation are not important for our purposes. We will use the rate at which these different classes of phrases uttered for the comparison here. The data is available as an R data file containing multiple data frames at `http://coltekin.net/cagri/R/data/par.rda`. The problem is inspired by Güneş (2014), but heavily simplified, and the data is modified for these exercises.

Exercise 12.1. Load the data from `http://coltekin.net/cagri/R/data/par.rda`. Make sure that you print and check the object(s) loaded from this file. ▷

You probably realized that the data set loaded in Exercise 12.1 consists of three data frames. The data frame `par.data` contains the individual observations. `par.item` contains information regarding 'items', the different phrases used in the study, and `par.subj` contains information about the participants. This sort of organization is more suitable for storing and organizing your data. It avoids replication of the same information. For example, if we have stored everything in a large data frame (or spreadsheet), we would need to replicate age of a participant for each observation from the same participant. The problem is not the storage (it is plenty and cheap nowadays). However, a data organization with replicated values calls for inconsistency, and it should be avoided while storing and updating the data.

Exercise 12.2. Although the organization of the data in Exercise 12.1 is the correct way of storing the data, The analyses we will perform are more convenient with a single large data frame. The function `merge()` in R can be used to combine two data frames into a single data frame by replicating the values in some columns when necessary. Note that the data frames will be merged based on the columns with the same name on both data frames. In our case, the variables are consistently named. So you would not need to reorganize the data frames, or specify which columns to be used as keys during the mere.

Merge all three data frames into a single data frame with name `par`. Remember from Section 6 that `par` is also a built in function name. This is fine. Different types of objects (a function and a data frame in our case) can have the same name in R. We do this here for demonstration, but you should avoid using the names of built-in R objects to prevent potential confusion. ▷

12.1 Background

We start with a(nother) reminder of the simple regression expressed by the following formula:

$$y_i = a + bx_i + e_i$$

where y is the response variable, x is the predictors, i indexes each observation or data point, a and b are the coefficients of the model, and e is the variation that cannot be explained by the model. In *general linear models*, we assume this error to be normally distributed with zero mean. *Generalized linear models* (GLMs) allow us to work with different error distributions. For example, in logistic regression, e is distributed binomially. However, a crucial assumption of all these models is that e is the only source of random variation. The rest of the model $a + bx$ is deterministic. Hence the coefficients a and b are unknown but fixed quantities.

The mixed-effect models we will study here allow other (more systematic) random sources of variation. This additional source of variation typically occurs due to multiple correlated measurements. For example, in the `newborn` data set we worked with in Section 5, we had multiple measurements of sucking rates from the same baby. Besides the random variation that we cannot account for, this includes a systematic variation due to individuals: some babies will be faster, and some will be slower. In repeated-measures designs, we make use of this variation to estimate the effects of interest more precisely. The mixed-effect models we will study here allow more flexible modeling of additional sources of variation. For example, extending the linear equation above to include variation due to each subject j would result in the following set of equations:

$$y_i = a_{j[i]} + bx_i + e_i$$

$$a_j = \mu_a + \epsilon_j$$

Although this can be written in other forms, the notation necessarily gets complicated with mixed-effect models. Here we introduce a random variation due to each subject j that only affects the intercept. In other words, now we have as many intercept terms as the subjects. Intercept vary between the subjects. The term $a_j[i]$ means the intercept that corresponding to the subject j that the observation i came from. Crucially, the a_j are estimated from all the data, not only from the data that belongs to subject j. As a result, the estimation of a_j will be pulled towards the μ_a. This means for subjects whose intercept is not estimated precisely from their own data (because of small number of observations or high variation), the population estimates will play a role.

The equation above defines a rather simple form of mixed-effect model. In this model we only vary the intercept for each subject. One can also define varying slope(s), in which case our assumption would be that the subjects not only have a different base rate (intercept) but they also have different slopes. In this case, we assume, for example, that the subjects react to experimental manipulation differently. Furthermore, we are not limited to a single source of variation. The mixed-effect models can incorporate multiple sources of variation. For example, subjects and the items (the experimental material, e.g., words, sentences, stories) can be defined as two different sources of variation each having systematic effect on the intercept, slope(s) or both. The mixed-effect models are a generalization of *generalized linear models*, which means you can also fit mixed-effect logistic regression, or any other GLM.

In this section, we will use the `lmer()` function from the `lme4` package for fitting mixed-effect models. To match the `lmer()` output better, we rewrite the equation for the mixed-effect model above as

$$y_i = \mu_a + \epsilon_{j[i]} + bx_i + e_i$$

The model is the same, but `lmer()` syntax follows a more flat representation as in here. Writing it this way allows us to understand `lmer()` output better. Now we have a common intercept for all subjects, but two error terms: one is due to subjects (ϵ) and the other (e) is the unexplained 'residual' variation. Both errors are normally distributed with zero mean for the models we will discuss here, but as noted earlier for mixed-effect GLMs that can have non-normal error terms.

This much of explanation is already too much for a 'hands-on' tutorial. Hopefully, the above gave you a general sense of the theory behind the exercises we will do in this section. You will learn the details while working on the exercises below. At the end of the section we list a few books that cover multilevel or mixed-effect models in a comprehensive but also accessible way.

12.2 Random intercepts

As usual, we will start with a simple case. In Section 5, we worked with a hypothetical experiment where we wanted to see if newborn babies react to their mother's native language differently than a foreign language. The response variable was the sucking rate, and the predictor was the input language. Earlier, we used paired t-test and repeated measures ANOVA for analyzing the same data set. It will also be our first example in mixed-effect models.

Exercise 12.3. Fit an ordinary regression for predicting the effect of input language on the sucking rate using the `newborn` data set. We will refer to this model as `nb.lm`. Note the coefficient estimates and the residual variance. Can you see any parallels with this analysis and the t-test performed in Exercise 5.3? ▷

Exercise 12.4. Plot a scatter plot of sucking rate vs. language in the `newborn` data set, and the fitted regression line over it.

Note that the default plot method for numeric vs. factor is a box plot. For a scatter plot, you need to convert the factor `language` to integer values between 0 and 1 (to match the indicator coding). **TIP:** adding small amount of noise to the data points in the x-axis will make the individual data points more visible. For this, you can use the function `jitter()`, or alternatively, you can add your own noise by random numbers sampled from a uniform distribution.

▷

We know that this analysis is not correct (the observations are not independent). This would only be correct if there would be no differences between the subjects. In other words, our analysis would be correct if all the variation we observe was due to random (or randomized) factors outside the experimental design.

We already hinted that there are multiple mixed-effect models that can be specified for the same the problem. The simplest form of the mixed-effects specification includes a single, 'intercept-only' random effect. In this type of model, we assume that the individuals measured have different 'base rates', but all react to the different conditions (e.g., experimental manipulation) the same way, except for the completely random variation. Hence, the intercept varies per subject, but slope is the same.

To specify a random intercept term in `lmer()` we use a notation like `(1|subject)`. Here we assume that the source of additional variation (the random effect) is the variable `subject`. This term needs to be added to the right side of our model formula. Listing 12 shows the way to fit a random-intercept-only model to the `newborn` data.

The first line in Listing 12 loads the relevant functions from the `lme4` library, and it is needed only once in an R session. The `lmer()` call is similar to `lm()`. Only big difference is the specification of the random intercept term.

Summary of the model fit first includes information about the 'random effects'. Here, the line labeled `participant` refers to the variation due to participants. It corresponds to the error term ϵ in the formula above. The column `Name` indicates that this random effect only affects the intercept. The line labeled `residuals` is the general variation that cannot be accounted for by the model. The `lmer()` summary presents both the estimated variance and the standard deviation (square root of the variance), since standard deviation is easier to interpret. In essence the summary tels us that the estimated intercept varies according to a normal distribution with zero mean and a standard deviation of 9.72. The residual variation is a lot smaller. This is a good case for using mixed-effect modeling (or repeated measures). One last thing note here is that in comparison to the ordinary regression fit from Exercise 12.3, residual variation is reduced a lot.

Listing 12: Random-intercept-only mixed-effects model. Parts of the output is supressed for clarity

```
1 > library(lme4)
2 > nb.lmer1 <- lmer(rate ~ language +
      (1|participant), data=newborn)
3 > summary(nb.lmer1)
4 Linear mixed model fit by REML ['lmerMod']
5 Formula: rate ~ language + (1 | participant)
6 Random effects:
7  Groups      Name        Variance Std.Dev.
8  participant (Intercept) 94.42    9.717
9  Residual                10.87    3.297
10 Number of obs: 60, groups: participant, 30
11 Fixed effects:
12                Estimate Std. Error t value
13 (Intercept)     31.8437     1.8734  16.997
14 languagenative   4.5237     0.8513   5.314
```

Exercise 12.5. Compare the total random variance of the mixed-effect model in Listing 12 with the residual variance of the ordinary linear regression model from Exercise 12.3.

▷

Returning to the fixed effects report in Listing 12, we see that the intercept is estimated as 31.84 and the slope of the language effect is estimated as 4.52. These correspond to mean sucking rate while listening to stories in the foreign language, and the rate difference between foreign and native languages, respectively. These estimates are almost the same as the estimates from the ordinary regression fit from Exercise 12.3. This is a coincidence, due to simple (and fabricated) the data set. The important difference lies in the standard error of the slope estimate. Compared to the ordinary regression estimate, we now have a much smaller standard error. Using the rough approximation $\pm 2 \times$SE, the 95% confidence interval for the slope estimate is $[2.8211, 6.2263]$. The difference from zero will certainly be statistically significant. However, `lmer()` summary does not include a p-value. We also do not see the r^2 or an F-test for the overall model fit. As in generalized linear models, due to the estimation method(s) used to fit mixed-effect models, we do not have straightforward ways to calculate these. For now, we will use approximate $\pm 2 \times$SE confidence intervals to asses the reliability of the coefficient estimates, and we will return to these problems later.

We said earlier that for the random-intercept model in Listing 12, each subject is assigned to a different intercept. However, the summary of the `lmer()` fit gives us only one intercept estimate in the fixed effects, which corresponds to the mean of the estimated intercepts. We only get the standard deviation of the estimated subject intercepts in the `summary()` output. In most cases, this is what we are interested in, we are interested in overall trends in the data, not the individual behavior. In some cases, on the other hand, investigating random effect

estimates may be crucial, and without any doubt, it is instrumental for learning what the mixed-effect models do.

To obtain the estimated random coefficients we use the function `ranef()`. The intercept value corresponding to a particular participant is the sum of the estimated random intercept and the mean intercept value reported in the fixed-effects section of the summary. To obtain only the fixed effects, you can use the function `fixef()`. The `coef()` function we used earlier with `lm()` returns both fixed and random effects.

Exercise 12.6. The `dotplot()` function from `lattice` package provides a convenient way to plot the random effects. Plot the random-intercept estimates of `nb.lmer1` from Listing 12 using `dotplot()`.

TIP: if you specify `condVar=T` to `ranef()`, it will return (co)variance information together with the random effects which will be picked up by `dotplot()` and it will plot 95% confidence intervals along with the estimated random effects.

And a note: the `lattice` library includes many useful, easy-to-use plotting commands multivariate data that you may want to explore further (see Sarkar 2008, for more information). ▷

Exercise 12.7. What is the estimated sucking rates during exposure to the foreign and the native language stimuli of participant 1 and 3 according to the model `nb.lmer1` from Listing 12?

Are these estimates different from the observed values?

▷

Exercise 12.8. Repeat Exercise 12.4, creating the scatter plot and the estimated regression line from the ordinary regression fit (`nb.lm`). Also plot regression lines for subjects 1 and 3 according to the model `nb.lmer1` from Listing 12.

Use different colors, and re-plot (without jitter) the data points belonging to these two participants as filled circles with the color of the corresponding regression line.

Make sure that the x-axis does not contain numeric labels, but two labels `foreign` and `native` at the corresponding values. **TIP:** for this you can first prevent drawing the x-axis with `xaxt='n'` during `plot()`, and then, you can use the function `axis()` to create a custom one. ▷

The least squares regression we fitted in Exercise 12.3 ignores the subject variation completely. We will now study another extreme example.

Exercise 12.9. Fit an oridinary regression model predicting `rate` from `language` and `participant` (without interactions). Name your model `nb.lm2`.

Compare your results to earlier models `nb.lmer1` and `nb.lm`.

1. Does intercept and slope estimates (of `langauge`) change?
2. How does the residual variance compare with the earlier models?
3. What is the estimated intercept and slope of `langauge` for participants 1 and 3?

▷

Exercise 12.10. Add the regression lines corresponding to participants 1 and 3 estimated by the model `nb.lm2` on the graph produced in Exercise 12.8. Use dotted lines, but the same color as the corresponding lines from the mixed-effects model estimate. ▷

Although it does not make much sense in our example, the model of the sorts we fit in Exercise 12.9 is not uncommon in practice. The variables that are entered into analyses to reduce the unknown variation are sometimes called blocking variables.

The point of the exercises 12.8 and 12.9 for us is to demonstrate that the estimates of a mixed-effect model will fall in between a model that ignores the subject variation completely, and the one that includes individual subjects as predictors. Including subject and language interaction in the model would be another instructive exercise. However, since our `newborn` data does not contain replication, such a model will be a saturated model, and it will not be useful in practice (you are encouraged to try it, though).

12.3 Random slopes

So far, we worked only with random intercepts. If there is interaction between the random and fixed effects, e.g., if we expect the subjects to react to the experimental manipulation differently, the way to express this is to allow slopes to vary. If we model random slopes, we will almost certainly model random intercepts as well. Varying-slopes-only models are conceivable but uncommon. When we talk about a model with 'random slopes' in this tutorial, it means we also include random intercepts.

Specification of random slopes for `lmer()` is easy. We extend the random effect expression like `(1 + language | participant)`. Note that as in other parts of the formula notation, the intercept is implicit in random effects term as well. The expression `(language | participant)` have the same meaning. If you happen to fit a model that only includes random slopes, you need to specify it as either `(language - 1 | participant)`, or `(language + 0| participant)`.

We will not be able to fit a random-slopes model with our newborn data. If we include both intercepts and slopes, the number of random coefficients will be equal to number of observations. Hence, the model cannot be fitted. We fit our first random-slopes model using the `bilingual` data. Remember that in this data set we are interested in differences in linguistic skills of bilingual children between the language spoken only at home, and the language which is also spoken in the language community, particularly at school. Listing 13 presents `lmer()` summary for predicting the language skills only from the type of language (`home.only` or `school`), using random slopes and intercepts.

The model specification does not need much explanation: we want to predict `mlu` based on `language`, but we also specify `subj` as a source of variation that affects both the slope and the intercept. The resulting model can be described mathematically as follows:

$$y_i = \mu_a + \epsilon_{j[i]} + (\mu_b + \delta_{j[i]}) \times x_i + e_i$$

Before going into the interesting part, we quickly observe that the fixed effects indicate an MLU of 4.3123 for the `home.only` language for an average kid, and on average they

Listing 13: Mixed-effects with random intercepts and slopes. Some output is supressed for clarity

```
> bl.lmer1 <- lmer(mlu ~ language +
    (1+language|subj), data=bilingual)
> summary(bl.lmer1)
Random effects:
 Groups   Name           Variance Std.Dev. Corr
 subj     (Intercept)    0.7576   0.8704
          languageschool 0.0402   0.2005   1.00
 Residual                1.1823   1.0873
Number of obs: 120, groups: subj, 20
Fixed effects:
               Estimate Std. Error t value
(Intercept)      4.3123     0.2400  17.970
languageschool   0.4898     0.2035   2.407
```

show an MLU of $4.3123 + 0.4898 = 4.8021$ for the school language. These correspond to the μ_a and μ_b in our formula above. Our rule of thumb of $\pm 2 \times$SE confidence interval indicates that the MLU difference between languages are statistically significant.

More interesting differences are in the random effects, since we now have both random intercepts and slopes. The individual values of $\epsilon_{j[i]}$ and $\delta_{j[i]}$ are not visible in the summary, but the random effects part indicate that the standard deviations of estimated random effects are 0.8704 and 0.2005 respectively. The residual standard deviation e is 1.0873 in this model fit. `lmer()` also reports the correlation between the two random effects. The correlation between the random intercepts and slopes are reported to be 1. In general, there is nothing wrong with high correlation between random intercepts or slopes. In our case, that would mean that the kids who are more proficient in their home language also show the higher difference between home and school languages (it may be difficult to reason about, but it can happen). However, a perfect or near perfect correlation (close to 1 or -1) indicates problems with fitting the model. One option is to specify a model without random effect correlations, which we will revisit later. Another option, especially after observing that the variance of random slope is close to zero is to check whether it is worth opting for this more complex model instead of random-slopes-only model.

12.4 Random intercepts or random slopes

So far, we have been fitting mixed-effect models with so-called *residual or restricted maximum likelihood* (REML) which is `lmer()` default. This method has some advantages, particularly for estimating variance components, over the *maximum likelihood estimation* (MLE) while fitting mixed-effect models. However, MLE fit provides some additional information such as AIC that we have used for model selection earlier. To fit a mixed-effect model with MLE, you need to specify the option `REML=FALSE`. In most cases, the differences between MLE and REML estimates should be small.

Exercise 12.11. Refit the model `bl.lmer1` from Listing 13 using MLE. Name this model `bl.lmer2`. Fit and another model without random slopes (call it `bl.lmer3`) and compare the AIC values reported for each model.

Which model does AIC prefer? ▷

Another way to compare two models is to test them using `anova()`. In case of models fit by MLE, `anova()` will do a χ^2 test for the differences between log likelihoods of two models. The test result will be statistically significant if the reduction in the log like likelihood is supported by the data.

Exercise 12.12. Compare the models `bl.lmer2` and `bl.mler3` from Exercise 12.11 using `anova()`.

What is your conclusion? ▷

Exercise 12.13. Plot the random intercepts and slopes of the model `bl.lmer2` from Exercise 12.11 using `dotplot()`. Make sure to include the confidence intervals in the plots. The above assumes that you have already typed `library(lattice)` to access this function from your current R environment. ▷

12.5 Where are my p-values?

With mixed-effect models, it is no more straightforward to calculate the p-values for the model coefficients. The difficulty has to do with calculating degrees of freedom for the t-values calculated from the coefficient standard errors. In most cases, especially if you have enough data, $\pm 2 \times$SE will give a crude but useful approximation to 95% confidence intervals for a coefficient estimate. Other more precise alternatives include,

- comparing models with and without the coefficient using `anova()`. If the models are significantly different, then we conclude that the 'effect' is real.
- calculating so-called 'profile-based' confidence intervals (we will not discuss these in detail here, see Bates (2010) for the details). Once you have confidence interval estimates, you can announce statistical significance at the corresponding level if the interval does not contain 0.
- another alternative to calculate simulation-based p-values. With older version of `lme4`, this was made easy with `pvals.fnc()` from `languageR` package by Baayen (2008).
- other packages such as `mixed` and `lmerTest` deals with calculating p-values from mixed-effect models fit by `lmer()`.

We will work only with the first two.

Exercise 12.14. this exercise we return to the `newborn` data set, and want to test whether the input language makes a statistically significant difference on sucking rate of babies. We fitted a mixed-effect model in Listing 12 for this data. Fit and alternative model excluding the input language as predictor, and compare the models to asses the effect of the input language.

Reminder: To fit a model without any predictors (i.e., only with an intercept), we use the syntax `response ~ 1`. Of course, you will need to specify the random-effects term for your new model to be comparable to the model `nb.lmer1` fitted earlier.

TIP: to make sure the likelihood-ratio test performed makes sense, you should normally (re)fit the models with the MLE. However, `anova()` will do it for you if your models were fit with REML.

▷

You can obtain 'profile' of a model with `profile()` command. The model profile can be inspected for inspecting the coefficient estimates. Here, we will only use the profiles to get the confidence intervals. To get the confidence intervals from a profile, all you need to do is to run the function `confint()` on the profile object. Although you can perform hypothesis tests at any significance level, it makes more sense to report the confidence intervals directly. This is particularly true for random effects whose confidence intervals tends to be asymmetric.

Exercise 12.15. Using profile-based confidence intervals, check whether the effect of input language is statistically significant at α-levels 0.05 and 0.01?

Does the random effect due to the participants statistically significant at the same levels?

TIP: 95% confidence intervals are the default for `confint()`. To obtain 99% confidence intervals, you need to specify `level=0.99`. ▷

12.6 Multiple fixed and random effects

As in factorial repeated measures ANOVA, we can have multiple fixed effects in an mixed-effect model. The specification of the fixed effects are straightforward in the model formula.

Exercise 12.16. In Section 5, we analyzed the bilingual data in a 3×2 repeated ANOVA design, where the response were measured for each level of two variables: language (`home.only` or `school`) and age of the children (`preschool`, `firstgrade` and `secondgrade`).

Fit a mixed effect model with random intercepts and slopes. (The repeated measure ANOVA corresponds to a mixed-effect model with both random intercepts and slopes.)

- Decide for the best random effects configuration based on AIC values. Note that not all random effect structures are possible to fit with `lmer()`. In some cases `lmer()` will fail to converge and indicate this in its output.
- Determine the significance of the fixed-effects at level $\alpha = 0.05$ for your final model.

▷

In our bilingual data, the bilingual kids were measured on both languages, and different ages. In repeated measures ANOVA terms, both `language` and `age` are *within-subjects* variables. As in ANOVA, we can also analyze 'mixed' data between- and within- subjects predictors with mixed-effect models. In multilevel modeling literature, a between-subjects predictor, such as `gender` in our `bilingual` data set, is associated with a different level in the hierarchy. In the formula notation used with `lmer()` you do not need to distinguish the between- or within-subject variables. This will be apparent in the data organization. Of course, you do not specify random slopes for such predictor if you are fitting a model with random slopes.

Exercise 12.17. Add `gender` as an additional predictor to your best model from Exercise 12.16.

Does gender have a significant effect (at $\alpha = 0.05$)? ▷

12.7 Crossed random effects

In the data sets we have been working with in this section, we have only one sensible random effect (due to the participants). Mixed-effect models has been advocated in (psycho)linguistic research, particularly for their capability of handling crossed, non-hierarchical random effects. The problem, named 'language-as-a-fixed-effect fallacy' after Clark 1973, arises because in a typical 'repeated measures' psycholinguistics study one does not only repeat the measurements over the participants, but also over the items, e.g., phrases, words, sentences and maybe pictures, that are used as experimental material. A typical repeated measures ANOVA generalizes to the population the participants are sampled from, but not to the language where the items are sampled from. In such an experiment, the subjects and the items are crossed. That is, all subject react to all items. Although there are some approximate solutions, including one suggested by Clark 1973, classical repeated measures methods cannot deal with this sort of crossed random effects in a straightforward way see Baayen 2008; Raaijmakers et al. 1999, for further discussion.

There is nothing special in specification of multiple random effects for `lmer()`. We simply add the additional random effects as before to the formula notation. For example specification of random intercepts both due to subject and item would be specified like `(1|subject)+(1|item)`.

Exercise 12.18. Fit a model model with random intercepts and slopes with subject and item as two sources of variation predicting speech `rate` from `context` using `par` data set. We will call this model `par.m1`.

How do you interpret the fixed and random effects? ▷

Exercise 12.19. Fit the following simplified models.

par.m2 random intercepts due to items and random slopes and intercepts for subjects.

par.m3 random intercepts due to subjects and random slopes and intercepts for items.

par.m4 random intercepts only (due to both subjects and items).

par.m5 random intercepts only due to subjects.

par.m6 random intercepts only due to items.

Pick the best model suggested by `AIC()`. ▷

Exercise 12.20. Add phrase length (`length`) as a fixed effect to model `par.m4` from Exercise 12.19. We will call this model `par.m4a`.

Compare the coefficient estimates with `par.m4`. Which fixed effect estimates (and their inferences) change? Why? ▷

Exercise 12.20 points to a common problem with the estimation of mixed-effect models with numeric predictors. In Exercise 12.20, besides the fact that we have an intercept term with no straightforward interpretation, you should have noticed that the now the estimate is less certain, and there is a large correlation between the intercept and the slope of the `lenght`. These problems are due to the fact that the intercept is now far away from the observations, and can be remedied by centering or scaling the numeric predictor.

Exercise 12.21. The `scale()` function in R centers and scales a its argument. Repeat Exercise 12.20, but add the scaled version of `length`.

How do you interpret the intercept term now? Do you observe improvements in the estimation of the intercept term? ▷

In the models we fitted in the last two exercises, we observe another interesting aspect of the mixed-effect models. In a normal linear model, the phrase id `item` and the phrase length would be collinear. As a result, the coefficient of one of these predictors would not be estimable. In mixed-effect models this is fine, since the predictor `length` is only used for estimating the intercept due to items. In other words, the length is a predictor in a different 'level'. You should also observe this in the reduction of the estimated variance of the random intercept due to `item` (but almost no other changes) with the addition of `length` as a predictor.

Exercise 12.22. Add the fixed predictors `age` and `sex` as predictors to the model `par.m4a` fitted in Exercise 12.21. We will call this model `par.m4b`.

What is the new interpretation of the intercept?

Which random effects has changed? Why? ▷

Exercise 12.23. Use `dotplot()` from the `lattice` library to plot the random intercepts with their confidence intervals due to subjects from models `par.m4` and `par.m4b`, and compare.

Do you observe the reduction in the subject variation in the graphs?

TIP: The argument `which="subject"` to `ranef()` causes to extract only the random effects due to subject from the model.

TIP2: `dotplot()` resets the values set by `par()`. So, plotting these side-by-side is a bit tricky (you will not be tortured with this). ▷

Exercise 12.24. Using profile-based confidence intervals, test whether the effects estimated by the model `par.m4b` are significant at level $\alpha = 0.01$. ▷

12.8 Where to go from here?

Although we try to include some introduction to mixed-effect models in this section, our aim in this tutorial, as usual, is not to be comprehensive. Here we mention a few books that you may want to consult for learning more about multilevel models, or possibly, study along with the exercises here, especially if you are using this tutorial for self-study. A rather accessible and comprehensive textbook on mixed-effect or multilevel modeling is Gelman and Hill (2007). Another 'must-read' book, if you will be using mixed-effect models in practice is Bates (2010). This book is written by the author of the `lme4` package, includes many practical examples with clear explanations, and also available online (see the references at the end for the URL). The focus of Baayen (2008, ch. 7) is also mixed-effect models, discussed through examples from linguistic data.

References

Baayen, R. H. (2008). *Analyzing Linguistic Data: A Practical Introduction to Statistics Using R*. Cambridge University Press.

Bates, D. M. (2010). *lme4: Mixed-Effects Modeling with R*. Prepublication version at: `http://lme4.r-forge.r-project.org/book/`. New York: Springer (in preparation).

Clark, H. H. (1973). "The language-as-fixed-effect fallacy: A critique of language statistics in psychological research." In: *Journal of Verbal Learning and Verbal Behavior* 12, pp. 335–359.

Çöltekin, Ç. (2011). "Catching Words in a Stream of Speech: Computational simulations of segmenting transcribed child-directed speech." PhD thesis. University of Groningen.

Fox, J. and S. Weisberg (2011). *An R Companion to Applied Regression*. Second. Thousand Oaks CA: Sage.

Gelman, A. and J. Hill (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Güneş, G. (2014). "Constraints on syntax-prosody correspondence: The case of clausal and subclausal parentheticals in Turkish." In: *Lingua* 150, pp. 278–314. ISSN: 0024-3841.

Mehl, M. R., S. Vazire, N. Ramírez-Esparza, R. B. Slatcher, and J. W. Pennebaker (2007). "Are Women Really More Talkative Than Men?" In: *Science* 317.5834, p. 82.

Nazzi, T., J. Bertoncini, and J. Mehler (1998). "Language discrimination by newborns: Toward an understanding of the role of rhythm." In: *Journal of Experimental Psychology: Human Perception and Performance* 24.3, pp. 756–766.

Nerbonne, J. (2010). "Measuring the Diffusion of Linguistic Change." In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 365. Special issue papers from "Cultural and Linguistic Diversity", conference held at AHRC Centre for Evolution of Cultural Diversity, London, Dec. 9-13, 2008, pp. 3821–3828.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Raaijmakers, J. G. W., J. M. C. Schrijnemakers, and F. Gremmen (1999). "How to Deal with "The Language-as-Fixed-Effect Fallacy": Common Misconceptions and Alternative Solutions." In: *Journal of Memory and Language* 41, pp. 416–426.

Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. Use R! Springer. ISBN: 9780387759692.

A Answers

A.1 Starting R and finding your way around

Answer of 1.1.

```
> ??'multivariate anova'
stats::manova           Multivariate Analysis of
    Variance
stats::summary.manova   Summary Method for
    Multivariate
                        Analysis of Variance
```

So, `manova()` should be the function we are looking for. Now it's time to type `?manova` and learn how to use it. Note that part of the output is trimmed for readability.

Answer of 1.2.

The answer may change depending on the version of R and the packages loaded in the R environment. On my system if I press tab after the third letter (the initial segment `sha`) the full function name is displayed. And, after `sh` pressing tab twice gives me a list of 9 commands.

Answer of 1.3.

Here is a way to do it:

```
1 > x = 20
2 > m = 10
3 > 5 = s
4 > t <- x - m
5 > t / s -> z
6 > z
7 [1] 2
```

Note the different assignment operators, `<-` and `->`, in lines 4 and 5. We have already mentioned `<-`. `->` is similar, but you place the target variable to the right of `->`, which is sometime handy when you start writing a complex expression and what to store the result in a variable.

Answer of 1.4.

```
z = (x - m) / s
```

Thing to note is that you need parentheses around the subtraction operation for correct interpretation. What is the result without parentheses?

Answer of 1.5.

First we search using the keyword 'logarithm'

```
> ??logarithm
base::log       Logarithms and Exponentials
nlme::logDet    Extract the Logarithm of the
    Determinant
```

Some additional information is excluded from this listing. In this case R finds two matches. We are interested in the first one. The notation `base::log` tels that there is a `log` function in package `base`. We will study use of packages later. It is reassuring that it is in `base` package. Which means it is directly accessible.

Now we know the function name, we calculate the logarithm of `2.7`:

```
> log(2.7)
[1] 0.9932518
```

Answer of 1.6.

If you have tried `help(log)` or `?log` (or `?base::log`), you see that you can give a second parameter as base of the logarithm. Furthermore, R gives you 'shortcut' functions for common bases like, `log2()` and `log10()`. We will use the first method:

```
> log(2.7, base=2)
[1] 1.432959
```

We could just type `log(2.7, 2)`, since the value is the first and `base` is the second argument. The notation we use specifies the `base` parameter explicitly, so that you do not need to remember the order of the parameters. This notation is particularly useful for well known parameters (like `mean` or `sd`) of other commonly used functions as well.

Answer of 1.7.

```
> ls()
[1] "m"  "s" "t"  "x"  "z"
> rm(t,x)
> ls()
[1] "m"  "s" "z"
```

Here is a tip for deleting all variables without needing to list all files (use with caution!):

```
> rm(list=ls())
> ls()
character(0)
```

Answer of 1.8.

```
> sd(nwords)/sqrt(length(nwords))
[1] 7.485096
```

The `sqrt(x)` function we used here is an alternative to `x^0.5`.

Answer of 1.9.

```
1 > znwords <- (nwords - mean(nwords)) /sd(nwords)
2 > znwords; mean(znwords); sd(znwords)
3 [1] -0.06759625 -0.15209156 -1.84199776 0.35488030
4 [5]  0.18588968  0.56611858 -0.27883452 0.31263265
5 [9]  1.96029119 -1.03929231
6 [1] 3.852463e-15
7 [1] 1
```

On line 2, we specified multiple commands on a single line, separating them with semicolons ';'. You could, of course, run each command on a separate line.

Note that R displays the mean on line 6 using the scientific notation. This means 3.86×10^{-15}, a very small number (but should it not be 0?).

Answer of 1.10.

```
> sort(c(36,35,8,71), decreasing=T)
[1] 71 36 35  8
```

Another side note here: `T` is a shorthand for `TRUE`. It is unlikely to make a difference for most purposes, but `TRUE` is a language reserved keyword, and `T` is just a convenience variable. If you are interested, the following proves the point:

```
> T
[1] TRUE
> T = FALSE
> T
[1] FALSE
> TRUE = FALSE
Error in TRUE = FALSE :
        invalid (do_set) left-hand side to
            assignment
```

If you try these, do not forget to `rm(T)` at the end of your trial. Otherwise, at some point later, you may have a hard time understanding 'why things stopped working as they used to be'.

Answer of 1.11.

```
> seats = c(36,35,8,71)
> (seats / sum(seats)) * 100
[1] 24.000000 23.333333  5.333333 47.333333
> round((sort(seats, dec=T) / sum(seats)) * 100, 2)
[1] 47.33 24.00 23.33  5.33
```

Line 3 gives the answer, but line 4 uses the sorted list and rounds the result to two significant digits after dot, which makes it easier to read.

Answer of 1.12.

```
> wdiff <- c(6,8,6,5,7,5,9,7,10,9)
```

Answer of 1.13.

```
> nwords2 = nwords
> nwords = nwords - wdiff
> nwords; nwords2
 [1] 3497 3495 3456 3506 3503 3515 3486 3504 3541
     3469
 [1] 3510 3508 3468 3520 3516 3525 3505 3519 3558
     3487
```

Note, again, the use of multiple commands on the same line. This time it is particularly useful, since it allows easier comparisons of the (short) vectors.

Answer of 1.14.

```
> mean(wdiff)
[1] 7.2
> mean(nwords2) - mean(nwords)
[1] 7.2
```

A.2 Basic data exploration and inference

Answer of 2.1.

```
> stem(words)
  The decimal point is 4 digit(s) to the right of
      the |

  0 | 255666
  1 | 0113445778
  2 | 011
  3 | 2
```

The value displayed on the last line (corresponding to the person who spoke 31955 words) seem to be quite far away from the others.

Answer of 2.2.

```
> hist(words)
```

you should see the histogram on a separate window. For now, we will only display our graphs on screen without much makeup. R allows you to control every aspect of your graphs, and you can save graphs from R in many different formats. We will frequently revisit graphs later in this tutorial.

Answer of 2.3.

```
> boxplot(words)
```

Answer of 2.4.

```
> boxplot(words.f, words.m)
```

Answer of 2.5.

```
> cor(words, age)
[1] -0.2604937
```

We have a negative correlation, indicating people speak less with age. The correlation is not strong, you should interpret the strength of the correlation with respect to the problem at hand, but typically absolute values less than 0.5 would not be regarded as a strong correlation.

Answer of 2.6.

```
> cor(words.m, words.f)
[1] -0.3212523
```

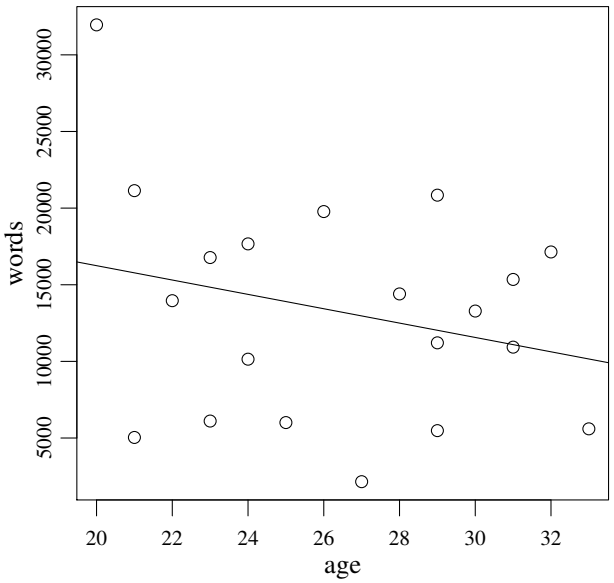
There is no reason for these two values to be correlated (unless sampling is done horribly wrong, and order of the numbers reflect it somehow). Best explanation for the above correlation coefficient is 'chance'. And we will be discussing inference for correlation later.

Answer of 2.7.

```
> plot(age, words)
```

The convention is to put the predictor on the x-axis and the response variable on the y-axis. Presumably, we are interested in effect of age on talkativeness (unless you hypothesize that talking keeps you young). We should put `age` on the x-axis and `words` on the y-axis.

Answer of 2.8. Here is the result, including the scatter plot:



The regression line indeed agrees, it indicates that as age increases, people become less talkative. In fact, the correlation coefficient, Pearson's r, is simply normalized simple regression slope. However, as we will stress later, least squares regression (and also correlation) is sensitive to extreme values. In our example, it seems the regression line is affected (possibly substantially) by the youngest and most talkative participant.

You may want to remove this data point (or replace with a more moderate value) and try last three exercises again. You are encoraged to exercise with removing or replacing data points as you like here. However, when you remove data points in 'real' research, you should be convinced (and be ready to convince others) that it is a sensible thing to do.

Answer of 2.9.

```
> plot(words.m, words.f)
abline(lm(words.f ~ words.m))
```

Answer of 2.10.

```
> se.words <- sd(words)/sqrt(length(words))
> se.words
[1] 1620.478
```

Answer of 2.11.

```
> mean(words) - 2 * se.words
[1] 10007.14
> mean(words) + 2 * se.words
[1] 16489.06
```

or a more practical notation:

```
> mean(words) - c(-2, 2) * se.words
[1] 16489.06 10007.14
```

The second calculation may be difficult to grasp at this point. However, it shows clearly how R treats the vectors.

Answer of 2.12.

```
> mean(words) + qnorm(0.025) * se.words
[1] 10072.02
> mean(words) + qnorm(0.975) * se.words
[1] 16424.18
```

Or simplifying it similar to the above, both for normal and t distributions:

```
> mean(words) + qnorm(c(0.025,0.975)) * se.words
[1] 10072.02 16424.18
> mean(words) + qt(c(0.025,0.975), df=19) *
    se.words
[1]  9856.401 16639.799
```

The confidence interval calculated using the t distribution is larger (less certain, or more conservative if testing for hypotheses). As expected, 'multiply by ± 2' approximation yields a slightly larger interval than the intervals calculated using the normal distribution.

Answer of 2.13. The confidence intervals calculated in Exercise 2.12 includes 16000. As a result, we cannot reject the null hypothesis (that there is no statistically significant difference).

Answer of 2.14.

```
> t.test(words, mu=16000)
        One Sample t-test
data:  words
t = -1.6982, df = 19, p-value = 0.1058
alternative hypothesis: true mean is not equal to
    16000
95 percent confidence interval:
  9856.401 16639.799
sample estimates:
mean of x
  13248.1
```

Not surprisingly, the difference is not significant at 0.05 significance level (p-value = 0.1058). Also note that the confidence interval reported by `t.test()` is the same as the confidence interval you calculated using the t distribution above.

Answer of 2.15. Again, we simply do a single-sample t test.

```
> t.test(words.f, mu=20000)
        One Sample t-test
data:  words.f
t = -4.2857, df = 9, p-value = 0.002033
alternative hypothesis: true mean is not equal to
    20000
95 percent confidence interval:
  9590.798 16783.202
sample estimates:
mean of x
    13187
```

This time we have a small p-value. So we can reject the hypothesis that population mean is 20000.

Answer of 2.16. We can already tell by looking at the mean values (in our sample, mean value for men is larger than that of women). However, we do the test nevertheless:

```
> t.test(words.f, words.m, alternative='greater')
        Welch Two Sample t-test
data:  words.f and words.m
t = -0.0367, df = 13.889, p-value = 0.5144
alternative hypothesis: true difference in means
            is greater than 0
95 percent confidence interval:
 -5990.058       Inf
sample estimates:
mean of x mean of y
  13187.0   13309.2
```

The above tests whether we can reject the *null hypothesis*, $\mu_f - \mu_m \leq 0$ in the population. Since p-value is (much) greater than 0.05, we can not reject the null hypothesis and can not conclude that 'women talk more'.

Two notes on the above output: First, the confidence interval reported is for the difference of means (as expected it includes 0). Second, R reports that this is a 'Welch Two Sample t-test' instead of calling the test simply 'independent samples t-test'. This is due to the fact that `t.test()` function by default applies a correction for unequal variances. You can force `t.test()` to assume equal variances (see `help(t.test)`).

A.3 Linear regression: a first introduction

Answer of 3.1.

```
> plot(mot~chi, data=mlu)
```

or, alternatively

```
> plot(mlu$chi, mlu$mot)
```

Answer of 3.2. No answer yet.

Answer of 3.3. Assuming you have decided the best line is with intercept 5.5 and slope 0.25,

```
> abline(5.5, 0.25, col="red", lwd=2)
```

Answer of 3.4. You should get an intercept of 5.7133 and a slope of 0.1503 from the `lm()` output. You can either draw it using these numbers,

```
> abline(5.7133, 0.1503, col="blue")
```

or let `abline()` extract a and b from the `lm()` output:

```
> abline(lm(mot ~ chi, data = mlu), col="blue")
```

Answer of 3.5.

```
> cor(mlu$chi, mlu$mot)^2
[1] 0.1272399
```

Not surprisingly this is the R^2 value reported by `lm()`.

Answer of 3.6. Histogram can be produced by:

```
> hist(resid(lm(mot~chi, data=mlu)))
```

and Q-Q plot, including the theoretical line:

```
> qqnorm(resid(lm(mot~chi, data=mlu)))
> qqline(resid(lm(mot~chi, data=mlu)))
```

Interpreting these diagnostic plots require some experience. In summary: we see some deviances from the normality, but in general it is normal to see this in small data sets.

Answer of 3.7. First the summary of the linear regression analysis:

```
 1 > summary(lm(age~chi, data=mlu))
 2 Call: lm(formula = age ~ chi, data = mlu)
 3 Residuals:
 4     Min      1Q  Median      3Q     Max
 5 -3.2881 -0.8938 -0.0091  0.8140  2.9785
 6 Coefficients:
 7             Estimate Std. Error t value Pr(>|t|)
 8 (Intercept)  13.2892     1.1795   11.27 1.32e-10
 9 chi           8.7177     0.4254   20.49 8.00e-16
10 ---
11 Residual standard error: 1.613 on 22 degrees of
       freedom
12 Multiple R-squared:  0.9502, Adjusted R-squared:
       0.948
13 F-statistic:   420 on 1 and 22 DF,  p-value:
       7.999e-16
```

1. Since we take age as unquestionable measure of the linguistic competence, and want to predict it using on `mlu`. Our response variable is age and the predictor is `chi`.
2. Intercept is 13.2892, and the slope is 8.7177. The intercept corresponds to the age (remember, in months) of a child who has 0 MLU. Although you should approach cautiously, it does not seem to be a very bad estimate. The slope means that each unit increase in MLU is associated with 8.7 months.
3. The commands `plot(age~chi, data=mlu)` and `abline(lm(age~chi, data=mlu))` should do the trick.
4. The p-value presented on line 9 in the listing above is very small, and it is definitely statistically significant.
5. The plot (not given here, you should really produce it) does not indicate any outliers.
6. As before you should use `resid()` to extract the outliers, and check using `qqnorm()`, and possibly with `shappiro.test()`. If we are looking at the same pictures, the graph and the test should not indicate non-normality.

A.4 Linear models with categorical predictors

Answer of 4.1.

```
> talk = data.frame(age=age, words=words)
```

Answer of 4.2.

```
> str(mlu.ses)
'data.frame':   60 obs. of  4 variables:
 $ subject: int  1 2 3 4 5 6 7 8 9 10 ...
 $ ses    : Factor w/ 3 levels "low","medium",..:
     1 1 1 ...
 $ gender : Factor w/ 2 levels "female","male": 2
     2 2 ...
 $ mlu    : num  1.81 1.91 1.35 2.84 2.54 2.92 1.2
     ...
```

The output looks fine, except, as we will see in the next Exercise, the `subject` variable is identified as an `int` (integer), although it is a categorical variable and should have been a `factor` variable.

Answer of 4.3.

```
> mlu.ses$subject = factor(mlu.ses$subject)
```

We convert the integer values in `mlu$subject` to `factor`, and store the result again in `mlu$subject`.

Answer of 4.4. Here are a few ways to do it:

```
> mlu.ses[mlu.ses$gender == 'male', 4]
> mlu.ses[mlu.ses$gender == 'male', 'mlu']
> mlu.ses$mlu[mlu.ses$gender == 'male']
```

Answer of 4.5.

```
> hist(talk$words[talk$gender == 'F'])
> hist(talk$words[talk$gender == 'M'])
```

Answer of 4.6.

```
> qqnorm(talk$words[talk$gender == 'F'])
> qqline(talk$words[talk$gender == 'F'])
> qqnorm(talk$words[talk$gender == 'M'])
> qqline(talk$words[talk$gender == 'M'])
```

Answer of 4.7. Here is the result only for men:

```
> shapiro.test(talk$words[talk$gender == 'M'])
        Shapiro-Wilk normality test
data:  talk$words[talk$gender == "M"]
W = 0.929, p-value = 0.438
```

which indicates that we have no evidence for non-normality (remember that the null hypothesis is 'the distribution is normal'). The same should hold for women.

Answer of 4.8.

```
> t.test(words~gender, data=talk, var.equal=T,
+        alternative='greater')
        Two Sample t-test
data:  words by gender
t = -0.0367, df = 18, p-value = 0.5144
alternative hypothesis: true difference in means is
                        greater than 0
95 percent confidence interval:
 -5896.008       Inf
sample estimates:
mean in group F mean in group M
        13187.0         13309.2
```

The p-value is the same up to four decimal points. So, the correction for unequal variances does not seem to affect the test results.

The degrees of freedom above is as expected, we have 20 subjects, we calculate two mean values from it, leaving us with 18 degrees of freedom. The interesting bit is the DF with the Welch correction, which was 13.889. Knowing that a t distribution with smaller degrees of freedom will have heavier tails, the correction seems to reduce the DF to force a more conservative test (although it does not make much difference in our case).

Answer of 4.9.

```
> boxplot(words ~ gender, data=talk)
```

You should see rather a large difference (variation for men is larger).

Answer of 4.10. We could use a command like `var.test(talk$words[talk$gender == 'F'], talk$words[talk$gender == 'M'])`, but `var.test()` accepts the formula notation which is neater:

```
> var.test(words ~ gender, data=talk)
        F test to compare two variances
data:  words by gender
F = 0.2953, num df = 9, denom df = 9,
p-value = 0.08358
alternative hypothesis: true ratio of variances is
    not equal to 1
95 percent confidence interval:
 0.07333846 1.18871596
sample estimates:
ratio of variances
         0.2952602
```

Despite the fact that variance form men is about three times larger than the variance for women, we get a p value of 0.08358, we cannot rule out the possibility that the variances differ by chance at conventional significance (α) levels.

Answer of 4.11.

```
> wilcox.test(words ~ gender, data=talk,
    alternative='greater')
        Wilcoxon rank sum test
data:  words by gender
W = 53, p-value = 0.4267
alternative hypothesis: true location shift is
    greater than 0
```

Nothing surprising here. Similar to the t tests, we cannot reject the null hypothesis.

Answer of 4.12. Here is how to generate the box plot:

```
> boxplot(mlu ~ ses, data=mlu.ses)
```

1. Compared to the other two between `high` SES seems to have a high variation, but it does not look extremely different. (You may want to verify this with a `var.test()`)
2. All box plots show some skew, but again, this may not necessarily indicate non-normality.
3. From the box plots the `high` and `medium` groups seems similar, but different from `low`. Box plots cannot answer significance questions reliably, but it seems there is a large difference between `low` and the others, and depending on sample size, it is likely to be statistically significant.

Answer of 4.13. You can run three separate commands, namely

```
> shapiro.test(mlu.ses$mlu[mlu.ses$ses == 'low'])
> shapiro.test(mlu.ses$mlu[mlu.ses$ses ==
    'medium'])
> shapiro.test(mlu.ses$mlu[mlu.ses$ses == 'high'])
```

However, R provides mechanisms to ease the repetitions like this one. Here is an example:

```
> by(mlu.ses$mlu, mlu.ses$ses, shapiro.test)
mlu.ses$ses: low
        Shapiro-Wilk normality test
data:  dd[x, ]
W = 0.9242, p-value = 0.1192
------------------------------------
mlu.ses$ses: medium
        Shapiro-Wilk normality test
data:  dd[x, ]
W = 0.942, p-value = 0.2614
------------------------------------
mlu.ses$ses: high
        Shapiro-Wilk normality test
data:  dd[x, ]
W = 0.9721, p-value = 0.798
```

`by()` partitions its first argument (`mlu` here) set based on the levels of its second argument (`ses` here), and applies the function given in the last argument to each group. None of the above tests suggests a significant divergence from normality.

Answer of 4.14. Here is ANOVA results on this data

```
> summary.aov(lm(words ~ gender, data=talk))
            Df    Sum Sq  Mean Sq F value Pr(>F)
gender       1     74664    74664   0.001  0.971
Residuals   18 997785410 55432523
```

And t tests repeated here for ease of comparison

```
> t.test(words ~ gender, data=talk, var.equal=T)
        Two Sample t-test
data:  words by gender
t = -0.0367, df = 18, p-value = 0.9711
alternative hypothesis: true difference in means
    is not equal to 0
95 percent confidence interval:
 -7117.515  6873.115
sample estimates:
mean in group F mean in group M
        13187.0         13309.2
```

No surprises, the p-value found by both tests are the same.

And, No, directional tests is not possible in the main ANOVA (one can do some directional post-hoc comparison).

Answer of 4.15. Here are multiple comparisons using both methods:

```
> pairwise.t.test(mlu.ses$mlu, mlu.ses$ses)
        Pairwise comparisons using t tests with
            pooled SD
data:  mlu.ses$mlu and mlu.ses$ses
       low     medium
medium 1.2e-06 -
high   1.8e-06 0.83
P value adjustment method: holm
```

```
> pairwise.t.test(mlu.ses$mlu, mlu.ses$ses,
    p.adjust.method='bonf')
        Pairwise comparisons using t tests with
            pooled SD
data:  mlu.ses$mlu and mlu.ses$ses
       low     medium
medium 1.2e-06 -
high   2.7e-06 1
P value adjustment method: bonferroni
```

Notice that all p-values except the largest difference are larger with the Bonferroni correction (which indicates that it is more conservative).

Answer of 4.16.

```
> summary.aov(lm(mlu ~ ses + gender, data=mlu.ses))
            Df Sum Sq Mean Sq F value   Pr(>F)
ses          2 20.714  10.357  22.159 8.14e-08
gender       1  1.852   1.852   3.961   0.0514
Residuals   56 26.173   0.467
```

Answer of 4.17.

```
> summary.aov(lm(mlu ~ ses * gender, data=mlu.ses))
            Df Sum Sq Mean Sq F value  Pr(>F)
ses          2 20.714  10.357  23.047 5.8e-08 ***
gender       1  1.852   1.852   4.120  0.0473 *
ses:gender   2  1.907   0.954   2.122  0.1297
Residuals   54 24.266   0.449
```

Answer of 4.18.

```
> interaction.plot(mlu.ses$gender, mlu.ses$ses,
    mlu.ses$mlu)
```

We see some interaction patterns. The lines for `high` and `medium` cross, and girls perform better within these two groups, boys perform better in the low-SES group. However, the interaction term `ses:gender` above is not statistically significant. We do not have enough evidence in support of any of the interaction patterns we observe.

Answer of 4.19.

```
> kruskal.test(mlu~ses, data=mlu.ses)
        Kruskal-Wallis rank sum test
data:  mlu by ses
Kruskal-Wallis chi-squared = 24.3514,
               df = 2, p-value = 5.154e-06
```

We again find a statistically significant effect of `ses` on `mlu`. As expected, the p-value found here is larger than the one found using ANOVA. Which indicates that the non-parametric test is likely to be more conservative.

Answer of 4.20.

```
> t.test(words ~ gender, data=talk, var.equal=T)
        Two Sample t-test
data:  words by gender
t = -0.0367, df = 18, p-value = 0.9711
alternative hypothesis: true difference in means
    is not equal to 0
95 percent confidence interval:
 -7117.515  6873.115
sample estimates:
mean in group F mean in group M
        13187.0         13309.2
```

Answer of 4.21.

```
> summary(lm(words ~ gender, data=talk))
Call: lm(formula = words ~ gender, data = talk)
Residuals:
     Min       1Q   Median       3Q      Max
-11157.2  -7184.8    374.4   4084.7  18645.8
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  13187.0     2354.4   5.601 2.58e-05
genderM        122.2     3329.6   0.037    0.971
Residual standard error: 7445 on 18 degrees of
    freedom
Multiple R-squared:  7.482e-05, Adjusted
    R-squared:  -0.05548
F-statistic: 0.001347 on 1 and 18 DF,  p-value:
    0.9711
```

Note that the p-value for the slope is the p-value we found in the t test.

A.5 Repeated measures

Answer of 5.1.

```
> bilingual <- read.delim('http://coltekin.net/
    cagri/R/data/bilingual.txt')
```

Answer of 5.2. You should realize that the variable `subj` is of type `integer` rather than a `factor`. To covert it:

```
> bilingual$subj = factor(bilingual$subj)
```

Another useful but not strictly necessary conversion is from factor to factor, such that the levels are ordered in a meaningful way:

```
> bilingual$age <- factor(bilingual$age,
    levels=c('preschool','firstgrade','secondgrade'))
```

R orders the levels alphabetically by default. The ordering here makes sure that the data levels show up in a reasonable while displaying in graphics or as text.

Answer of 5.3. Here is the full listing of the test.

```
> t.test(rate ~ language, data=newborn)
        Welch Two Sample t-test
data:  rate by language
t = -1.7074, df = 57.994, p-value = 0.0931
alternative hypothesis: true difference in means
    is not equal to 0
95 percent confidence interval:
 -9.8271059  0.7797726
sample estimates:
mean in group foreign  mean in group native
             31.84367              36.36733
```

Two normal Q-Q plots, one for each language, and `var.test()` testing for equivalence of variances would be ways to test for normality and homogeneity of variance assumptions. A `boxplot()` is also useful for visualizing the distributions of both groups. However, as in this case, it may be very difficult or impossible to see the effects of assumption of independence.

Answer of 5.4.

```
> t.test(rate ~ language, data=newborn, paired=T)
        Paired t-test
data:  rate by language
t = -5.3138, df = 29, p-value = 1.06e-05
alternative hypothesis: true difference in means
    is not equal to 0
95 percent confidence interval:
 -6.264775 -2.782559
sample estimates:
mean of the differences
              -4.523667
```

Nothing surprising, the paired test is a lot more sensitive, resulting in a smaller p-value.

Answer of 5.5.

```
> qqnorm(newborn$rate[newborn$language ==
    'native'] -
          newborn$rate[newborn$language ==
              'foreign'])
```

The point to remember is that the paired test compares the 'mean of the differences' to 0, not *differences of the means* as in the independent samples test.

Answer of 5.6.

```
> boxplot(rate ~ language, data=newborn)
```

You should see some difference, it is rather small, and the distributions largely overlap. What we see includes the individual variation which clouds the real paired differences. To demonstrate we can plot,

```
> boxplot(newborn$rate[newborn$language ==
    'native']
        - newborn$rate[newborn$language ==
            'foreign'])
```

Note that here, we are interested whether the mean of the distribution visualized by the single box plot is 0 or not.

Answer of 5.7.

```
> summary(aov(rate ~ language, data=newborn))
```

Answer of 5.8.

```
> summary(aov(rate ~ language +
        Error(participant/language), data=newborn))
```

Answer of 5.9.

```
> summary(aov(mlu ~ language*age +
        Error(subj/(language*age)),
            data=bilingual))
```

Answer of 5.10.

```
> interaction.plot(bilingual$age,
    bilingual$language,
    bilingual$mlu)
```

Answer of 5.11.

```
> summary(aov(mlu ~ language*age*gender +
    Error(subj/(language*age)), data=bilingual))
```

Note that the only difference from the purely within-subjects syntax is that the between-subjects variable does not go into the `Error()` term.

A.6 Graphics

Answer of 6.1. There is nothing special with loading this CSV file:

```
> read.csv('http://coltekin.net/cagri/R/data/seg.
    csv')
```

The data types you see (e.g., when you check with `str()`) should be in general sensible, but two of them are questionable. Depending on your analysis `utterance` may be integer (numeric), or categorical (factor). Similarly, having `phoneme` as a factor variable is fine, in some cases you may want to convert it to to a `character` string. For the rest of the exercises here, there is no need to convert these data types, but for the sake of exercise here is how we convert them.

```
> seg$utterance <- factor(seg$utterance)
> seg$phoneme <- as.character(seg$phoneme)
```

Answer of 6.2.

```
> plot(seg$h[seg$utterance == '1'])
```

Answer of 6.3.

```
> plot(h ~ pmi, data=seg, col='red')
> abline(lm(h ~ pmi, data=seg), col='blue')
```

Answer of 6.4.

```
> plot(seg[,4:7])
```

or more clear notation with the expense of some typing:

```
> plot(seg[,c('pmi', 'h', 'rh', 'sv')])
```

Answer of 6.5.

```
> hist(seg$pmi)
> hist(seg$h)
```

Answer of 6.6.

```
> qqnorm(seg$pmi);qqline(seg$pmi)
> qqnorm(seg$h);qqline(seg$h)
```

The PMI values should look approximately normal, while the H values should show a serious divergence from normality.

Note that you can run two or more commands at once by separating each command by a semicolon '`;`' on the same line.

Answer of 6.7.

```
> boxplot(pmi ~ boundary, data=seg)
```

Answer of 6.8. Assuming we the variable `x` from the above listing is in your R environment,

```
> plot(x, dt(x, df=5), type='l', col='green',
    lwd=3)
```

Answer of 6.9.

```
> x <- seq(-4,4,by=0.1)
> plot(x, dnorm(x), type='l', col='blue')
> lines(x, dt(x, df=5), col='green')
```

Answer of 6.10.

```
> plot(x, dnorm(x), type='l')
> lines(x, dt(x, df=1), col=2, lty=2)
> lines(x, dt(x, df=5), col=3, lty=3)
> lines(x, dt(x, df=20), col=4, lty=4)
```

Answer of 6.11.

```
> plot(seg$h[seg$utterance == '1'], pch=19)
```

Answer of 6.12. It may take a bit of experimenting, but here is an example:

```
> plot(x, dnorm(x), type='l', col='blue')
> lines(x, dt(x, df=5), col='green')
> grid()
> text(0, 0.4, 'standard normal', pos=3,
    col='blue', offset=-0.05)
> text(0, 0.35, 't(5)', col='green')
```

Note that the `grid()` function plots a grid, which is helpful for deciding the coordinates of the points on the graph.

Answer of 6.13.

```
> y <- seg$h[seg$utterance == '1']
> x <- 1:length(y)
> plot(x, y, type='l', lty='dotted')
> text(x, y ,
      labels=seg$phoneme[seg$utterance == '1'],
      pos=3,
      col=c('blue', 'red')[1 +
          seg$boundary[seg$utterance == '1']])
```

Answer of 6.14.

```
> legend('topright',
         c('normal', 't(1)', 't(5)', 't(20)'),
         lty=1:4,
         col=1:4)
```

Answer of 6.15. Only listing the first `plot()` command, the other are not affected:

```
> plot(x, dnorm(x), type='l',
       main='normal and t  distributions',
       ylab='density',
       xlab='')
```

Answer of 6.16.

```
> plot(h ~ pmi, data=seg, xlim=c(0,max(pmi)))
```

should do. However, if you want to be certain without peeking into the data:

```
> plot(h ~ pmi, data=seg,
    xlim=c(min(c(0, pmi)), max(c(0, pmi))),
    ylim=c(min(c(0, h)), max(c(0, h))))
```

Answer of 6.17. Again, only listing the first `plot()` command, the other are not affected:

```
> plot(pmi ~ h, data=seg, subset =(boundary == T),
       pch='+', col='red',
       main='PMI vs. H',
       xlab='PMI',
       ylab='H')
> points(pmi ~ h, data=seg, subset =(boundary ==
    F),
       pch='-', col='blue')
> legend('bottomleft', c('boundary', 'word
    internal'),
        pch=c('+', '-'),
        col=c('red', 'blue'))
```

Answer of 6.18.

```
> par(mfrow=c(2,2))
> boxplot(pmi ~ boundary, main='PMI', data=seg)
> boxplot(h ~ boundary, main='H', data=seg)
> boxplot(rh ~ boundary, main='RH', data=seg)
> boxplot(sv ~ boundary, main='SV', data=seg)
```

Answer of 6.19.

```
> pdf(file='noramlity-check.pdf', width=6.27,
    height=3)
> par(mfrow=c(1,2))
> hist(seg$pmi, main='Histogram of PMI',
    xlab='PMI')
> qqnorm(seg$pmi, pch=24, col='blue', bg='blue')
> qqline(seg$pmi)
> dev.off()
```

Answer of 6.20.

Here is only the sequences of commands to produce the PNG with resolution 1024x512.

```
> par(mfrow=c(1,2))
> png('mlu-ttr-1024x512.png', width=1024,
    height=512)
> boxplot(chi.mlu, mot.mlu, names=c('child',
    'mother'), main='Mean length of utterance')
> boxplot(chi.ttr, mot.ttr, names=c('child',
    'mother'), main='Type/token ratio')
> dev.off()
```

If you use bitmap graphics a lot, you'll realize that they scale poorly on higher resolution medium, particularly on paper. If you create bigger images, and scale them when necessary, the text on the graphs will typically become poorly readable, or even unreadable. To optimize for the intended medium, you also need to specify the resolution (`res`). By default bitmap graphics are screen optimized (not for new fancy Retina Displays, for good old 72dpi screens). You should change this to match the printer when you need to use them on paper (for printing, you should really use vector graphics, unless there is a good reason for using bitmap).

Answer of 6.21.

```
> x <- c(0,1,2,4)
> y <- c(0,1,3,4)
> plot(x, y, type='l')
```

Answer of 6.22.

```
> pie(c(36,35,8,71),
    labels=c('A', 'B', 'C', 'D'))
```

Answer of 6.23.

```
> barplot(c(36,35,8,71),
          names.arg=c('A', 'B', 'C', 'D'))
```

Answer of 6.24.

```
> x <- seq(-pi, pi, by=0.1)
> plot(x, sin(x), type='l', col='blue')
> lines(x, cos(x), col='red')
```

Answer of 6.25. First, a not-so-correct attempt:

```
> abline(a=0,b=0)
> abline(a=0,b=10^50)
```

The horizontal line drawn first is straightforward, intercept and slope are both 0. However, the way we drew the vertical line is a bit of cheating. The slope of a vertical line is undefined (well, if infinity is easier to deal with for you, we can also agree on positive infinity). As a result, we just put a very big number with the hope that approximation is fine enough. However, for horizontal and vertical lines, `abline()` provides a simpler and neater alternative,

```
> abline(h=0)
> abline(v=0)
```

Answer of 6.26. Although the actual code that generates Figure 1 is slightly different, the following produces a very similar figure.

```
> par(mfrow=c(1,2), mar=c(1,1,3,1))
> plot(0,
      type='n',
      xlim=c(0,2), ylim=c(0,2),
      xaxt='n', yaxt='n',
      xlab='', ylab='',
      main='(a) par(mfrow=c(2,2))',
      col.main='blue')
> abline(h=1)
> abline(v=1)
> text(c(0.5,1.5,0.5,1.5),c(1.5,1.5,0.5,0.5),
    c('1', '2','3', '4'),cex=9)
> plot(0,
      type='n',
      xlim=c(0,2), ylim=c(0,2),
      xaxt='n', yaxt='n',
      xlab='', ylab='',
      main='(b) par(mfcol=c(2,2))',
      col.main='blue')
> abline(h=1)
> abline(v=1)
> text(c(0.5,0.5,1.5,1.5),c(1.5,0.5,1.5,0.5),
    c('1', '2','3', '4'),cex=9)
```

Two things that is worth noticing above are: (1) the `mar` graphical option that adjusts (shrinks compared to the defaults in this case) the graphics margins, and (2) the `cex` options that adjusts the font size.

A.7 Regression again

Answer of 7.1.

```
> print(load(
  url('http://coltekin.net/cagri/R/data/ling-geo.rda')
  ))
```

Answer of 7.2.

```
> cor(seg$pmi, seg$h)
> cor(seg$pmi, seg$h, method='kendall')
> cor(seg$pmi, seg$h, method='spearman')
```

Nothing surprising here: all indicate that the measures are negatively correlated (`pmi` is a measure of cooccurrence, while `h` is a measure of surprise), and non-parametric tests indicate lower correlation. To check we can reliable use Pearson's r, we need to check its assumptions which we will leave to the regression analysis.

Answer of 7.3. Again, nothing special.

```
> cor.test(seg$pmi, seg$h)
> cor.test(seg$pmi, seg$h, method='kendall')
> cor.test(seg$pmi, seg$h, method='spearman')
```

Note, however, the non-parametric tests gives you approximate p-values because of ties (the data points with the `pmi` or `h` values resulting in the same rank).

Answer of 7.4.

```
> cor(seg[,-c(1,2)])
```

Answer of 7.5. It is easy to obtain the summary, although it may take a while since our data set is rather large:

```
> summary(lm(ling ~ geo, data=lg))
```

1. The slope is highly significant indicating, although it is very small in the original scale, the effect very unlikely to be due to chance effects.
2. The intercept is the expected linguistic difference in the same location (when geographic distance is 0km).
3. The r^2 indicates about 37% of the variance in the linguistic differences to be due to geographic distance.

Note, however, these conclusions are not viable unless we check the assumptions of the model.

Answer of 7.6. The first one is easy:

```
> m <- lm(mot ~ chi, data=mlu, subset=-20)
```

For the second one, you can calculate `mean()` or `median()` of `mlu$chi`, and find find the row index of the closest value in the data frame. The following solution does is in a slightly subtle way, that you should try to understand:

```
> m <- lm(mot ~ chi, data=mlu,
      subset=(chi != sort(chi)[sort(chi) >
          median(chi)][1]))
```

When the outlier is removed, the model coefficients show a visible change. In our example you should see an intercept estimate of 5.54 and a slope estimate of 0.23, as opposed to 5.71 and 0.15 respectively in the full model. When we remove the non-influential (middle) data point, we get almost the same results as the complete model. Furthermore, the model fit (r^2) becomes much better when you remove an influential data point.

Answer of 7.7.

```
> par(mfrow=c(2,2))
> plot(lm(ling ~ geo, data=lg), pch='.',
    col='gray')
```

1. The 'residuals vs. fitted' graph should indicate a clear sign of non-linearity.
2. You should also be observing in 'residuals vs. fitted' graph that variance is reduced as fitted values increase.
3. The normal Q-Q plot of the residuals also show a clear pattern of divergence on tails. For a smaller data set this might not be a clear indication, but for large data sets, like the one we use here, this is clearly a concern.
4. Not really, the 'residuals vs. leverage' plot does not even include the 0.5 Cook's distance contour line we observed earlier. Although there are points with higher leverage and large residuals, for large data sets, an individual observation can seldom be very influential.

Answer of 7.8.

```
> png(file='diagnostics.png', width=1024,
    height=768)
> par(mfrow=c(2,2))
> plot(lm(ling ~ geo, data=lg), pch='.',
    col='gray')
> dev.off()
```

Answer of 7.9.

```
> lg$log.geo <- log(lg$geo)
```

Answer of 7.10. The commands to use are the same as before. You should find an intercept of about −0.03, and a slope of 0.06, again with highly significant estimates.

1. The conclusions do not change drastically. However, you should observe an increase in the r^2 which is an indication of a better model fit.
2. In all graphs you should see some improvements. Particularly, no clear sign non-linearity, and variance should look more constant across the range of the predicted values.

Answer of 7.11.

```
> plot(mot ~ chi, data=mlu)
> m <- lm(mot ~ chi, data=mlu)
> abline(m, col='red')
> seq(min(mlu$chi), max(mlu$chi), length.out=20)
    -> x
> y <- predict(m, newdata=data.frame(chi=x),
    interval='conf')
> lines(x, y[,'lwr'], col='red', lty='dashed')
> lines(x, y[,'upr'], col='red', lty='dashed')
```

Answer of 7.12.

```
> plot(ling ~ geo, data=lg, col='gray', pch='.',
    xlab='Geographic distance (km)',
    ylab='Linguistic difference')
> m <- lm(ling ~ geo, data=lg)
> mlog <- lm(ling ~ log.geo, data=lg)
> lines(lg$geo, predict(m), col='red')
> lines(lg$geo, predict(mlog), col='blue',
    lty='dashed')
```

```
> legend('bottomright',
    c('no transformation', 'log transformed'),
    col=c('red', 'blue'),
    lty=c('solid', 'dashed'))
```

Note that the line and the curve plotted by `lines()` commands above works fine since the `lg$geo` values in this data set is sorted. Otherwise, we would either need to order x and y values, or alternatively, we could get predictions for new data points along the x axis.

Answer of 7.13. The answer is left as an exercise, noting that this exercises is almost the same as Exercise 7.11, except you need to be careful not to use the same x values for prediction and plotting.

What you should observe is a very tight confidence interval all around the prediction curve that it is not possible to see three separate lines.

A.8 Multiple Regression

Answer of 8.1.

```
> load(
  url('http://coltekin.net/cagri/R/data/tv.rda'),
  verbose=T
  )
```

Answer of 8.2. Only the commands (without the output) is given below.

```
> lm(cdi ~ tv.hours, data=tv) -> m1
> summary(m1)
> par(mfrow=c(2,2))
> plot(m1)
> par(mfrow=c(1,1))
> plot(cdi ~ tv.hours, data=tv)
> abline(m1)
> x <- min(tv$tv.hours):max(tv$tv.hours)
> y <- predict(m1, newdata=data.frame(tv.hours=x),
    interval='conf')
> lines(x, y[,'lwr'], lty='dashed', col='blue')
> lines(x, y[,'upr'], lty='dashed', col='blue')
> y <- predict(m1, newdata=data.frame(tv.hours=x),
    interval='pred')
> lines(x, y[,'upr'], lty='dotted', col='green')
> lines(x, y[,'lwr'], lty='dotted', col='green')
```

Answer of 8.4.

```
> m3 <- lm(cdi ~ tv.hours + mot.education, data =
    tv)
```

The slope estimates of the multiple regression model is different than the corresponding single predictor models. we also see that the estimates are slightly less certain in the multiple regression model (smaller t-, larger p-values). Furthermore, r^2, the variation explained by the multiple regression model is less than (about 10%) the sum of the variations explained by the individual predictors.

The explanation for all comes from collinearity. The predictors are correlated, hence, they share the part of the variation explained by each other (so, r^2's do not sum up). This also means that the regression estimation cannot assign the credit (or blame) for some the explained variation (so, variability and lower confidence in the parameter estimates).

Answer of 8.5.

```
> par(mfrow=c(1,2))
> plot(cdi ~ tv.hours, data=tv)
> abline(m1)
> abline(a=coef(m3)[1], b=coef(m3)[2], col='red')
> plot(cdi ~ mot.education, data=tv)
> abline(m2)
> abline(a=coef(m3)[1], b=coef(m3)[3], col='red')
```

Answer of 8.6.

```
> x <- seq(min(tv$tv.hours),max(tv$tv.hours),
          length.out=20)
> y <-
    seq(min(tv$mot.education),max(tv$mot.education),
          length.out=20)
> plot(0, type='n',
       xlim=range(x), xlab="TV time",
       ylim=range(y), ylab="Mother's education")
> new.data=data.frame(tv.hours=rep(x,20),
                      mot.education=rep(y,
                          each=20))
> z <- predict(m3, newdata=new.data)
> gray.levels <- 1 - ((z-min(z)) / (max(z)-min(z)))
> points(new.data$tv.hours,
         new.data$mot.education,
         pch=22, bg=gray(gray.levels), cex=3)
```

Answer of 8.7.

```
> cor(tv$tv.hours, tv$mot.education)
[1] -0.2920446
```

This is not an alarmingly large correlation. In fact, there is no clear threshold after which you should get alerted. In general, the larger the correlation, the higher the expected effect of the collinearity. Furthermore, with more predictors, correlation is not an adequate tool for detecting multicollinearity, since the predictor of interest may correlate with a linear combination of more than one of the other predictors. We will later discuss other ways of detecting multicollinearity.

It is not a complete coincidence that if you square the above correlation coefficient will be closer to the difference between the sum of the r^2 from `m1` and `m2` and the multiple regression model `m3`.

Answer of 8.8.

```
> summary(lm(cdi ~  tv.hours + mot.education +
    I(tv.hours + mot.education), data=tv))
```

In the output, you should see that the coefficient for the predictor `I(tv.hours + mot.education)` cannot be estimated, and indicated by `NA`, which is a special value used for missing data.

Answer of 8.9.

```
> vif(m3)
     tv.hours mot.education
     1.093243      1.093243
```

Note that since we have only two predictors, both have the same VIF value. This is simply because r^2 is the square of the correlation coefficient, and correlation is symmetric.

For two-predictor case, we could alternatively calculate it with

```
> 1/(1-cor(tv$tv.hours, tv$mot.education)^2)
[1] 1.093243
```

Answer of 8.10.

```
> tv$rnd.1 <- runif(80, 0, 1)
> tv$rnd.2 <- runif(80, 0, 1)
> summary(lm(cdi ~ tv.hours + mot.education
                + rnd.1 + rnd.2, data=tv))
[...]
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   99.29090    0.82075 120.976  < 2e-16
tv.hours      -0.11671    0.03047  -3.831 0.000264
```

```
mot.education  0.13000    0.03475   3.741 0.000357
rnd.1         -0.32504    0.39126  -0.831 0.408758
rnd.2          0.37200    0.44865   0.829 0.409645

Residual standard error: 1.057 on 75 degrees of
    freedom
Multiple R-squared: 0.3557, Adjusted R-squared:
    0.3213
F-statistic: 10.35 on 4 and 75 DF,  p-value:
    9.951e-07
```

As expected, the effects of the new variables are non-significant. The values you get will be different since our predictors are randomly generated.

Remember that the model without `rnd.1` and `rnd.2` had $r^2 = 0.3444$ and $\bar{r}^2 = 0.3274$.

Answer of 8.11. The update command is:

```
> update(m3, . ~ . + daycare.hours) -> m4
```

The model fit is slightly better. The increase in r^2 is expected since it will increase with any additional predictor. However since $\bar{r}^2$ is also (slightly) higher, the increase in r^2 is probably not just a chance effect.

The summary (not presented above) should tell you that the effect of the new predictor is not statistically significant in this model (given other predictors).

Answer of 8.12.

```
> m0 <- lm(cdi ~ 1, data=tv)
```

The single coefficient estimated (intercept) should be the same as (`mean(tv$cdi)`).

The standard error of the intercept that you can see from the summary of the model is the standard error of the mean. (You are encouraged to calculate this manually and compare).

Answer of 8.13.

```
> anova(m0, m1)
Analysis of Variance Table

Model 1: cdi ~ 1
Model 2: cdi ~ tv.hours
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     79 129.99
2     78 100.33  1    29.662 23.061 7.438e-06 ***
```

The addition of the predictor causes a statistically significant reduction in the mean residual sums of squares. The F-test reported in summary of a linear regression model is effectively the same test.

Answer of 8.14.

```
> anova(m1, m3)
Analysis of Variance Table
Model 1: cdi ~ tv.hours
Model 2: cdi ~ tv.hours + mot.education
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     78 100.325
2     77  85.222  1    15.104 13.647 0.0004106 ***
> anova(m3, m4)
Analysis of Variance Table
Model 1: cdi ~ tv.hours + mot.education
Model 2: cdi ~ tv.hours + mot.education +
    daycare.hours
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     77 85.222
2     76 83.971  1    1.2503 1.1316 0.2908
```

The first ANOVA indicates that the amount of reduction in residual sum of squares (15.104) is unlikely to be by chance. F-test yields a very low p-value. The second one, on the other hand, indicates that the small reduction in the residual sums of squares due to the new variable `daycare.hours` is not statistically significant.

Answer of 8.15.

```
> anova(m4)
```

You should check which F-values match with the F-values from the earlier exercises.

Answer of 8.16.

```
> AIC(m1,m3,m4)
   df      AIC
m1  3 251.1416
m3  4 240.0884
m4  5 240.9060
```

We observe a decreased in AIC by adding `mot.education` to `m1` (`m3` is better). However, adding `daycare.hours` increases the AIC value. The best model among the ones we compared according to AIC is `m3`.

Answer of 8.17. `step(m4)` will present you the steps taken, and the choice made at the end. Note that the direction of serach (backward, forward or both) may change the result.

A.9 Probability distributions

Answer of 10.1.

```
> pnorm(3000, mean=3500, sd=200)
[1] 0.006209665
> 1 - pnorm(4000, mean=3500, sd=200)
[1] 0.006209665
> pnorm(4000, mean=3500, sd=200) - pnorm(3000,
    mean=3500, sd=200)
[1] 0.9875807
> dnorm(3400, mean=3500, sd=200)
[1] 0.001760327
> qnorm(0.01, mean=3500, sd=200)
[1] 3034.73
> qnorm(0.005, mean=3500, sd=200); qnorm(1-0.005,
    mean=3500, sd=200)
[1] 2984.834
[1] 4015.166
```

Answer of 10.2.

```
> x <- seq(-4, 4, by=0.1)
> plot(x, pnorm(x), type='l', ylab='CDF(x)')
> lines(x, pt(x, df=3), col='red')
> lines(x, pt(x, df=10), col='blue')
> lines(x, pt(x, df=30), col='orange')
> legend('bottomright',
    c('normal', 't(2)', 't(10)', 't(30)'),
    col=c('black', 'red', 'blue', 'orange'),
    lty='solid')
```

Answer of 10.3.

```
> sample <- rbinom(200, size=100, p=0.55)
> hist(sample, probability=T,
+      xlim=c(0,100), ylim=c(0,0.1))
> curve(dbinom(x, size=100, p=0.55),
+       add=T, xlim=c(0,100))
```

A few points to note here:

- If you repeat the exercise, you will get a (slightly) different histogram. Like other '**r**' functions, `rbinom()` produces *random* numbers (although they are distributed according to binomial distribution).
- The parameter `probability=T` tells `hist()` to plot a 'density' histogram instead of counts (or frequencies).

- `xlim=c(0,100)` tells that the x-axis should be in range 0 to 100 (all possible values for a binomial distribution with n=100). Likewise, `ylim=c(0,0.1)` sets the range of the y-axis to make sure that the distribution function we plot next fits into the graph regardless off the differences in the sampling (although, you should work a bit harder if you want to make sure that everything fits into the graph for all possible samples).
- Last, the `curve()` function is an alternative way of plotting smooth curves. Previously we generated x-range manually and use `plot()` or `lines()` to plot corresponding values of a function. For example:

```
> lines(0:100, dbinom(0:100, size=100,
    p=0.55))
```

Answer of 10.4. You should be drawing histograms with

```
> hist(rbinom(N,size=20, p=0.5))
```

for increasing `N`. Your results will differ even for the same value of `N`, since `rbinom()` will produce random samples. But you should clearly observe that as you increase `N`, the histogram resembles more and more like the bell curve.

Answer of 10.5. Compared to Exercise 10.4, you should find it more difficult to convince yourself that the distribution looks normal. Binomial distributions with extreme p parameters will be more skewed, and more difficult to be approximated by the normal distribution.

Nevertheless, here is an example with sample size 100000:

```
> x <- rbinom(100000,size=20, p=0.9)
> hist(x, probability=T)
> lines(min(x):max(x), dnorm(min(x):max(x),
    mean=mean(x), sd=sd(x)))
```

Answer of 10.6.

```
> plot(1:50, dpois(1:50, lambda=3), type='l')
> lines(1:50, dpois(1:50, lambda=10), type='l')
> lines(1:50, dpois(1:50, lambda=30), type='l')
```

Answer of 10.7.

```
> N=10000
> sample.n <- rnorm(N)
> sample.t <- rt(N, df=3)
> sample.f <- rf(N, df1=3, df2=10)
> sample.l <- rlnorm(N)
> sample.p <- rpois(N, lambda=3)
> sample.b <- rbinom(N, size=10, p=0.1)
> qqnorm(sample.n, main='normal');qqline(sample.n)
> qqnorm(sample.t, main='t');qqline(sample.t)
> qqnorm(sample.f, main='F');qqline(sample.f)
> qqnorm(sample.l, main='log
    normal');qqline(sample.l)
> qqnorm(sample.p, main='Poisson');qqline(sample.p)
> qqnorm(sample.b,
    main='binomial');qqline(sample.b)
```

A.10 Logistic Regression

Answer of 11.1.

```
> seg <-
    read.csv('http://coltekin.net/cagri/R/data/seg-large.csv')
> seg$utterance <- factor(seg$utterance)
> seg$phoneme <- factor(seg$phoneme)
```

The last line is most probably not necessary.

Answer of 11.2.

```
> or <-
    read.csv('http://coltekin.net/cagri/R/data/past-tense-or
> or$correct <- 1 - (or$n.or / or$n.past)
```

Answer of 11.3. The diagnostic plots you get with

```
> plot(lm(correct ~ age, data=or))
```

should indicate that the data points 68 is the most influential observation.

The command

```
> or.ols <- lm(correct ~ age, data=or, subset=-68)
```

fits the model without the outlier.

If you plot the diagnostics again, you should still observe that the variance is not constant. The Q-Q plot indicates some non-normality, and we can also observe some slight non-linearity in the 'Scale-location' graph.

Answer of 11.4.

```
> predict(or.ols,
          newdata=data.frame(age=c(1.5*12, 8*12)))
        1         2
0.8072069 1.0024616
```

Answer of 11.5.

```
> p <- seq(0,1, 0.01)
> plot(p, log(p/(1-p)), type='l')
```

Answer of 11.6.

```
> or$log.odds <- log(or$correct/(1 - or$correct))
> or.logit <- lm(log.odds ~ age, data=or,
    subset=-68)
> summary(or.logit)
> plot(or.logit)
```

The coefficients now reflect the linear equation predicting log odds. If we want to predict the probabilities, we need to calculate the log odds for the given age, take the exponent of it (to cancel the log), the expected probability or correct responses for the given age can be calculated with $p = \frac{odds}{1+odds}$ (it is a simple arithmetic exercise to get this equation from the definition of the odds) .

In this model the relation between the probability and the age is non-linear.

The transformation seem to improve the model diagnostics. In particular, the variance seems more constant now (although you should see clear differences between the residuals below 0 and above 0). We still see the effects of non-linearity and non-normality in the graphs.

Answer of 11.7. We do it piece-by-piece for demonstration:

```
> logit <- predict(or.logit,
                   newdata=data.frame(age=12*(1:10)))
> odds <- exp(logit)
> odds / (1 + odds)
```

The first line gets the predictions for each age, second line takes the exponent (undoes the log), and the last line converts the odds to probabilities.

Answer of 11.8. Command is almost the same.

```
> summary(glm(cbind(n.past-n.or, n.or) ~ age,
      family='binomial', data=or, subset-68))
```

In the summary, you should realize that an estimated 'dispersion' parameter is used instead of 1. This affects our model marginally since we do not have real overdispersion here. Since the estimated dispersion parameter is less than one, the

standard error for the coefficients are slightly tighter. In most cases (when there is overdispersion), `quasibinomial` will result in less certain coefficient estimates.

Answer of 11.9.

```
The command
> 1 - predict(or.glm, type='response',
        newdata=data.frame(age=12*(1:10)))
should give you the result as a vector.
```

Answer of 11.10.

```
> plot(correct ~ age, data = or)
> x <- min(or$age):max(or$age)
> y <- predict(or.glm, type='response',
        newdata=data.frame(age=x))
> lines(x, y)
```

Note that the curve you see is only the part of the logistic curve. Although it is not useful in this case, you are encouraged to try the plot using a larger age range, e.g., `-200:200`, to see the complete curve.

Answer of 11.11.

```
> seg.pmi <- glm(boundary~pmi, data=seg,
    family=binomial)
> plot(boundary ~ pmi, data = seg)
> x <- seq(min(seg$pmi), max(seg$pmi), by=0.2)
> y <- predict(seg.pmi, type='response',
    newdata=data.frame(pmi=x))
> lines(x, y)
```

Again, the difference between the residual deviance and the degrees of freedom is not large (we even have a bit of underdispersion). However, in general, overdispersion does not occur with binary response variables.

Answer of 11.12. Here are three ways to get the accuracy (all results the same value):

```
> success <- seg$boundary == (predict(seg.pmi,
    type='response') > 0.5)
> length(success[success == TRUE])/length(success)
[1] 0.772229
> length(success[success])/length(success)
> sum(success)/length(success)
```

The first one (line 2) is more descriptive, but understanding the tricks used in more compact notations are worth the effort.

Answer of 11.13.

```
> length(seg$boundary[seg$boundary == F]) /
  length(seg$boundary)
[1] 0.6394641
```

and the trick to compact it:

```
> sum(!seg$boundary)/length(seg$boundary)
```

result is again the same. Indeed, our model performs better than the baseline (we will soon test our confidence in this).

Answer of 11.14.

```
> seg.full <- glm(boundary~ ., data=seg,
    family=binomial)
> summary(seg.full)
```

First thing to note here is that we get a warning about estimated probability values exceeding 0 or 1. By itself, this is not necessarily alarming. In this case, it only means that the some of the estimated probabilities are equal to 0 or 1 with the available numeric precision. Here we will not worry about this, but if you get such warnings from R, you should understand what is going on. We will shortly see an example where the warning is important.

The GLM summary, otherwise, is too crowded to try to interpret. We will first simplify the model.

Answer of 11.15.

```
> anova(seg.pmi, seg.full, test='Chisq')
```

Answer of 11.16.

```
> step(model.full)
```

should present you the stepwise elimination beginning from the full model. The result indicates that `utterance` and `h` were dropped. The first one is a clear choice, as the utterance number has nothing to do with the word boundaries. The second one is due to high multicollinearity (mainly high correlation between `h` and `sv`, you are encouraged to check this with `cor()` and/or `vif()` from the `car` package).

Answer of 11.17.

```
> success.model <- seg$boundary ==
                (predict(seg.model,
                    type='response') > 0.5)
> sum(success.model)/length(success.model)
[1] 0.9123021
> success.pmi <- seg$boundary ==
                (predict(seg.pmi, type='response')
                    > 0.5)
> fisher.test(success.model, success.pmi,
              alternative='greater')
```

Fisher's test indicates that the difference between these two models are very unlikely to be due to chance.

Note that you should not interpret this as an indication that the larger model is better outside this data set. It just means that correct/incorrect decisions made by the two models on this data set are unlikely to be the same.

Answer of 11.18.

```
> seg.half <- glm(boundary ~ pmi + h + rh,
                  family = binomial,
                  data=seg, subset=(utterance < 50))
> success.training <- seg$boundary[seg$utterance <
    50] ==
                (predict(seg.half, type='response')
                    > 0.5)
> sum(success.training)/length(success.training)
[1] 0.8495146
> success.test <- seg$boundary[seg$utterance >=
    50] ==
    (predict(seg.half,
             newdata=seg[seg$utterance >= 50,],
            type='response') > 0.5)
> sum(success.test)/length(success.test)
[1] 0.799511
```

As expected, the accuracy is better on training data (the data we used for fitting the model) compared to the novel data. This is a sign of overfitting. The model do not only model the systematic relationship, but also some of the noise in the data.

Variations of this procedure is common in statistical machine learning literature to protect against overfitting.

Answer of 11.19. The R commands do not change much, so they are not included here.

The accuracy values you find should be surprisingly close (the accuracy on the test data is even higher). This means that the model did not overfit at all (thanks to small number of predictors).

A.11 Multilevel / mixed-effect models

Answer of 12.1.

```
> load(url('http://coltekin.net/cagri/R/data/par.
    rda'), verbose=T)
```

Answer of 12.2.

```
> par <- merge(merge(par.subj, par.item), par.data)
```

Answer of 12.3. There is nothing special about the syntax, we have done this many times:

```
> summary(nb.lm <- lm(rate ~ language,
    data=newborn))
```

We also know the relation between the t-test and the ordinary linear regression, e.g., from Exercise 9.2. The intercept in linear regression will be the mean of the one of the groups, the slope will indicate the difference between them, and the p-values will be the same (to make sure that the analyses are equivalent, you should disable the 'Welch correction' in t.test() by `var.equal=T`).

Noting the residual variance (or standard deviation) reported in the `lm()` output is important for comparing this exercise to the ones that follow.

Answer of 12.4.

```
> plot(rate ~ jitter(c(0,1)[language],
    amount=0.05),
       data=newborn)
> abline(nb.lm)
```

Answer of 12.5. You will find that the sum of the participant and residual variances in `nb.lmer1` is exactly the same as the residual variance of the model `nb.lm` (remember variance is the squared standard deviation).

Answer of 12.6.

```
> library(lattice)
> dotplot(ranef(nb.lmer1, condVar=T))
```

Answer of 12.7.

```
> fixef(nb.lmer1)[1] +
    ranef(nb.lmer1)$participant[1,]
    22.7941
```

Returns the estimated intercept for the participant number 1, which should correspond to the estimated sucking rate of this baby while listening to the foreign language input. We add fixed slope to this value to obtain the rate for the native language input.

```
> fixef(nb.lmer1)[1] + fixef(nb.lmer1)[2] +
    ranef(nb.lmer1)$participant[1,]
   27.31777
```

Repeating it for participant 3 in a somewhat compact notation, we get:

```
> c(fixef(nb.lmer1)[1], sum(fixef(nb.lmer1))) +
    ranef(nb.lmer1)$participant[3,]
   49.07146    53.59512
```

To compare it with the observed values, one way to get the information we need is:

```
> newborn[newborn$participant %in% c(1,3),]
  participant language  rate
1           1   native 29.01
2           1  foreign 20.06
5           3   native 50.92
6           3  foreign 53.73
```

The values are definitely not the same. The differences in `native` (the slope) are expected since we estimate a single slope for all babies. The difference between the intercept estimates and the observed values are more interesting here. The important thing to notice here is that participant 1 has a rather small observed intercept (20.06), while participant 3 has a rather large one (53.73). The estimated values, 22.79 and 49.07, are not too far from the observed values. However, they are pulled towards the common mean.

Answer of 12.8.

```
> plot(rate ~ jitter(c(0,1)[language],
    amount=0.05),
        xaxt="n", xlab='language', data=newborn)
> axis(1, at=c(0,1), labels=c('foreign', 'native'))
> abline(nb.lm)
> abline(a=fixef(nb.lmer1)[1] +
          ranef(nb.lmer1)$participant[1,],
         b=fixef(nb.lmer1)[2], col='red')
> abline(a=fixef(nb.lmer1)[1] +
          ranef(nb.lmer1)$participant[3,],
         b=fixef(nb.lmer1)[2], col='blue')
> x <- c(0,1,0,1)
> y <- with(newborn,
    c(rate[participant == '1' & language ==
        'foreign'],
      rate[participant == '1' & language ==
        'native'],
      rate[participant == '3' & language ==
        'foreign'],
      rate[participant == '3' & language ==
        'native']))
> points(x, y, pch=21,
        bg=c('red','red', 'blue','blue'))
```

(the last part, plotting the colored points, can be done with a more compact syntax)

Answer of 12.9.

```
> nb.lm2 <- lm(rate ~ language + participant,
    data=newborn)
> summary(nb.lm2)
```

The slope should be the same as `nb.lm`, and almost the same as `nb.lmer1`. The intercept now is the 'foreign' rate for the first participant. The new (slope) coefficients for `participant` are the differences of the respective participant and the intercept term.

Adding the subject variable reduces the variation. Actually, it is now the same as the residual variance in `nb.lmer1`.

The slope estimate of `language` is not differnt for different subjects as in other models. The intercept is now associated with the first subject, and to find the 'intercept' of the third subject we need to add these coefficients.

```
> coef(nb.lm2)[1];coef(nb.lm2)[1] +
    coef(nb.lm2)['participant3']
```

Answer of 12.10.

```
> abline(a=coef(nb.lm2),
         b=coef(nb.lm2)['languagenative'],
         col='red', lty='dotted')
> abline(a=coef(nb.lm2)[1]+coef(nb.lm2)[3],
         b=coef(nb.lm2)['languagenative'],
         col='blue', lty='dotted')
```

You should notice that the lines for mixed-effect model is closer to the `lm()` estimate ignoring the subject variation compared to the model that includes participants as fixed effects.

Answer of 12.11.

```
> bl.lmer2 <- lmer(mlu ~ language +
    (language|subj), data=bilingual, REML=F)
> bl.lmer3 <- lmer(mlu ~ language + (1|subj),
    data=bilingual, REML=F)
```

Summaries of the models fits, or if you run AIC(bl.lmer3, bl.lmer2), should indicate that bl.lmer3 has a smaller AIC value (remember that we prefer models with smaller AIC values).

Answer of 12.12.

```
> anova(bl.lmer3, bl.lmer2)
```

Remember that convention is to specify the smaller model first, but it does not change the results. The test above returns a p-value of 0.6604, indicating that there isn't enough evidence in the data in support of more complex model.

Answer of 12.13. > dotplot(ranef(bl.lmer2, condVar=T))

Answer of 12.14.

```
> nb.lmer2 <- lmer(rate ~ 1 + (1|participant),
    data=newborn)
> anova(nb.lmer2, nb.lmer1)
```

You should see that the p-value reported is really small (6.293×10^6), indicating that the predictor has a statistically significant effect.

Answer of 12.15.

```
> confint(profile(nb.lmer1))
> confint(profile(nb.lmer1), level=0.99)
```

In both cases you should observe that the confidence interval for languagenative does not include 0. Hence, you can reject the null hypothesis (that the babies react to both stimuli the same way) at level $p < .01$ (and of course at $p < .05$). It is not surprising that you get this value given the p-value found in Exercise 12.14.

The coefficient of the random effect, participant, is reported as .sig01. You should observe that the intervals calculated does not include 0 for the random effect either.

Answer of 12.16. First we fit a series of models:

```
> bl.la <- lmer(mlu ~ language*age +
    (language+age|subj), data=bilingual, REML=F)
> bl.l <- lmer(mlu ~ language*age +
    (language|subj), data=bilingual, REML=F)
> bl.a <- lmer(mlu ~ language*age + (age|subj),
    data=bilingual, REML=F)
> bl.i <- lmer(mlu ~ language*age + (1|subj),
    data=bilingual, REML=F)
```

First line fits random intercepts and slopes for both language and age, the second line fits only random slopes for language, the third line fits only random slopes for age, and the last line fits a random-intercepts-only model. Random intercepts are implicitly specified in first three models.

Checking AIC values

```
> AIC(bl.i, bl.a, bl.l, bl.la)
      df       AIC
bl.i   8 370.7229
bl.a  13 375.8502
bl.l  10 373.0892
bl.la 17 375.9423
```

indicate that there is no point in using the larger models. The simplest random-intercepts-only model is the best.

For the hypothesis testing part,

```
> confint(profile(bl.i))
```

Indicate that confidence intervals of intercept, agesecondgrade and interaction term languageschool:agesecondgrade do not include 0, hence statistically significant at the given level.

We will skip the full interpretation here, but you should try to interpret the result.

Answer of 12.17.

```
> bl.i2 <- update(bl.i, . ~ . + gender)
> confint(profile(bl.i2))
```

The confidence interval for genderM contains 0. Hence we decide that the gender is not significant.

Answer of 12.18.

```
> par.m1 <- lmer(rate ~ context +
    (context|subject) + (context|item), data=par)
> summary(par.m1)
...
Random effects:
 Groups   Name        Variance Std.Dev. Corr
 subject  (Intercept) 1.34353  1.1591
          contextpp   0.01033  0.1017   0.76
          contextip   0.01895  0.1377   0.63  0.98
 item     (Intercept) 6.91181  2.6290
          contextpp   0.01416  0.1190   -0.41
          contextip   0.05385  0.2321   -0.19  0.97
 Residual             0.97588  0.9879
Number of obs: 900, groups: subject, 30; item, 10
Fixed effects:
            Estimate Std. Error t value
(Intercept)  4.26505    0.85978   4.961
contextpp    0.79375    0.09092   8.730
contextip   -1.00695    0.11190  -8.998
...
```

In the above listing only part of the output is included. Note that we have random intercepts and two random slopes because of both random effects. The correlation between the random effects are also modeled.

In the fixed effects listing, (Intercept) indicates the average speech rate for an average speaker (subject) and average phrase (item) in our base level context, in this case a parenthetical (par). The slopes of pp and ip indicate the differences from the base level. Again, the slope estimates here are the average estimates for speakers and phrases.

Answer of 12.19.

```
par.m1 <- lmer(rate ~ context + (context|subject)
    + (context|item), data=par, REML=F)
par.m2 <- lmer(rate ~ context + (context|subject)
    + (1|item), data=par, REML=F)
par.m3 <- lmer(rate ~ context + (1|subject) +
    (context|item), data=par, REML=F)
par.m4 <- lmer(rate ~ context + (1|subject) +
    (1|item), data=par, REML=F)
par.m5 <- lmer(rate ~ context + (1|subject) ,
    data=par, REML=F)
par.m6 <- lmer(rate ~ context + (1|item),
    data=par, REML=F)
AIC(par.m1, par.m2, par.m3, par.m4, par.m5, par.m6)
```

The best model suggested by the lowest AIC value is par.m4.

Answer of 12.20. Fitting the model is simple

```
> par.m4a <- lmer(rate ~ context + length +
    (1|subject) + (1|item), data=par, REML=F)
```

or

```
> par.m4a <- update(par.m4, . ~ . + length)
```

Now we have new fixed effect for length which indicates a higher speech rate for longer phrases. However, the intercept estimate is also changed drastically. Since the intercept now correspond to a parenthetical with 0 length. Furthermore, intercept estimate becomes less certain.

Answer of 12.21.

```
> par.m4a <- update(par.m4, . ~ . + scale(length))
```

Intercept now corresponds to a parenthetical phrase with average length. You should observe that the standard error for the intercept (in comparison to the model `par.m4`) is smaller now, indicating that our estimate of intercept has improved.

Answer of 12.22.

```
> par.m4b <- lmer(rate ~ context + scale(age) +
    sex + scale(length) + (1|subject) + (1|item),
    data=par,   REML=F)
```

The intercept now corresponds to the speech rate of a average-length parenthetical phrase for an average-aged female subject. The intercept estimate should be more certain than the earlier models. And now the subject variation (the standard deviation of random intercept due to subject) should also be smaller than before.

Answer of 12.23.

```
> dotplot(ranef(par.m4, which="subject",
    condVar=T))
> dotplot(ranef(par.m4b, which="subject",
    condVar=T))
```

If you pay attention to the x-axis, you will see that the overall variation is reduced considerably by the predictors `age` and `gender`.

Answer of 12.24.

```
> profile.m4b <- profile(par.m4b)
> confint(profile.m4b, level=0.99)
```

You should observe that none of the confidence intervals include 0. Hence, all effects are significant at α-level 0.01.

B Model formulas

Various commands in R accept a notation called *model formula*, or simply *formula*. The simplest form of the formula is,

```
y ~ x
```

where `x` and `y` are two variables. You can read this as '`y` is explained by `x`'. The *dependent* or *response* variable goes to the left of the tilde '`~`' and the *explanatory* or *independent* variables goes to the right. This formula roughly corresponds to the linear equation,

$$y = a + bx$$

The interpretation is slightly different if the variables are categorical. Note that the intercept, a, is implicit in the model formula. If you like, you can be explicit by using the notation `+ 1`. Or if you want to exclude it, e.g., force a regression line passing through the origin, you can exclude it by `- 1`. In case you have multiple explanatory variables, it is easy to include them using the same notation. For example if you had two explanatory variables `x1` and `x2`, you can specify it like this:

```
y ~ x1 + x2
```

The linear equation that correspond to this notation would be $y = a + b_1x_1 + b_2x_2$.

As you may have figured out already, the arithmetic operators such as `+` and `-` have different meanings in a formula. So, if the variable you are interested is a combination of R variables, then you need a special notation. For example, you might be interested in fitting a linear model where `y` is explained by the sum of `x1` and `x2`. That is, the equation you want to describe is $y = a + b \times (x_1 + x_2)$. In such cases you need to use a special function, `I()`, to protect the arithmetic operation from being interpreted as part of the formula. In the case of our example, the correct formula notation is

```
y ~ I(x1 + x2)
```

If your explanatory variables are categorical, as in ANOVA, you may fit a model where interaction of the variables is important. Interaction of variables in a formula is expressed with a term where variable names are concatenated with column(s) between the variables. For example, the formula

```
y ~ x1 + x2 + x1:x2
```

expresses a model where interaction of `x1` and `x2` are also included in the model fitting. For two variables, we have only one possible interaction. If you have many variables, and want to include all interaction terms, it may be a hassle to type all the interaction terms separately. For example, all interactions of three variables `x1`, `x2` and `x3` consist of the two-way interactions `x1:x2`, `x1:x3`, `x2:x3` and the three way interaction `x1:x2:x3`. To include all interactions, you can use '`*`' instead of '`+`'. For example, to include three variables and all interactions in a model formula, we simply type `y ~ x1 * x2 * x3`.

The formula notation is quite flexible and can express many other forms of 'models'. The above explanation should be enough to get you started. R documentation you can find on CRAN is the main reference, and you can find further information in the many books and documents on R.