

## Chapter 3

### Descriptive statistics

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

#### 1. Univariate statistics

##### 1.1. Frequency data

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

```
> UHM<-read.table(file.choose(),header=T,sep="\t",
  comment.char="",quote="")
> attach(UHM)
> str(UHM)
# inspect the structure of the data frame
'data.frame': 1000 obs. of 5 variables:
 $ CASE: int 1 2 3 4 5 6 7 8 9 10 ...
 $ SEX: Factor w/ 2 levels "female", "male": 2 1 1 1 2 ...
 $ FILLER: Factor w/ 3 levels "silence", "uh", "...": 3 1 1 3 ...
 $ GENRE: Factor w/ 2 levels "dialog", "monolog": 2 2 1 1 ...
 $ LENGTH: int 1014 1188 889 265 465 1278 671 1079 643 ...
```

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

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

If you also want to know the percentages of each disfluency, then you can either do this rather manually

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

... or you use the function `prop.table`, whose argument is a table generated with `table` and which returns the percentages of the frequencies in that table. (More arguments to `prop.table` will be discussed below.)

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

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

```
> 1:5.#.the.values.from.1.to.5¶
[1] 1 2 3 4 5
> cumsum(1:5).#.cumulative.sums.of.the.values.from.1.to.5¶
[1] 1 3 6 10 15
```

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

```
> cumsum(table(FILLER))¶
silence.....uh.....uhm
....332.....726....1000
> cumsum(prop.table(table(FILLER)))¶
silence.....uh.....uhm
..0.332...0.726...1.000
```

Usually, it is instructive to represent the observed distribution graphically and the sections below introduce a few graphical formats. For reasons of space, I can only discuss some ways to tweak graphs, but you can turn to the documentation of these functions and Murrell (2005) for more info.

## 1.1.1. Scatterplots and line plots

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

```
> a<-c(1,.3,.5,.2,.4);b<-1:5
> plot(a).#left panel of Figure 18
```

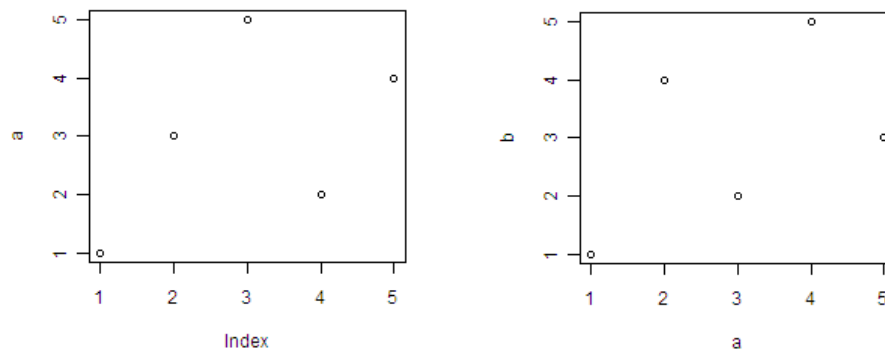


Figure 18. Simple scatterplots

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

```
> plot(a,b).#right panel of Figure 18
```

With the argument `type=...`, you can specify the kind of graph you want. The default, which was used because you did not specify anything else, is `type="p"` (for *points*). If you use `type="b"` (for *both*), you get points and lines connecting the points; if you use `type="l"` (for *lines*), you get a line plot; cf. Figure 19. (With `type="n"`, nothing gets plotted into the main plotting area, but the coordinate system is set up.)

```
> plot(b, a, type="b") # left panel of Figure 19
> plot(b, a, type="l") # right panel of Figure 19
```

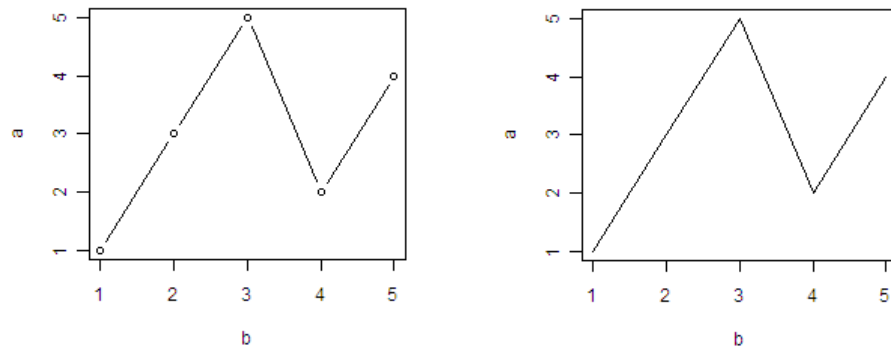


Figure 19. Simple line plots

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

```
> plot(b, a, xlab="A vector b", ylab="A vector a",
       xlim=c(0, 8), ylim=c(0, 8), type="b") # Figure 20
> grid()
```

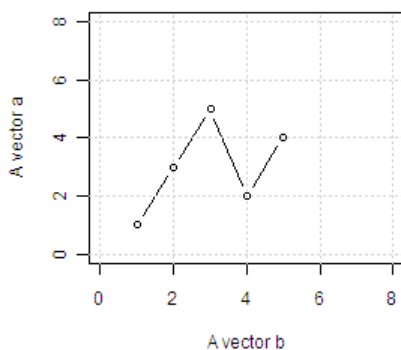


Figure 20. A scatterplot exemplifying a few simple plot settings

An important rule of thumb is that the ranges of the axes must be chosen in such a way that the distribution of the data is represented most meaningfully. It is often useful to include the point (0, 0) within the ranges of the

axes and to make sure that graphs that are to be compared have the same axis ranges. For example, if you want to compare the ranges of values of two vectors  $x$  and  $y$  in two graphs, then you usually may not want to let R decide on the ranges of axes. Consider the upper panel of Figure 21.

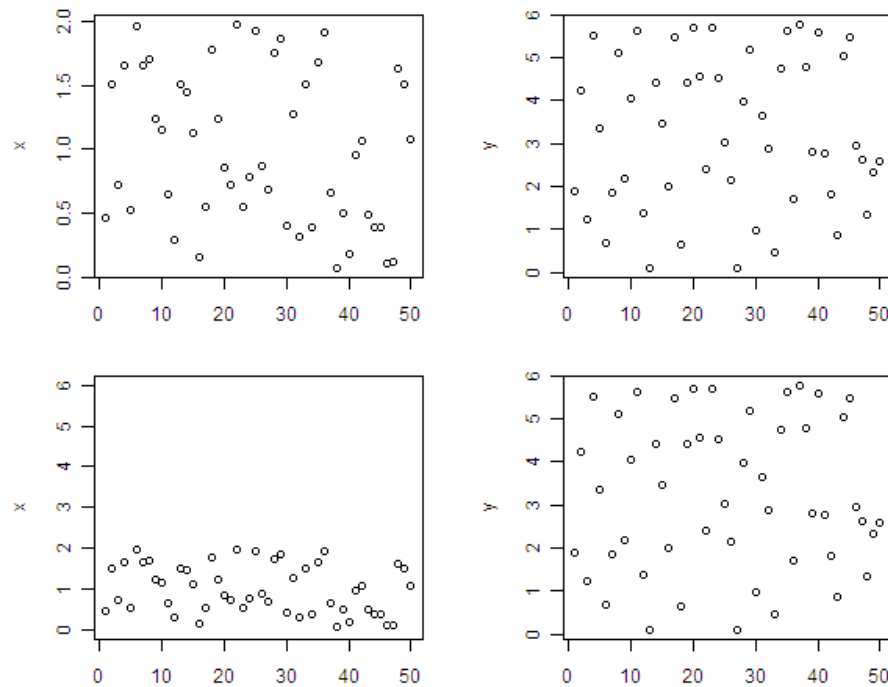


Figure 21. Scatterplots and the importance of properly-defined ranges of axes

The clouds of points look very similar and you only notice the distributional difference between  $x$  and  $y$  when you specifically look at the range of values on the  $y$ -axis. The values in the upper left panel range from 0 to 2 but those in the upper right panel range from 0 to 6. This difference between the two vectors is immediately obvious, however, when you use `ylim=...` to manually set the ranges of the  $y$ -axes to the same range of values, as I did for the lower panel of Figure 21.

Note: whenever you use `plot`, by default a new graph is created and the old graph is lost. If you want to plot two lines into a graph, you first generate the first with `plot` and then add the second one with `points` (or `lines`; sometimes you can also use the argument `add=T`). That also means that you must define the ranges of the axes in the first plot in such a way that the

values of the second graph can also be plotted into it. An example will clarify that point. If you want to plot the points of the vectors *m* and *n*, and then want to add into the same plot the points of the vectors *x* and *y*, then this does *not* work, as you can see in the left panel of Figure 22.

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

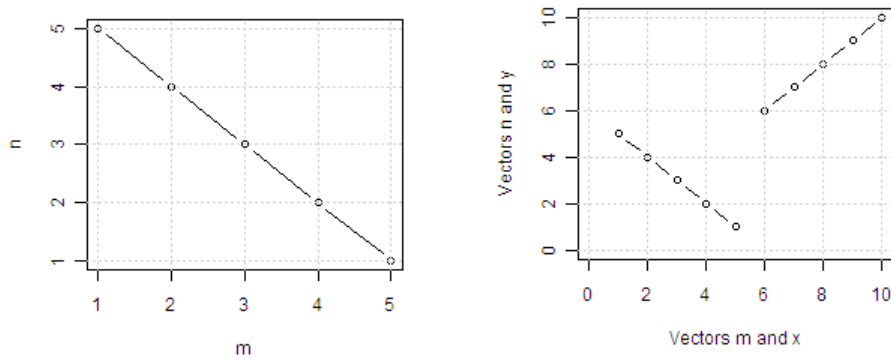


Figure 22. Scatterplots and the importance of properly-defined ranges of axes

The left panel of Figure 22 shows the points defined by *m* and *n*, but not those of *x* and *y* because the ranges of the axes that R used to plot *m* and *n* are too small for *x* and *y*, which is why you must define those manually while creating the first coordinate system. One way to do this is to use the function `max`, which returns the maximum value of a vector (and `min` returns the minimum). The right panel of Figure 22 shows that this does the trick. (In this line, the minimum is set to 0 manually – of course, you could also use `min(m, .x)` and `min(n, .y)` for that, but I wanted to include (0, 0) in the graph.)

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

#### Recommendation(s) for further study

the functions `pmin` and `pmax` to determine the minima and maxima at each position of different vectors (try `pmin(c(1, .5, .3), .c(2, .4, .6))`)

*1.1.2. Pie charts*

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

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



Figure 23. A pie chart with the frequencies of disfluencies

*1.1.3. Bar plots*

To create a bar plot, you can use the function `barplot`. Again, its most important argument is a table generated with `table` and again you can create either a standard version or more customized ones. If you want to define your own category names, you unfortunately must use `names.arg=...`, not `labels=...` (cf. Figure 24 below).

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

```
> barplot(table(FILLER)) # left panel of Figure 24
> barplot(table(FILLER), col=c("grey20", "grey40", "
```

```
"grey60"), .names.arg=c("Silence", ".Uh", ".Uhm")) .#  
right.panel.of.Figure.24¶
```

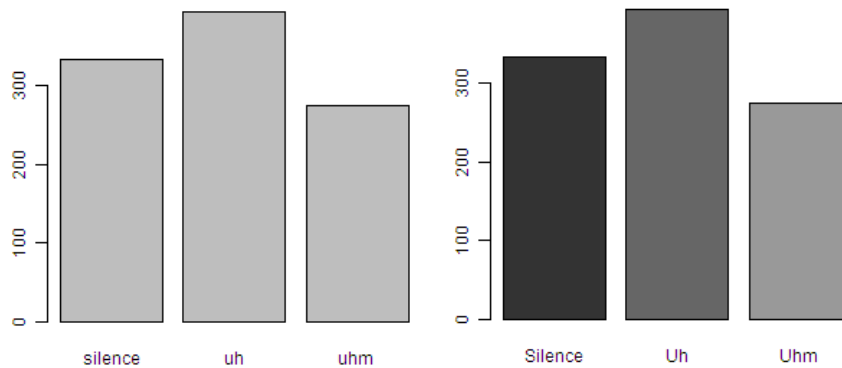


Figure 24. Bar plots with the frequencies of disfluencies

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

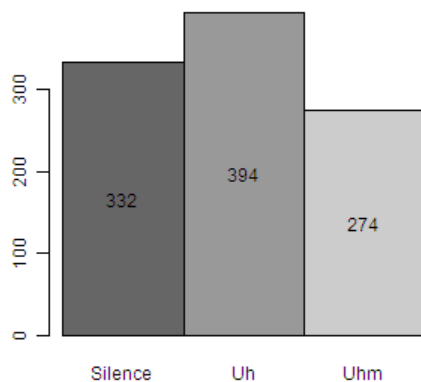


Figure 25. Bar plot with the frequencies of disfluencies

The functions `plot` and `text` allow for another interesting and easy-to-understand graph: first, you generate a plot that contains nothing but the axes and their labels (with `type="n"`, cf. above), and then with `text` you plot not points, but words or numbers. Try this:

```
> plot(c(394, 274, 332), .type="n", .xlab="Disfluencies", .ylab=  
  "Observed frequencies", .xlim=c(0, 4), .ylim=c(0, 500))¶
```



```
> text(1:3, c(394, 274, 332), labels=c("uh", "uhm", "silence"))
```

This probably already illustrates the potential of this kind of graph, of which you will see further examples below.

**Recommendation(s) for further study**

the parameter settings `cex`, `srt`, `col`, and `font` to tweak the size, rotation, color, and font type of text you plot: `?par`. Also, the function `segments` can be very useful to plot lines

#### 1.1.4. Pareto-charts

A related way to represent the frequencies of the disfluencies is a pareto-chart. In pareto-charts, the frequencies of the observed categories are represented as in a bar plot, but they are first sorted in descending order of frequency and then overlaid by a line plot of cumulative percentages that indicates what percent of all data one category and all other categories to the left of that category account for. The function `pareto.chart` comes with the library `qcc` that you must (install and/or) load first; cf. Figure 26.

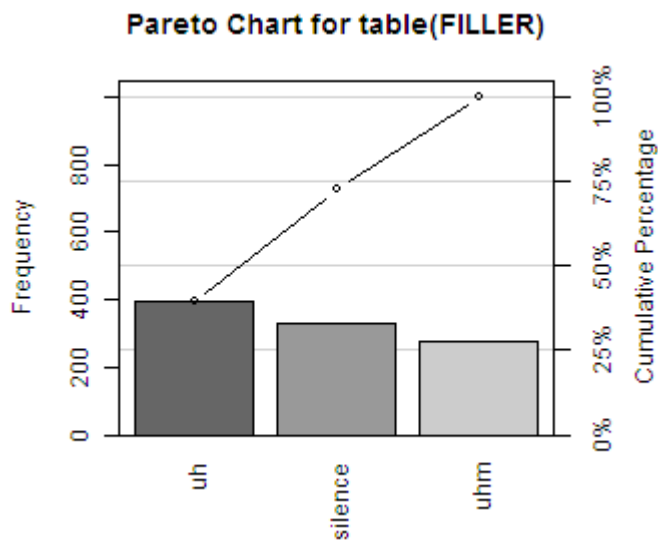


Figure 26. Pareto-chart with the frequencies of disfluencies

```

> library(qcc)
> pareto.chart(table(FILLER))
Pareto chart analysis for table(FILLER)
.....Frequency Cum. Freq. Percentage Cum. Percent.
..uh.....394.....394.....39.4.....39.4
..silence.....332.....726.....33.2.....72.6
..uhm.....274.....1000.....27.4.....100.0

```

### 1.1.5. Histograms

While pie charts and bar plots are probably the most frequent forms of representing the frequencies of nominal/categorical variables (many people advise against pie charts, though, because humans are notoriously bad at inferring proportions from angles in charts), histograms are most widespread for the frequencies of interval/ratio variables. In R, you can use the function `hist`, which just requires the relevant vector as its argument.

```

> hist(LENGTH) # standard graph

```

For some ways to make the graph nicer, cf. Figure 27, whose left panel contains a histogram of the variable `LENGTH` with axis labels and grey bars:

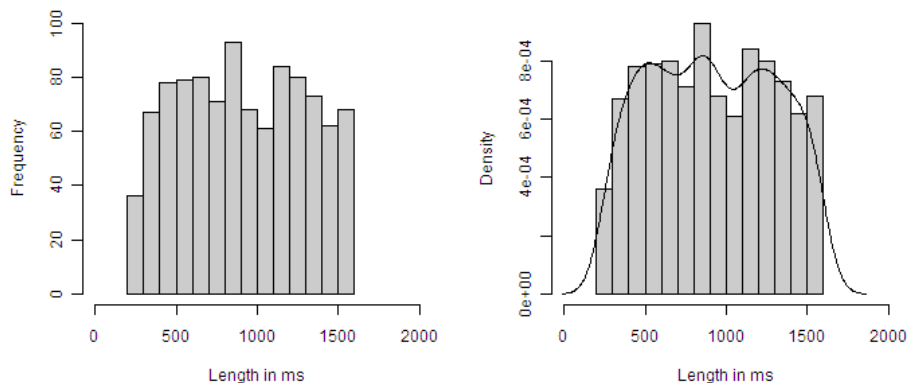


Figure 27. Histograms for the frequencies of lengths of disfluencies

```

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

```

The right panel of Figure 27 contains a histogram of the probability densities (generated by `freq=F`) with a corresponding curve:

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

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

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

#### **Recommendation(s) for further study**

- the functions `dotchart` and `stripchart` (with `method="jitter"`) to represent the distribution of individual data points
- the function `scatterplot` (from the `library(car)`) for more sophisticated scatterplots
- the functions `scatterplot3d` (from the `library(scatterplot3d)`) for 3-dimensional scatterplots

## 1.2. Measures of central tendency

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

### 1.2.1. The mode

The mode of a variable or distribution is the value that is most often ob-

served. As far as I know, there is no function for the mode in R, but you can find the mode easily. For example, the mode of `FILLER` is *uh*:

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

### 1.2.2. The median

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

```
> median(LENGTH)
[1] .897
```

### 1.2.3. The arithmetic mean

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

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

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

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

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

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

Second, you can just use the median:

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

Using the median is also a good idea if the data whose central tendency you want to report are not normally distributed.

#### **Warning/advice**

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

#### **Recommendation(s) for further study**

Good and Hardin (2006: 49f.) for additional measures of central tendency for asymmetric distributions

#### *1.2.4. The geometric mean*

The geometric mean is used to compute averages of factors or ratios (whereas the arithmetic mean is computed to get the average of sums). Let's assume you have six recordings of a child at the ages 2;1 (two years and one month), 2;2, 2;3, 2;4, 2;5, and 2;6. Let us also assume you had a vector

lexicon that contains the cumulative numbers of different words (types!) that the child produced at each age:

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

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

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

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

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

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

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

Instead, you must compute the geometric mean. The geometric mean of a vector  $x$  with  $n$  elements is computed according to formula (15):

$$(15) \quad \text{mean}_{\text{geom}} = (x_1 \cdot x_2 \cdot \dots \cdot x_{n-1} \cdot x_n)^{1/n}$$

That is:

```
> rate.increase<-prod(increases)^(1/length(increases));
  rate.increase
[1] 1.127009
```

If you use this value as the average rate of increase, you get the desired result:

```
> 132*rate.increase^5
[1] 240
```

True, the difference between 240 – the correct value – and 241.1681 – the incorrect value – may seem negligible, but 241.1681 is still wrong and the difference is not always that small, as an example from Wikipedia (s.v. *geometric mean*) illustrates: If you do an experiment and get an increase rate of 10.000 and then you do a second experiment and get an increase rate of 0.0001 (i.e., a decrease), then the average rate of increase is not approximately 5.000 – the arithmetic mean of the two rates – but 1 – their geometric mean.<sup>17</sup>

Finally, let me again point out how useful it can be to plot words or numbers instead of points, triangles, ... Try to generate Figure 28, in which the position of each word on the y-axis corresponds to the average length of the disfluency (e.g., 928.4 for women, 901.6 for men, etc.). (The horizontal line is the overall average length – you may not know yet how to plot that one.) Many tendencies are immediately obvious: men are below the average, women are above, silent disfluencies are of about average length, etc.

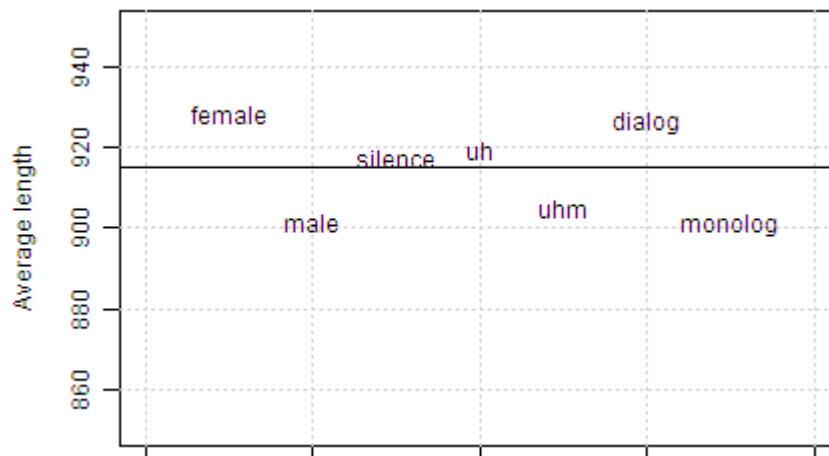


Figure 28. Mean lengths of disfluencies

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

### 1.3. Measures of dispersion

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

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

On the basis of the means alone, the towns seem to have a very similar climate, but even a quick glance at Figure 29 shows that that is not true – in spite of the similar means, I know where I would want to be in February.



Figure 29. Temperature curves of two towns



Obviously, the mean of Town 2 summarizes the central tendency of Town 2 much better than the mean of Town 1 does for Town 1: the values of Town 1 vary much more widely around their mean. Thus, always provide a measure of dispersion for your measure of central tendency: relative entropy for the mode, the interquartile range or quantiles for the median, the standard deviation or the variance or quantiles for interval/ratio-scaled data.

### 1.3.1. Relative entropy

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

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

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

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

The distribution of articles is rather heterogeneous. It is worth pointing out that the above formula does not produce the desired result of 0 when only no-determiner cases are observed:

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

This is because the logarithm of 0 is not defined. Usually, this is taken care of by simply setting the result of  $\log(0)$  to zero (or sometimes also by

incrementing all values by 1 before logging). While I do not talk at all about how to write your own functions here, this is a case where this can be useful: you could simply define your own logarithm function `logw0` and then use that function to get the desired result:

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

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

### 1.3.2. The range

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

```
> range(LENGTH)
[1] 251 1600
> diff(range(LENGTH)) # diff computes pairwise differences
[1] 1349
> max(LENGTH)-min(LENGTH)
[1] 1349
```

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

### 1.3.3. Quantiles and quartiles

Another very simple, but very useful measure of dispersion involves the quantiles of a distribution. You compute quantiles by sorting the values in ascending order and then counting which values delimit the lowest  $x\%$ ,  $y\%$ , etc. of the data; when these percentages are 25%, 50%, and 75%, then they are called quartiles. In R you can use the function `quantile`, and the following example makes all this clear:

```
> a<-1:100.#.a.vector.with.the.numbers.from.1.to.100¶
> quantile(a,.type=1)¶
..0%..25%..50%..75%..100%
..1...25...50...75...100
```

If you write the integers from 1 to 100 next to each other, then 25 is the value that cuts off the lower 25%, etc. The value for 50% corresponds to the median, and the values for 0% and 100% are the minimum and the maximum. Let me briefly mention two arguments of this function. First, the argument `probs` allows you to specify other percentages:

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

Second, the argument `type=...` allows you to choose other ways in which quantiles are computed. For discrete distributions, `type=1` is probably best, for continuous variables the default setting `type=7` is best. The bottom line of course is that the more the 25% quartile and the 75% quartile differ from each other, the more heterogeneous the data are, which is confirmed by looking at the data for the two towns: the so-called interquartile range – the difference between the 75% quartile and the 25% quartile – is much larger for Town 1 than for Town 2.

```
> quantile(town1)¶
..0%..25%..50%..75%..100%
-12.0...4.0..13.5..18.5..23.0
> IQR(town1).#.the.function.for.the.interquartile.range¶
[1] 14.5
> quantile(town2)¶
..0%..25%..50%..75%..100%
.6.00..7.75..9.00..11.25..16.00
> IQR(town2)¶
[1] 3.5
```

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

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

That is, the central 20% of all the lengths of disfluencies are between 788 and 1,039, 20% of the lengths are smaller than or equal to 519, 20% of the values are larger than 1,307, etc.

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

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

#### 1.3.4. The average deviation

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

```
> town1¶
· [1] -5 -12 -5 -12 -15 -18 -22 -23 -20 -16 -8 -1
> town1-mean(town1)¶
· [1] -15.25 -22.25 -5.25 -1.75 -4.75 -7.75 -11.75 -
12.75 -9.75 -5.75 -2.25 -9.25
> abs(town1-mean(town1))¶
· [1] 15.25 22.25 5.25 1.75 4.75 7.75 11.75 12.75
9.75 5.75 2.25 9.25
> mean(abs(town1-mean(town1)))¶
[1] 9.041667
> mean(abs(town2-mean(town2)))¶
[1] 2.472222
```

For the lengths of the disfluencies, we obtain:

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

Although this is a very intuitively accessible measure, it is hardly used anymore. Instead, you will most often find the dispersion measure discussed in the next section, the standard deviation.

*1.3.5. The standard deviation*

The standard deviation  $sd$  of a distribution  $x$  with  $n$  elements is defined in (17). This may look difficult at first, but the standard deviation is conceptually similar to the average deviation. For the average deviation, you compute the difference of each data point to the mean and take its absolute value, for the standard deviation you compute the difference of each data point to the mean and square it (and after the division by  $n-1$ , you take the square root).

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

Once we ‘translate’ this into R, it probably becomes much clearer:

```
> town1
[1] -5 -12 5 12 15 18 22 23 20 16 8 1
> town1 - mean(town1)
[1] -15.25 -22.25 -5.25 1.75 4.75 7.75 11.75 12.75 9.75 5.75 -2.25 -9.25
> (town1 - mean(town1))^2
[1] 232.5625 495.0625 27.5625 3.0625 22.5625 60.0625 138.0625 162.5625 95.0625 33.0625 5.0625 85.5625
> sum((town1 - mean(town1))^2) # the numerator
[1] 1360.25
> sum((town1 - mean(town1))^2) / (length(town1) - 1)
[1] 123.6591
> sqrt(sum((town1 - mean(town1))^2) / (length(town1) - 1))
[1] 11.12021
```

There is of course an easier way ...

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

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

**Recommendation(s) for further study**

the function `mad` to compute another very robust measure of dispersion, the median absolute deviation

*1.3.6. The variation coefficient*

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

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

When the values, and hence the mean, is increased by one order of magnitude, then so is the standard deviation. You can therefore not compare standard deviations from distributions with different means if you do not first normalize them. If you divide the standard deviation of a distribution by its mean, you get the variation coefficient:

```
> sd(town1)/mean(town1)
[1] 1.084899
> sd(town1*10)/mean(town1*10) # now you get the same value
[1] 1.084899
> sd(town2)/mean(town2)
[1] 0.3210999
```

You see that the variation coefficient is not affected by the multiplication with 10, and Town 1 still has a larger degree of dispersion.

**Recommendation(s) for further study**

- the functions `skewness` (from the `library(e1071)`), `skewness` (from the `library(fBasics)`), `basicStats` (from the `library(fBasics)`), `skewness` and `agostino.test` (from the `library(moments)`) to compute the so-called skewness
- the functions `kurtosis` (from the `library(e1071)`) and `kurtosis` (from the `library(fBasics)`) to compute the so-called kurtosis

### 1.3.7. Summary functions

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

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

**Recommendation(s) for further study**

the function `fivenum` for a different kind of summary

An immensely useful graph is the so-called boxplot. In its simplest form, the function `boxplot` just requires one vector as an argument, but we also add `notch=T` (recall, `T` is the short form for `TRUE`), which I will explain shortly, as well as a line that adds little plus signs for the arithmetic means.

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

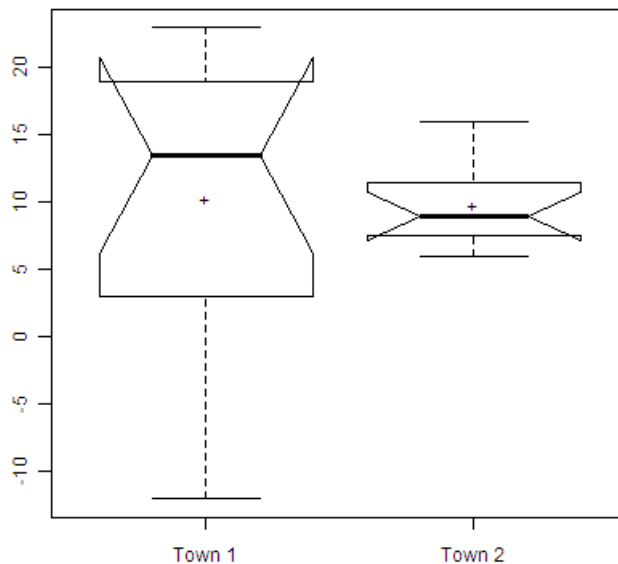


Figure 30. Boxplot of the temperatures of the two towns

This plot contains a lot of valuable information:

- the bold-typed horizontal lines represent the medians of the two vectors;
- the regular horizontal lines that make up the upper and lower boundary of the boxes represent the hinges (approximately the 75%- and the 25% quartiles);
- the whiskers – the dashed vertical lines extending from the box until the upper and lower limit – represent the largest and smallest values that are not more than 1.5 interquartile ranges away from the box;
- each outlier that would be outside of the range of the whiskers would be represented with an individual dot;
- the notches on the left and right sides of the boxes extend across the range  $\pm 1.58 \cdot \text{IQR} / \sqrt{n}$ : if the notches of two boxplots overlap, then these will most likely not be significantly different.

Figure 30 shows that the average temperatures of the two towns are very similar and not significantly different from each other. Also, you can see that the dispersion of Town 1 is much larger than that of Town 2. Sometimes, a good boxplot nearly obviates the need for further analysis, which is why they are extremely useful and will often be used in the chapters to follow.

#### **Recommendation(s) for further study**

the functions `hdr.boxplot` (from the `library(hdrcde)`), `vioplot` (from the `library(vioplot)`), and `bpplot` (from the `library(Hmisc)`) for interesting alternatives to, or extensions of, boxplots

#### *1.3.8. The standard error*

The standard error of an arithmetic mean is defined as the standard deviation of the means of equally large samples drawn randomly from a population with replacement. Imagine you took a sample from a population and computed the arithmetic mean of some variable. This mean will most likely not correspond exactly to the arithmetic mean of that variable in the population, and it will also most likely not correspond exactly to the arithmetic mean you would get from another equally large sample from the same population. If you took many random and equally large samples from the population with replacement and computed the arithmetic mean of each of them, then the standard deviation of all these means is the standard error. The standard error of an arithmetic mean is computed according to the formula



in (18), and from (18) you can already see that the larger the standard error of a mean, the smaller the likelihood that that mean is a good estimate of the population mean.

$$(18) \quad SE_{\text{mean}} = \sqrt{\frac{\text{var}}{n}} = \frac{sd}{\sqrt{n}}$$

Thus, the standard error of the mean length of disfluencies in our example is:

```
> mean(LENGTH)¶
[1] .915.043
> sqrt(var(LENGTH)/length(LENGTH)) .# or sd(LENGTH)/
  sqrt(length(LENGTH))¶
[1] .12.08127
```

This also means that, the larger sample size  $n$ , the smaller the standard error becomes.

You can also compute standard errors for statistics other than arithmetic means but the only other example we look at here with an example is the standard error of a relative frequency  $p$ , which is computed according to the formula in (19):

$$(19) \quad SE_{\text{percentage}} = \sqrt{\frac{p \cdot (1-p)}{n}}$$

Thus, the standard error of the percentage of all silent disfluencies out of all disfluencies (33.2%) is:

```
> prop.table(table(FILLER))¶
FILLER
silence . . . . . uh . . . . . uhm
..0.332 . . . 0.394 . . . 0.274
> sqrt(0.332*(1-0.332)/1000)¶
[1] .0.01489215
```

Standard errors will be much more important in Section 3.1.5 because they are used to compute so-called confidence intervals. In Chapter 5 you will also get see standard errors of differences of means, which are computed according to the formula in (20).

$$(20) \quad SE_{\text{difference between means}} = \sqrt{SE_{\text{mean\_group1}}^2 + SE_{\text{mean\_group2}}^2}$$

**Warning/advice**

Standard errors are only really useful if the data to which they are applied are distributed normally or when the sample size  $n \geq 30$ .

## 1.4. Centering and standardization (z-scores)

Very often it is useful or even necessary to compare values coming from different scales. An example (from Bortz 2005): if a student X scored 80% in a course and a student Y scored 60% in another course, can you then say that student X was better than student Y? On the one hand, sure you can: 80% is better than 60%. On the other hand, the test in which student Y participated could have been much more difficult than the one in which student X participated. It can therefore be useful to relativize/normalize the individual grades of the two students on the basis of the overall performance of students in their courses. (You encountered a similar situation above in Section 3.1.3.6 when you learned that it is not always appropriate to compare different standard deviations directly.) Let us assume the grades obtained in the two courses look as follows:

```
> grades.course.X<-rep((seq(0,100,20)),1:6);
grades.course.X
[1] 0 20 20 40 40 40 60 60 60 60 60 80 80 80 80
80 100 100 100 100 100 100
> grades.course.Y<-rep((seq(0,100,20)),6:1);
grades.course.Y
[1] 0 0 0 0 0 0 20 20 20 20 20 40 40 40
40 60 60 60 80 80 100
```

One way to normalize the grades is called *centering* and simply involves subtracting from each individual value within one course the average of that course.

```
> a<-1:5
> centered.scores<-a-mean(a); centered.scores
[1] -2 -1 0 1 2
```

You can see how these scores relate to the original values in a: since the mean of a is obviously 3, the first two centered scores are negative (they

are smaller than a's mean), the third is 0 (it does not deviate from the mean of a), and the last two centered scores are positive (they are larger than the mean of a).

Another more sophisticated way involves *standardizing*, i.e. transforming the values to be compared into so-called *z-scores*, which indicate how many standard deviations each value deviates from the mean of the vector. The *z-score* of a value from a vector is the difference of that value from the mean of the vector, divided by the vector's standard deviation. You can compute that manually as in this simple example:

```
> z.scores<-(a-mean(a))/sd(a); z.scores
[1] -1.2649111 -0.6324555 0.0000000 0.6324555 1.2649111
```

The relationship between the *z-scores* and a's original values is very similar to that between the centered scores and a's values: since the mean of a is obviously 3, the first two *z-scores* are negative (they are smaller than a's mean), the third *z-score* is 0 (it does not deviate from the mean of a), and the last two *z-scores* are positive (they are larger than the mean of a). Note that the *z-scores* have a mean of 0 and a standard deviation of 1:

```
> mean(z.scores)
[1] 0
> sd(z.scores)
[1] 1
```

Both normalizations can be performed with the function `scale`, which takes three arguments, the vector to be normalized, `center=...` (the default is  $\tau$ ) and `scale=...` (the default is  $\tau$ ). If you do not provide any arguments other than the vector to be standardized, then `scale`'s default setting returns a matrix that contains the *z-scores* and the attributes of which correspond to the mean and the standard deviation of the vector:

```
> scale(a)
.....[,1]
[1,] -1.2649111
[2,] -0.6324555
[3,] 0.0000000
[4,] 0.6324555
[5,] 1.2649111
attr(,"scaled:center")
[1] 3
attr(,"scaled:scale")
[1] 1.581139
```

If you set `scale` to `F`, (or `FALSE`), then you get centered scores:

```
> scale(a, .scale=F)
...[,1]
[1,] ...-2
[2,] ...-1
[3,] ...0
[4,] ...1
[5,] ...2
attr(,"scaled:center")
[1] 3
```

If we apply both versions to our example with the two courses, then you see that the 80% scored by student X is only 0.436 standard deviations and 13.33 percent points better than the mean of his course whereas the 60% scored by student Y is actually 0.873 standard deviations and 26.67 percent points above the mean of his course. Thus, X's score is higher than Y's score, but if we include the overall results in the two courses, then Y's performance is better. It is therefore often useful to standardize data in this way.

### 1.5. Confidence intervals

In most cases, you will not be able to investigate the whole population you are actually interested in, e.g., because that population is too large and investigating it would be too time-consuming and/or expensive. However, even though you also know that different samples will yield different statistics, you of course hope that your sample would yield a reliable estimate that tells you much about the population you are interested in:

- if you find in a sample of 1000 disfluencies that their average length is approximately 915 ms, then you hope that you can generalize from that to the population and future investigations;
- if you find in a sample of 1000 disfluencies that 33.2% of these are silences, then you hope that you can generalize from that to the population and future investigations.

So far, we have only discussed how you can compute percentages and means for samples – the question of how valid these are for populations is the topic of this section. In Section 3.1.5.1, I explain how you can compute confidence intervals for arithmetic means, and Section 3.1.5.2 explains how

to compute confidence intervals for percentages. The relevance of such confidence intervals must not be underestimated: without a confidence interval it is unclear how well you can generalize from a sample to a population; apart from the statistics we discuss here, one can also compute confidence intervals for many other statistics.

### 1.5.1. Confidence intervals of arithmetic means

If you compute a mean on the basis of a sample, you of course hope that it represents that of the population well. As you know, the average length of disfluencies in our example data is 915.043 ms (standard deviation: 382.04). But as we said above, other samples' means will be different so you would ideally want to quantify your confidence in this estimate. The so-called confidence interval, which you should provide most of the time together with your mean, is the interval of values around the sample mean around which we accept there is no significant difference with the sample mean. From the expression "significant difference", it already follows that a confidence interval is typically defined as 1-significance level, i.e., usually as  $1-0.05 = 0.95$ , and the logic is that "if we derive a large number of 95% confidence intervals, we can expect the true value of the parameter [in the population] to be included in the computed intervals 95% of the time" (Good and Hardin 2006:111).

In a first step, you again compute the standard error of the arithmetic mean according to the formula in (18).

```
> se<-sqrt(var(LENGTH)/length(LENGTH)); se
[1] 12.08127
```

This standard error is used in (21) to compute the confidence interval. The parameter  $t$  in formula (21) refers to the distribution mentioned in Section 1.3.4.2, and its computation requires the number of degrees of freedom. In this case, the number of degrees of freedom  $df$  is the length of the vector-1, i.e. 999. Since you want to compute a  $t$ -value on the basis of a  $p$ -value, you need the function `qt`, and since you want a two-tailed interval – 95% of the values around the observed mean, i.e. values larger and smaller than the mean – you must compute the  $t$ -value for 2.5% (because 2.5% on both sides result in the desired 5%):

$$(21) \quad CI = \bar{x} \pm t \cdot SE$$

```
> t<-qt(0.025, df=999, lower.tail=F); t
[1] 1.962341
```

Now you can compute the confidence interval:

```
> mean(LENGTH)-(se*t)
[1] 891.3354
> mean(LENGTH)+(se*t)
[1] 938.7506
```

This 95% confidence interval means that the true population means that could have generated the sample mean (of 915.043) with a probability of 95% are between 891.34 and 938.75; the limits for the 99% confidence interval are 883.86 and 946.22 respectively.

To do this more simply, you can use the function `t.test` with the relevant vector and use `conf.level=...` to define the relevant percentage. R then computes a significance test the details of which are not relevant yet, which is why we only look at the confidence interval (with `$conf.int`):

```
> t.test(LENGTH, conf.level=0.95)$conf.int
[1] 891.3354 938.7506
attr(,"conf.level")
[1] 0.95
```

When you compare two means and their confidence intervals do not overlap, then the sample means are significantly different and, therefore, you would assume that there is a real difference between the population means, too. Note however that means can be significantly different from each other even when their confidence intervals overlap (cf. Crawley 2005: 169f.).

### 1.5.2. Confidence intervals of percentages

The above logic with regard to means also applies to percentages. Given a particular percentage from a sample, you want to know what the corresponding percentage in the population is. As you already know, the percentage of silent disfluencies in our sample is 33.2%. Again, you would like to quantify your confidence in that sample percentage. As above, you compute the standard error for percentages according to the formula in (19), and then this standard error is inserted into the formula in (22).

```
> se<-sqrt(0.332*(1-0.332)/1000);.se
[1] 0.01489215
```

$$(22) \quad CI = a \pm z \cdot SE$$

The parameter  $z$  in this formula corresponds to the  $z$ -score mentioned above in Section 1.3.4.2, which defines 5% of the area under a standard normal distribution – 2.5% from the upper part and 2.5% from the lower part. Thus, for a 95% confidence interval you enter:

```
> z<-qnorm(0.025,.lower.tail=F);.z
[1] 1.959964
```

And for a 99% confidence interval:

```
> z<-qnorm(0.005,.lower.tail=F);.z
[1] 2.575829
```

For a 95% confidence interval for the percentage of silences, you enter:

```
> z<-qnorm(0.025,.lower.tail=F)
> 0.332+z*.se;.332-z*.se
[1] 0.3611881
[1] 0.3028119
```

The simpler way requires the function `prop.test`, which tests whether a percentage obtained in a sample is significantly different from an expected percentage. Again, the functionality of the significance test is not relevant yet (however, cf. below Section 4.1.1.2), but this function also returns the confidence interval for the observed percentage. R needs the observed frequency (332), the sample size (1000), and the probability for the confidence interval. R uses a formula different from ours but returns nearly the same result.

```
> prop.test(332,.1000,.conf.level=0.95)$conf.int
[1] 0.3030166 0.3622912
attr(,"conf.level")
[1] 0.95
```

#### **Recommendation(s) for further study**

Dalgaard (2002: Ch. 7.1 and 4.1), Crawley (2005: 167ff.)

**Warning/advice**

Since confidence intervals are based on standard errors, the warning from above applies here, too: if data are not normally distributed or the samples too small, then you must often use other methods to estimate confidence intervals (e.g., bootstrapping).

**2. Bivariate statistics**

We have so far dealt with statistics and graphs that describe one variable or vector/factor. In this section, we now turn to methods to characterize two variables and their relation. We will again begin with frequencies, then we will discuss means, and finally talk about correlations. You will see that we can use many functions from the previous section.

**2.1. Frequencies and crosstabulation**

We begin with the case of two nominal/categorical variables. Usually, one wants to know which combinations of variable levels occur how often. The simplest way to do this uses cross-tabulation. Let us return to the disfluency data:

```
> UHM<-read.table(choose.files(default="03-1_uh(m).txt"),
+   header=T, .sep="\t", .comment.char="", .quote="")
> attach(UHM)
```

Let's assume you wanted to see whether men and women differ with regard to the kind of disfluencies they produce. First two questions: are there dependent and independent variables in this design and, if so, which?



**THINK  
BREAK**

In this case, SEX is the independent variable and FILLER is the dependent variable. Computing the frequencies of variable level combinations in R is easy because you can use the same function that you use to compute frequencies of an individual variable's levels: `table`. You just give `table` a



second vector or factor as an argument and R lists the levels of the first vector in the rows and the levels of the second in the columns:

```
> freqs<-table(FILLER,.SEX);.freqs
.....SEX
FILLER....female..male
..silence...171..161
..uh.....161..233
..uhm.....170..104
```

In fact you can provide even more vectors to `table`, just try it out, and in Section 5 we will return to this. Again, you can create tables of percentages with `prop.table`, but with two-dimensional tables there are different ways to compute percentages and you can specify one with `margin=...`. The default is `margin=NULL`, which computes the percentages on the basis of the total number of elements in the table. In other words, the sum of all percentages in the table is 1. A different possibility is to compute row percentages: set `margin=1` and in the table you then get percentages that add up to 1 in every row. Finally, you can choose column percentages by setting `margin=2`: the percentages in each column add up to 1. This is probably the best way here since then the percentages that add up to 1 are those of the dependent variable.

```
> percents<-prop.table(table(FILLER,.SEX),.margin=2)
> percents
.....SEX
FILLER....female.....male
..silence..0.3406375..0.3232932
..uh.....0.3207171..0.4678715
..uhm.....0.3386454..0.2088353
```

You can immediately see that men appear to prefer *uh* and disprefer *uhm* while women appear to have no real preference for any disfluency. However, this is of course not yet a significance test, which we will only deal with in Section 4.1.2.2 below. The function `addmargins` outputs row and column totals (or other user-defined margins):

```
> addmargins(freqs)#.cf.also.colSums.and.rowSums
.....SEX
FILLER....female..male..Sum
..silence...171..161..332
..uh.....161..233..394
..uhm.....170..104..274
..Sum.....502..498..1000
```

**Recommendation(s) for further study**

the functions `ftable` and `xtabs` to generate other kinds of more complex tables (cf. also Chapter 5 below)

*2.1.1. Bar plots and mosaic plots*

Of course you can also represent such tables graphically. The simplest way to do this is to provide a formula as the main argument to `plot`. Such formulae consist of a dependent variable (here: `FILLER: FILLER`), a tilde (“~”), and an independent variable (here: `GENRE: GENRE`), and the following line produces Figure 31.

```
> plot(FILLER~GENRE)
```

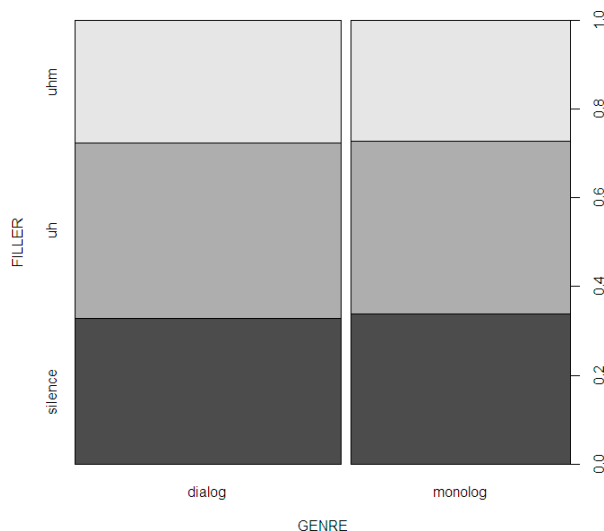


Figure 31. Bar plot / mosaic plot for `FILLER~SEX`

The width and the height of rows, columns, and the six individual boxes represent the observed frequencies. For example, the column for dialogs is a little wider than the columns for monologs because there are more dialogs in the data; the row for *uh* is widest because *uh* is the most frequent disfluency, etc. Other similar graphs can be generated with the following lines:

```
> plot(GENRE, ·FILLER)¶
> plot(table(GENRE, ·FILLER))¶
> mosaicplot(table(GENRE, ·FILLER))¶
```

These graphs are called stacked bar plots or mosaic plots and are – apart from association plots to be introduced below – among the most useful ways to represent crosstabulated data. In the code file for this chapter you will find R code for another kind of useful (although too colorful) graph. (You may not immediately understand the code, but with the help files for these functions you will understand the code; consider this an appetizer.)

### 2.1.2. Spineplots

Sometimes, the dependent variable is nominal/categorical and the independent variable is interval/ratio-scaled. Let us assume that FILLER is the dependent variable, which is influenced by the independent variable LENGTH. (This does not make much sense here, we just assume this for expository purposes and so that I don't have to invent more datasets). You can use the function `spineplot` with a corresponding formula:

```
> spineplot(FILLER~LENGTH)¶
```

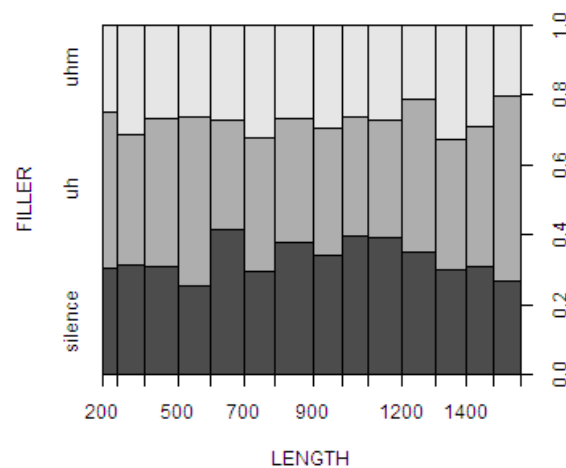


Figure 32. Spineplot for FILLER~LENGTH

The y-axis represents the dependent variable and its three levels. The x-

axis represents the independent ratio-scaled variable, which is split up into the value ranges that would also result from `hist` (which also means you can change the ranges with `breaks=...`; cf. Section 3.1.1.5 above).

### 2.1.3. Line plots

Apart from these plots, you can also generate line plots that summarize frequencies. If you generate a table of relative frequencies, then you can create a primitive line plot by entering the following:

```
> fil.table<-prop.table(table(FILLER,.SEX),.2);.fil.table
.....SEX
FILLER.....female.....male
..silence..0.3406375..0.3232932
..uh.....0.3207171..0.4678715
..uhm.....0.3386454..0.2088353
> plot(fil.table[,1],ylim=c(0,.5),.xlab="Disfluency",.
      ylab="Relative frequency",.type="b")#.column.1
> points(fil.table[,2],.type="b")#.column.2
```

However, somewhat more advanced code in the companion file shows you how you can generate the graph in Figure 33. (Again, you may not understand the code immediately, but it will not take you long.)

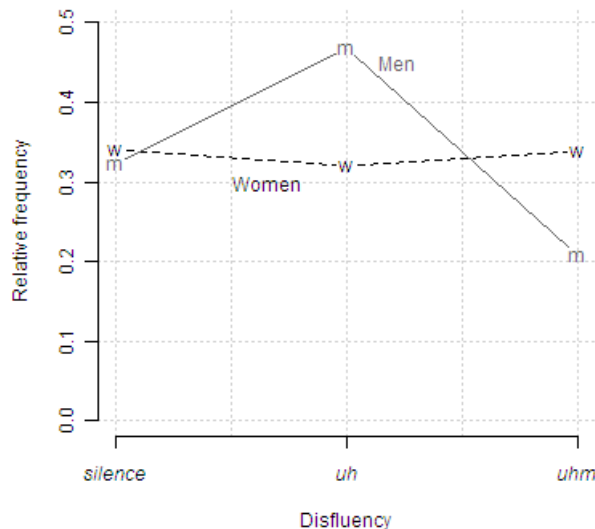


Figure 33. Line plot with the percentages of the interaction of SEX and FILLER

**Warning/advice**

Sometimes, it is recommended to not represent such frequency data with a line plot like the above because the lines ‘suggest’ that there are frequency values between the levels of the categorical variable, which is of course not the case.

**2.2. Means**

If the dependent variable is interval/ratio-scaled or ordinal and the independent variable is nominal/categorical, then one is often not interested in the frequencies of particular values of the dependent variable, but the central tendencies of each level of the independent variable. For example, you might want to determine whether men and women differ with regard to the average disfluency lengths. One way to get these means is the following:

```
> mean(LENGTH[SEX=="male"])
[1] 901.5803
> mean(LENGTH[SEX=="female"])
[1] 928.3984
```

This approach is too primitive for three reasons:

- you must define the values of LENGTH that you want to include manually, which requires a lot of typing (especially when the independent variable has more than two levels or, even worse, when you have more than one independent variable);
- you must know the levels of the independent variables – otherwise you couldn’t use them for subsetting in the first place;
- you only get the means of the variable levels you have explicitly asked for. However, if, for example, you made a coding mistake in one row – such as entering “malle” instead of “male” – this approach will not show you that.

Thus, we use `tapply` and I will briefly talk about three arguments of this function. The first is a vector or factor to which you want to apply a function – here, this is LENGTH, to which we want to apply `mean`. The second argument is a vector or factor that has as many elements as the first one and that specifies the groups to which the function is to be applied. The last argument is the relevant function, here `mean`. We get:

```
> tapply(LENGTH, SEX, mean)
..female....male
928.3984 901.5803
```

Of course the result is the same as above, but you obtained it in a better way. Note also that you can of course use functions other than mean: median, IQR, sd, var, ..., even functions you wrote yourself. For example, what do you get when you use length instead of mean?



**THINK  
BREAK**

You get the numbers of lengths that were observed for each sex.

### 2.2.1. Boxplots

In Section 3.1.3.7 above, we looked at boxplots, but restricted our attention to cases where we have one or more vectors dependent variables (such as town1 and town2). However, you can also use boxplots for cases where you have one or more independent variables and a dependent variable. Again, the easiest way is to use a formula again (cf. also Section 3.2.2.2 below):

```
> boxplot(LENGTH~GENRE, notch=T, ylim=c(0, 1600))
```

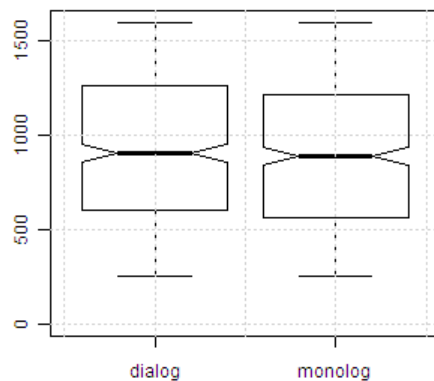


Figure 34. Boxplot for LENGTH~GENRE

(If you only want to plot a boxplot and not provide any further arguments, it is actually enough to just enter `plot(LENGTH~GENRE)`¶: R ‘knows’ you want a boxplot because `LENGTH` is a numerical vector and `GENRE` is a factor.) Again, you can infer a lot from that plot: both medians are close to 900 ms and do most likely not differ significantly from each other (since the notches overlap). Both genres appear to have about the same amount of dispersion since the notches, the boxes, and the whiskers are nearly equally large, and both genres have no outliers.

**Recommendation(s) for further study**

the function `plotmeans` (from the `library(gplots)`) to plot line plots with means and confidence intervals

### 2.2.2. Interaction plots

So far we have looked at graphs representing one variable or one variable depending on another variable. As we have seen in Section 1.3.2.3 above, however, there are also cases where you would want to characterize the distribution of one interval/ratio-scaled variable depending on two nominal/categorical variables. You can obtain the means of the variable level combinations of the independent variables with `tapply`. You must specify the two independent variables in the form of a list, and the following two examples show you how you get the same means in two different ways (so that you see which variable goes into the rows and which into the columns):

```
> tapply(LENGTH, list(SEX, FILLER), .mean)¶
.....silence.....uh.....uhm
female.942.3333.940.5652.902.8588
male...891.6894.904.9785.909.2788
> tapply(LENGTH, list(FILLER, SEX), .mean)¶
.....female.....male
silence.942.3333.891.6894
uh.....940.5652.904.9785
uhm.....902.8588.909.2788
```

Such results are best shown in tabular form in such that you don’t just provide the above means of the interactions, but also the means of the individual variables as they were represented in Figure 28 above. Consider Table 17 and especially its caption. A plus sign between variables refers to just adding *main effects* of variables (i.e., effects of variables in isolation as

when you inspect the two means for SEX in the bottom row and the three means for FILLER in the rightmost column). A colon between variables refers to only the interaction of the variables (i.e., effects of combinations of variables as when you inspect the six means in the main body of the table where SEX and FILLER are combined). Finally, an asterisk between variables denotes both the main effects *and* the interaction (here, all 12 means). With two variables A and B,  $A*B$  is the same as  $A \cdot + \cdot B \cdot + \cdot A:B$ .

Table 17. Means for LENGTH~FILLER\*SEX

	SEX: FEMALE	SEX: MALE	Total
FILLER: UH	940.57	904.98	919.52
FILLER: UHM	902.86	909.28	905.3
FILLER: SILENCE	942.33	891.69	917.77
TOTAL	928.4	901.58	915.04

Now to the results. These are often easier to understand when they are represented graphically. You can create and configure an interaction plot manually, but for a quick and dirty glance at the data, you can also use the function `interaction.plot`. As you might expect, this function takes at least three arguments:

- `x.factor`: a vector/factor whose values/levels are represented on the *x*-axis;
- `trace.factor`: the second argument is a vector/factor whose values/levels are represented with different lines;
- `response`: the third argument is a vector whose means for all variable level combinations will be represented on the *y*-axis by the lines.

That means, you can choose one of two formats, depending on which independent variable is shown on the *x*-axis and which is shown with different lines. While the represented means will of course be identical, I advise you to always generate and inspect both graphs anyway because I usually find that one of the two graphs is much easier to interpret. In Figure 35 you find both graphs for the above values and I for myself find the lower panel easier to interpret.

```
> interaction.plot(SEX, FILLER, LENGTH); grid()
> interaction.plot(FILLER, SEX, LENGTH); grid()
```



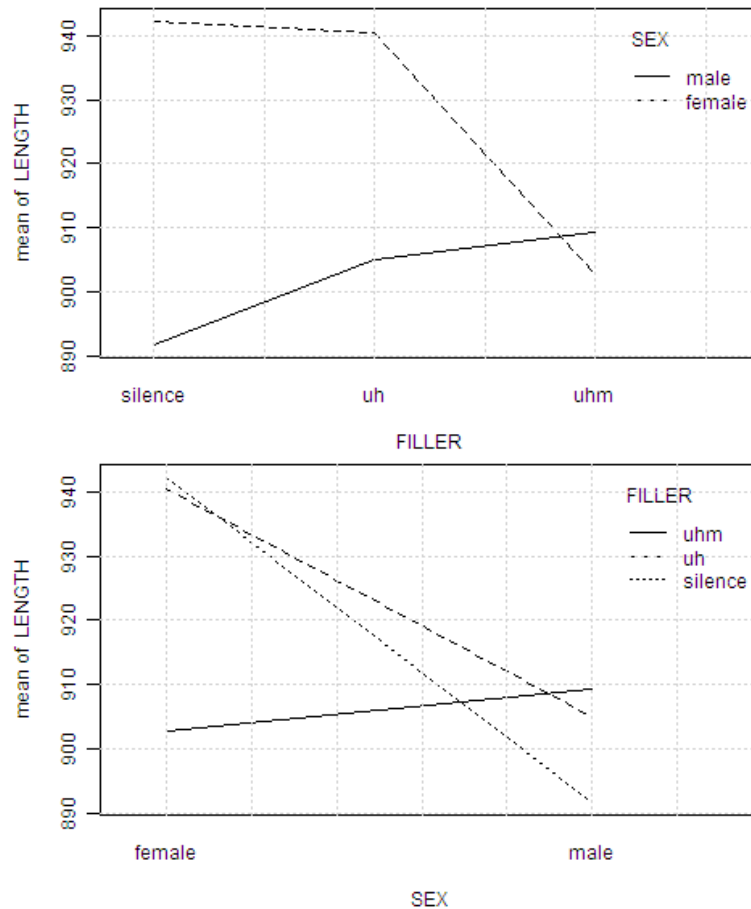


Figure 35. Interaction plot for LENGTH~FILLER:SEX

Obviously, *uhm* behaves differently from *uh* and silences: the average lengths of women's *uh* and silence are larger than those of men, but the average length of women's *uhm* is smaller than that of men. But now an important question: why should you now not just report the means you computed with `tapply` and the graphs in Figure 35 in your study?



**THINK  
BREAK**

First, you should not just report the means like this because I told you above in Section 3.1.3 that you should never ever report means without a measure of dispersion. Thus, when you want to provide the means, you must also add, say, standard deviations (cf. Section 3.1.3.5), standard errors (cf. Section 3.1.3.8), confidence intervals (cf. Section 3.1.5.1):

```
> tapply(LENGTH, list(SEX, FILLER), sd)
.....silence.....uh.....uhm
female 361.9081 397.4948 378.8790
male   370.6995 397.1380 382.3137
```

How do you get the standard errors and the confidence intervals?



**THINK  
BREAK**

```
> se<-tapply(LENGTH, list(SEX, FILLER), sd)/
  sqrt(tapply(LENGTH, list(SEX, FILLER), length)); se
.....silence.....uh.....uhm
female 27.67581 31.32698 29.05869
male   29.21522 26.01738 37.48895

> t<-qnorm(0.025, df=999, lower.tail=F); t
[1] 1.962341
> tapply(LENGTH, list(SEX, FILLER), mean)-(t*se) - # lower
.....silence.....uh.....uhm
female 888.0240 879.0910 845.8357
male   834.3592 853.9236 835.7127
> tapply(LENGTH, list(SEX, FILLER), mean)+(t*se) - # upper
.....silence.....uh.....uhm
female 996.6427 1002.0394 959.882
male   949.0197 956.0335 982.845
```

And this output immediately shows again why measures of dispersion are so important: the standard deviations are large, the means plus/minus one standard error and the confidence intervals overlap clearly, which shows that the differences, which seem so large in Figure 35, are most likely insignificant. You can test this with `boxplot`, which also allows formulae with more than one independent variable (combined with an asterisk for the interaction):

```
> boxplot(LENGTH~SEX*FILLER, notch=T)
```

Second, the graphs should not be used as they are (at least not uncritically) because R has chosen the range of the y-axis such that it is as small as possible but still covers all necessary data points. However, this small range on the y-axis has visually inflated the differences, and a more realistic representation would have either included the value  $y = 0$  (as in the first pair of the following four lines) or chosen the range of the y-axis such that the complete range of LENGTH is included (as in the second pair of the following four lines):

```
> interaction.plot(SEX, FILLER, LENGTH, ylim=c(0, 1000))
> interaction.plot(FILLER, SEX, LENGTH, ylim=c(0, 1000))
> interaction.plot(SEX, FILLER, LENGTH, ylim=range(LENGTH))
> interaction.plot(FILLER, SEX, LENGTH, ylim=range(LENGTH))
```

### 2.3. Coefficients of correlation and linear regression

The last section in this chapter is devoted to cases where both the dependent and the independent variable are ratio-scaled. For this scenario we turn to a new data set. First, we clear out memory of all data structures we have used so far:

```
> rm(list=ls(all=T))
```

We look at data to determine whether there is a correlation between the reaction times in ms of second language learners in a lexical decision task and the length of the stimulus words. We have

- a dependent ratio-scaled variable: the reaction time in ms MS\_LEARNER, whose correlation with the following independent variable we are interested in;
- an independent ratio-scaled variable: the length of the stimulus words LENGTH (in letters).

Such correlations are typically quantified using a so-called coefficient of correlation  $r$ . This correlation coefficient, and many others, are defined to fall in the range between -1 and +1. Table 18 explains what the values mean: the sign of a correlation coefficient reflects the *direction* of the correlation, and the absolute size reflects the *strength* of the correlation. When the correlation coefficient is 0, then there is no correlation between the two

variables in question, which is why the null hypothesis says  $r = 0$  – the two-tailed alternative hypothesis says  $r \neq 0$ .

Table 18. Correlation coefficients and their interpretation

Correlation coefficient	Labeling the correlation	Kind of correlation
$0.7 < r \leq 1$	very high	positive correlation: the more/higher ..., the more/higher ... the less/lower ..., the less/lower ...
$0.5 < r \leq 0.7$	high	
$0.2 < r \leq 0.5$	intermediate	
$0 < r \leq 0.2$	low	
$r \approx 0$	no statistical correlation	
$0 > r \geq -0.2$	low	negative correlation: the more/higher ..., the less/lower ... the less/lower ..., the more/higher ...
$-0.2 > r \geq -0.5$	intermediate	
$-0.5 > r \geq -0.7$	high	
$-0.7 > r \geq -1$	very high	

Let us load and plot the data, using by now familiar lines of code:

```
> ReactTime<-read.table(choose.files(),header=T,sep="\t",
  comment.char="",quote="")
> attach(ReactTime);str(ReactTime)
'data.frame':..20 obs. of..3 variables:
.$CASE.....int..1.2.3.4.5.6.7.8.9.10....
.$LENGTH.....int..14.12.11.12.5.9.8.11.9.11....
.$MS_LEARNER..int..233.213.221.206.123.176.195.207.172....
> plot(MS_LEARNER~LENGTH,xlim=c(0,15),ylim=c(0,300),
  xlab="word.length.in.letters",ylab="Reaction.time.of.
  learners.in.ms");grid()
```

Cf. Figure 36 before you read on – what kind of correlation is that, a positive or a negative one?



**THINK  
BREAK**

This is a positive correlation, because we can describe the correlation between reaction time and word length with a “the more ..., the more ...” statement: the longer the word, the longer the reaction time because when you move from the left (short words) to the right (long words), the reaction

times get higher. But we also want to quantify the correlation and compute the so-called Pearson product-moment correlation  $r$ .

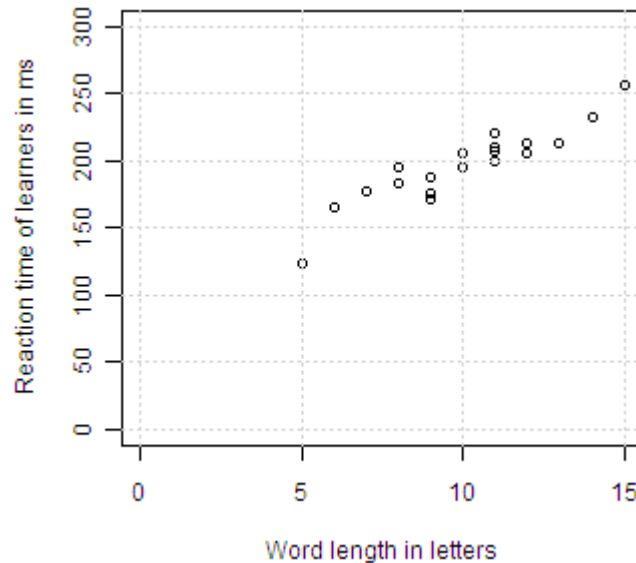


Figure 36. Scatterplot for MS\_LEARNER~LENGTH

First, we do this manually: We begin by computing the *covariance* of the two variables according to the formula in (23).

$$(23) \quad \text{Covariance}_{x,y} = \frac{\sum_{i=1}^n (x_i - \bar{x}) \cdot (y_i - \bar{y})}{n-1}$$

As you can see, the covariance involves computing the differences of each variable's value from the variable's mean. For example, when the  $i$ -th value of both the vector  $x$  and the vector  $y$  are above the averages of  $x$  and  $y$ , then this pair of  $i$ -th values will contribute a high positive value to the covariance. In R, we can compute the covariance manually or with the function `cov`, which requires the two relevant vectors:

```
> covariance<-sum((LENGTH-mean(LENGTH))*(MS_LEARNER-
  mean(MS_LEARNER)))/(length(MS_LEARNER)-1)¶
> covariance<-cov(LENGTH, MS_LEARNER); covariance¶
[1] 79.28947
```

The sign of the covariance already indicates whether two variables are positively or negatively correlated; here it is positive. However, we cannot use the covariance to quantify the correlation between two vectors because its size depends on the scale of the two vectors: if you multiply both vectors with 10, the covariance becomes 100 times as large as before although the correlation as such has of course not changed:

```
> cov(MS_LEARNER*10, .LENGTH*10)¶
[1] 7928.947
```

Therefore, we divide the covariance by the product of the standard deviations of the two vectors and obtain  $r$ :

```
> covariance/(sd(LENGTH)*sd(MS_LEARNER))¶
[1] 0.9337171
```

This is a very high positive correlation,  $r$  is close to the theoretical maximum of 1. In R, we can do all this much more efficiently with the function `cor`. Its first two arguments are the two vectors in question, and the third argument specifies the desired kind of correlation:

```
> cor(MS_LEARNER, .LENGTH, method="pearson")¶
[1] 0.9337171
```

The correlation can be investigated more closely, though. We can try to predict values of the dependent variable on the basis of the independent one. This method is called *linear regression*. In its simplest form, it involves trying to draw a straight line in such a way that it represents the scattercloud best. Here, *best* is defined as ‘minimizing the sums of the squared vertical distances of the observed y-values (here: reaction times) and the y-values predicted on the basis of the regression line.’ That is, the regression line is drawn fairly directly through the scattercloud because then these deviations are smallest. It is defined by a regression equation with two parameters, an intercept  $a$  and a slope  $b$ . Without discussing the relevant formulae here, I immediately explain how to get these values with R. Using the formula notation you already know, you define and inspect a so-called linear model using the function `lm`:

```
> model<-lm(MS_LEARNER~LENGTH); model¶
Call:
lm(formula = .MS_LEARNER ~ .LENGTH)
Coefficients:
```

```
(Intercept).....LENGTH
.....93.61.....10.30
```

That is, the intercept – the y-value of the regression line at  $x = 0$  – is 93.61, and the slope of the regression line is 10.3, which means that for every letter the estimated reaction time increases by 10.3 ms. For example, our data do not contain a word with 16 letters, but since the correlation between the variables is so strong, we can come up with a good prediction for the reaction time this word might result in:

$$\begin{array}{rclclcl} \text{predicted reaction time} & = & \text{intercept} & + & b \cdot \text{LENGTH} \\ 258.41 & \approx & 93.61 & + & 10.3 \cdot 16 \end{array}$$

```
> 93.61+10.3*16
[1] 258.41
```

(This prediction of the reaction time is of course overly simplistic as it neglects the large number of other factors that influence reaction times but within the current linear model this is how it would be computed.) Alternatively, you can use the function `predict`, whose first argument is the linear model and whose second argument is a list called `newdata` that contains a set of those values for the independent variable(s) for which you want to make a prediction. With the exception of differences resulting from me only using two decimals, you get the same result:

```
> predict(model, newdata=list(LENGTH=16))
[1] 258.4850
```

If you only use the linear model as an argument, you get all predicted values in the order of the data points (as you would with `fitted`).

```
> round(predict(model), 2)
.....1.....2.....3.....4.....5.....6.....7.....8
237.88 217.27 206.96 217.27 145.14 186.35 176.05 206.96
.....9.....10.....11.....12.....13.....14.....15.....16
186.35 206.96 196.66 165.75 248.18 227.57 248.18 186.35
.....17.....18.....19.....20
196.66 155.44 176.05 206.96
```

The first value of `LENGTH` is 14, so the first of the above values is the reaction time we expect for a word with 14 letters, etc. Since you now have the needed parameters, you can also draw the regression line. You do this

with the function `abline`, which either takes a linear model object as an argument or the intercept and the slope:

```
> plot(MS_LEARNER~LENGTH, xlim=c(0, 15), ylim=c(0, 300),
       xlab="word length in letters", ylab="Reaction time of
       learners in ms"); grid()
> abline(93.61, 10.3) # or abline(model)
```

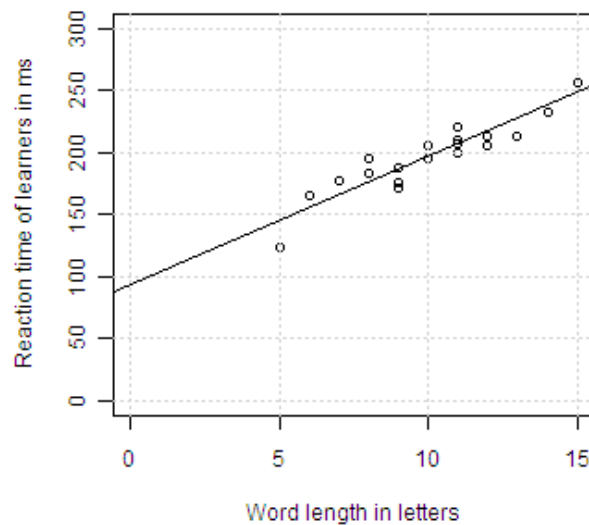


Figure 37. Scatterplot with regressions line for `MS_LEARNER~LENGTH`

It is immediately clear why the correlation coefficient summarizes the data so well: the regression line is an excellent summary of the data points since all points are fairly close to the regression line. (The code file for this chapter will show you two ways of making this graph a bit more informative.) We can even easily check how far away every predicted value is from its observed value. This difference – the vertical distance between an observed  $y$ -value / reaction time and the  $y$ -value on the regression line for the corresponding  $x$ -value – is called a *residual*, and the function `residuals` requires just the linear model object as its argument:

```
> round(residuals(model), 2)
 1  2  3  4  5  6  7  8
-4.88 -4.27 14.04 -11.27 -22.14 -10.35 18.95 0.04
 9 10 11 12 13 14 15 16
-14.35 -6.96 8.34 11.25 7.82 -14.57 7.82 1.65
```



```
.....17.....18.....19.....20
...-1.66..10.56...6.95...3.04
```

You can easily test manually that these are in fact the residuals:

```
> round(MS_LEARNER-(predict(model)+residuals(model)),.2)
.1..2..3..4..5..6..7..8..9..10..11..12..13..14..15..16..17..18..19..20
.0..0..0..0..0..0..0..0..0..0..0..0..0..0..0..0..0..0..0..0
```

Note two important points though: First, regression equations and lines are most useful for the range of values covered by the observed values. Here, the regression equation was computed on the basis of lengths between 5 and 15 letters, which means that it will probably be much less reliable for lengths of 50+ letters. Second, in this case the regression equation also makes some rather non-sensical predictions because theoretically/mathematically it predicts reaction times of around 0 ms for word lengths of -9. Such considerations will become important later on.

The correlation coefficient  $r$  also allows you to specify how much of the variance of one variable can be accounted for by the other variable. What does that mean? In our example, the values of both variables – MS\_LEARNER and LENGTH – are not all identical: they vary around their means (199.75 and 10.3), and this variation was called dispersion and measured or quantified with the standard deviation or the variance. If you square  $r$  and multiply the result by 100, then you obtain the amount of variance of one variable that the other variable accounts for. In our example,  $r = 0.933$ , which means that 87.18% of the variance of the reaction times can be accounted for on the basis of the word lengths. This value is referred to as coefficient of determination,  $r^2$ .

The product-moment correlation  $r$  is probably the most frequently used correlation. However, there are a few occasions on which it should not be used. First, when the relevant variables are not interval/ratio-scaled but ordinal or when they are not both normally distributed (cf. below Section 4.4), then it is better to use another correlation coefficient, for example Kendall's tau  $\tau$ . This correlation coefficient is based only on the ranks of the variable values and thus more suited for ordinal data. Second, when there are marked outliers in the variables, then you should also use Kendall's  $\tau$ , because as a measure that is based on ordinal information only it is, just like the median, less sensitive to outliers. Cf. Figure 38.

In Figure 38 you see a scatterplot, which has one noteworthy outlier in the top right corner. If you cannot justify excluding this data point, then it can influence  $r$  very strongly, but not  $\tau$ . Pearson's  $r$  and Kendall's  $\tau$  for all

data points but the outlier are 0.11 and 0.1 respectively, and the regression line with the small slope shows that there is clearly no correlation between the two variables. However, if we include the outlier, then Pearson's  $r$  suddenly becomes 0.75 (and the regression line's slope is changed markedly) while Kendall's  $\tau$  remains appropriately small: 0.14.

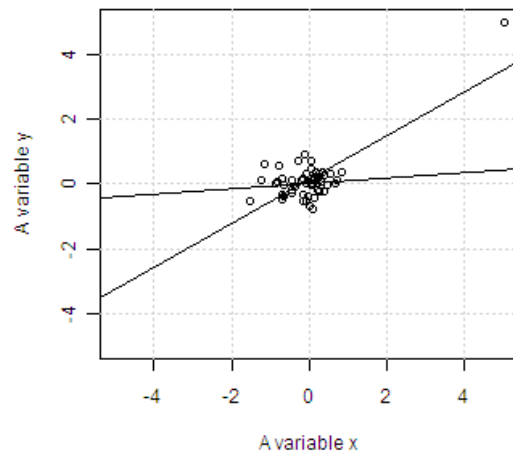


Figure 38. The effect of outliers on  $r$

But how do you compute Kendall's  $\tau$ ? The computation of Kendall's  $\tau$  is rather complex (especially with larger samples and ties), which is why I only explain how to compute it with R. The function is actually the same as for Pearson's  $r$  – `cor` – but the argument `method=...` is changed. For our experimental data we again get a high correlation, which turns out to be a little bit smaller than  $r$ . (Note that correlations are bidirectional – so the order of the vectors does not matter – but linear regressions are not because you have a dependent and an independent variable and it very much matters what goes before the tilde – namely what is predicted – and what goes after it.)

```
> cor(LENGTH, MS_LEARNER, method="kendall")
[1] 0.8189904
```

The previous explanations were all based on the assumption that there is in fact a linear correlation between the two variables. This need not be the case, though, and a third scenario in which neither  $r$  nor  $\tau$  are particularly useful involves nonlinear relations between the variables. This can often be seen by just looking at the data. Figure 39 represents a well-known exam-

ple from Anscombe (1973) (from `<C:/_sflwr/_inputfiles/03-2-3_anscombe.txt>`), which has the intriguing characteristic that

- the means and variances of the  $x$ -variable;
- the means and variances of the  $y$ -variable;
- the correlations and the regression line of  $x$  and  $y$ ;

are all identical although the distributions are obviously very different. In the top left of Figure 39, there is a case where  $r$  and  $\tau$  are unproblematic. In the top right we have a situation where  $x$  and  $y$  are related in a curvilinear fashion – using a linear correlation here does not make much sense.<sup>18</sup> In the two lower panels, you see distributions in which individual outliers have a huge influence on  $r$  and the regression line. Since all the summary statistics are identical, this example illustrates most beautifully how important, in fact indispensable, a visual inspection of your data is, which is why in the following chapters visual exploration nearly always precedes statistical computation.

Now you should do the exercise(s) for Chapter 3 ...

#### Recommendation(s) for further study

- the argument `method="spearman"` for the function `cor` to compute a different kind of rank correlation
- the function `s.hist` (from the library(ade4)) to produce more refined scatterplots with histograms
- Good and Hardin (2006: Ch. 8), Crawley (2007: Ch. 5, 27), and Braun and Murdoch (2008: Section 3.2) for much advice to create good graphs; cf. also <http://cran.r-project.org/src/contrib/Views/Graphics.html>

#### Warning/advice

Do not let the multitude of graphical functions and settings of R and/or your spreadsheet software tempt you to produce visual overkill. Just because you can use 6 different fonts, 10 colors, and three-dimensional graphs does not mean you should. Also, make sure your graphs are understood by using meaningful graph and axis labels, legends, etc.

18. I do not discuss nonlinear regressions here; cf. Crawley (2007: Ch. 18 and 20) for overviews.

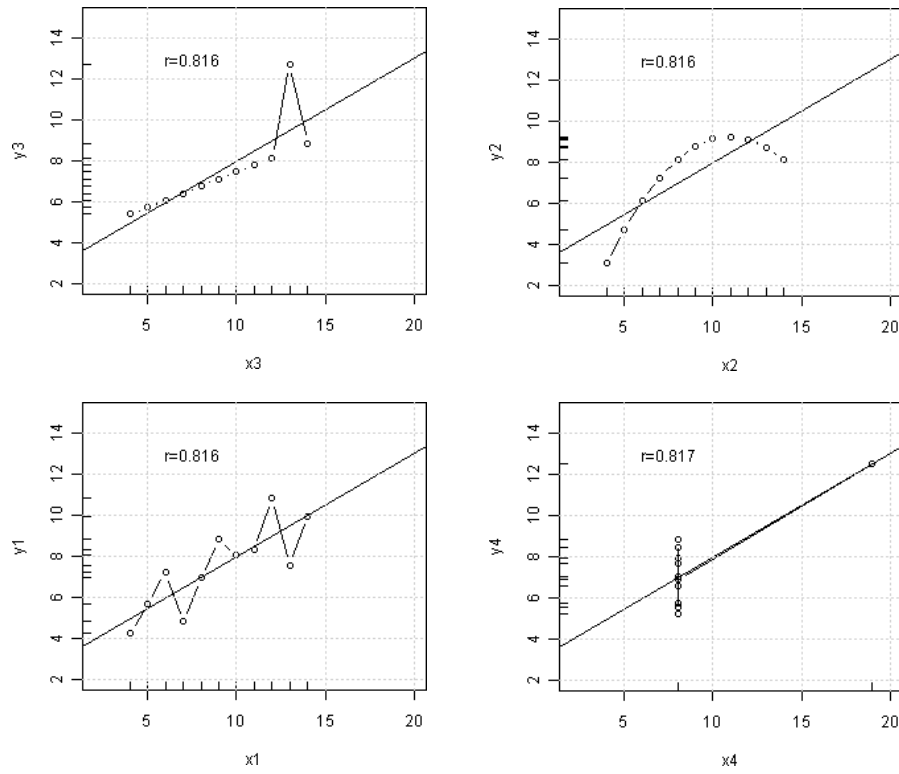


Figure 39. The sensitivity of linear correlations: the Anscombe data