Computer Exercise 1: An introduction to R and Descriptive statistics

This is the first computer exercise for the course Statistics I: Basic Statistics.

1. An introduction to R – Expressions, transformations and new variables

The software R has become one of the most widely used statistical softwares in the world. This is partly because, as open software, it is free for anyone to download and partly because it has become quite powerful. Unfortunately, it is not that user-friendly so it takes some time to get used to working with it, but once you are over the threshold it becomes a valuable tool for advanced data analysis.

The simplest way to get access to it is to visit the R-project website www.r-project.org, where you can download the latest version and also read manuals and get information about other resources. This brief introduction makes no claim to be exhaustive, but rather to offer a quick way to get started. For those who want a more thorough introduction, read *An Introduction to R* found under "Manuals" on the R-project website or the book *Introductory Statistics with R* by Peter Dalgaard.

On a Windows platform, you start R from the Program Menu. When you do that, a console window opens where you enter commands and where results from analyses are displayed. The program is interactive in the sense that you enter various commands at the prompt > and get results immediately when pressing "Enter".

You can enter arithmetic expressions like

```
> 5+2*3.6
```

Thus, this works just like a pocket calculator, and the result is displayed immediately after pressing "Enter":

```
 [1] 12.2
```

The common arithmetic operators +, -, * and / are available as well as common mathematical functions like *exp*, *sqrt* and *log* (which denotes the natural logarithm).

The mathematical expression 5^3 is computed as

```
> 5^3
[1] 125
```

If you want to save your R code so that you can directly use it next time, you can save the code in a script file. To open a new script window: From the main menu, select File → New Script (in RStudio, select File → New File → R Script). This gives you a script window, where you can enter your code. To save a script, click on the script window, and then go to File → Save As… in the menu bar.

To execute one line of the script window: Position the cursor on the line and then press Ctrl-R to execute it (in RStudio, press Ctrl-Enter).

To execute several lines of the script window: Highlight the lines using your mouse; then press Ctrl-R to execute them (in RStudio, press Ctrl-Enter).

The operator `<-` is the *assignment operator*. If you want to assign the value 10 to the variable x, then just assign a value to the name of the variable and R will create the variable:

```
> x <- 10
```

When you define a variable at the command prompt like this, you can use x in your computations. For example,

```
> y <- 3*sqrt(5)
> x + y
[1] 16.7082
```

In statistics, you often have sample data consisting of observations on a variable. R conveniently handles this using *vectors*. For assigning observations (in kg) to a weight variable, we may use the command:

```
> weight <- c(5.2,5.3,3.9,4.2,5.0)
```

(Assuming that you wrote the above R code in the script window, press Ctrl-R (or Ctrl-Enter) and see what happens in the Console window. Write `weight*10` in the Console and press Enter. What happens?)

The c in the function `c(...)` stands for *concatenate* (or *combine*), and this function provides the easiest way to construct vectors.

The command 0:4 creates a vector consisting of the integers from 0 to 4:

```
> n <- 0:4
> n
[1] 0 1 2 3 4
```

The command `seq(0,4,1)` creates the same vector (`from=0, to=4, by=1`):

```
> seq(0,4,1)
```

You can also use a similar expression to extract parts of vectors. The command

```
> weight[1:3]
[1] 5.2 5.3 3.9
```

yields the first three values of the vector `weight`.

If you want to select specific observations according to a certain condition, e.g., all observations that are smaller than 5, then write

```
> weight[weight<5]
[1] 3.9 4.2
```

2. Data handling

A data frame is the most common way of storing data in R, and if used systematically makes data analysis easier. Under the hood, a data frame is a list of equal-length vectors. You create a data frame using `data.frame()`, which takes named vectors as input:

```
> df <- data.frame(x = 1:3, y = c(3.2,1.7,6.2))
> df
  x   y
1 1 3.2
2 2 1.7
3 3 6.2
```

The rows in a data frame represent the objects (or individuals) and the columns represent the variables. In some sense, the data frame may be regarded as the R version of a spreadsheet. In R, there are several ways to read (and write) spreadsheets.

For example, Excel offers many options for saving your data sets and one of them is the tab-delimited text file or *.txt file. If your data is saved as such, you can use one of the easiest and most general options to import your file to R: the `read.table()` function.

On the course homepage, you find the tab delimited text file `cats.txt` containing information on a sample of 57 domestic cats. For importing the data, one may use the command

```
> cats <- read.table("enter the file's web address",sep="\t",header=TRUE)
```

Make sure that the web address links directly to the file and not to a web page that links to the file. A better way is usually to download the file to a directory on your hard drive. For example, if the file cats.txt is located in C:\\mydata\\cats.txt, then import the data file into R as follows:

```
cats <- read.table("C:\\mydata\\cats.txt",sep="\t",header=TRUE)
```

In addition, you can read in files using the `file.choose()` function in R. After typing in this command in R, you can manually select the directory and file where your dataset is located:

```
cats <- read.table(file.choose(),sep="\t",header=TRUE)
```

The two extra arguments, `sep="\t"` and `header=TRUE`, are called *options* and are needed to get things right. The option `sep="\t"` in this case tells R that *cats.txt* is tab delimited and `header=TRUE` means that the first row in the data file contains headers, which are used as variable names in the data frame.

Press Ctrl-R and look at the console window. Write `cats` in the Console, press Enter, and look at what you get. The contents of your Console should look like this:

```
> cats
Name      Breed                    Sex   Neutered  Age       Weight
1 Messi   Domestic short-haired    male  yes       1         5.2
2 Forlan  Domestic short-haired    male  yes       1         5.3
3 Galadriel Domestic short-haired female yes      3         3.3
4 Gandalf Domestic short-haired    male  yes       3         3.3
          ...
```

Note, in R you get all rows of the data, if you want to see only the first 4, use:

```
> head(cats, 4)
```

You see that you have six different variables for the cats: 1) The name of the cat, 2) the breed, 3) the sex, 4) whether it has been neutered, 5) the age in years, and 6) the weight in kg.

A convenient way of quickly getting an overview of a data frame is through the function `summary(...)`.

```
> summary(cats)
Name                Breed                          Sex       Neutered
Alfons : 1          Birman : 2                     female:27 no :15
Anton : 1           Burmese : 3                    male :30  yes:42
Ask : 1             Domestic short-haired:42
Astrid : 1          Norwegian forest : 3
Bons : 1            Ragdoll : 2
Brorsan: 1          Somali : 5
(Other):51

Age                 Weight
Min. : 1.000        Min. :1.500
1st Qu.: 2.000      1st Qu.:3.300
Median : 5.000      Median :4.800
Mean : 4.965        Mean :4.488
3rd Qu.: 7.000      3rd Qu.:5.300
Max. :12.000        Max. :7.600
```

For the numerical variables Age and Weight, you get the summary statistics minimum and maximum, mean and median, and the first and third quartiles. For the categorical variables, you get a small frequency table.

Individual variables can be extracted using the $ sign:

```
> cats$Sex
[1] male male female male female female female female male
[10] male female male female female male male female male
[19] female female female male male male female female female
[28] male male male female male male male female male
[37] female male male female female male male female female
```

```
[46] female male male female female male female male female
[55] male male male
Levels: female male

> cats$Age
[1] 1 1 3 3 6 7 7 2 5 4 5 2 2 3 2 2 6 2 8 3 2
[22] 8 10 4 5 9 7 3 3 2 2 3 7 8 7 6 4 7 7 7 2 5
[43] 3 8 5 2 10 2 2 10 8 12 7 6 6 3 7
```

Here, R automatically interprets all columns containing numbers as numerical and all other columns as categorical. Sometimes you want a numerical variable to be interpreted as categorical and in that case you can transform it using the function `factor(...)`:

```
> factor(cats$Age)

> factor(cats$Age)
[1] 1 1 3 3 6 7 7 2 5 4 5 2 2 3 2 2 6 2 8 3 2
[22] 8 10 4 5 9 7 3 3 2 2 3 7 8 7 6 4 7 7 7 2 5
[43] 3 8 5 2 10 2 2 10 8 12 7 6 6 3 7
Levels: 1 2 3 4 5 6 7 8 9 10 12
```

Usually you will add the new variable to the existing data frame:

```
cats$Age.F<-factor(cats$Age)
mean(cats$Age)
mean(cats$Age.F)
```

Note: When you compute the mean cat age using `mean(cats$Age)`, R will provide the desired answer. However, if you use `mean(cats$Age.F)`, it will not work. The reason is that in `cats$Age.F`, 2 is interpreted as a character (a category) and not as a number, and you cannot carry out arithmetic calculations on this variable anymore.

Sometimes in statistics we want to do the analysis for data on the log-scale. The logarithm of weight is computed as:

```
log(cats$Weight)
```

Again, we may add the new variable to the existing data frame:

```
cats$logweight<-log(cats$Weight)
```

To create age classes, we may use the function `cut(...)`:

```
> cats$Age.class <- cut(cats$Age,seq(0,12,2))
> cats$Age.class
[1] (0,2] (0,2] (2,4] (2,4] (4,6] (6,8] (6,8]
[8] (0,2] (4,6] (2,4] (4,6] (0,2] (0,2] (2,4]
[15] (0,2] (0,2] (4,6] (0,2] (6,8] (2,4] (0,2]
[22] (6,8] (8,10] (2,4] (4,6] (8,10] (6,8] (2,4]
[29] (2,4] (0,2] (0,2] (2,4] (6,8] (6,8] (6,8]
[36] (4,6] (2,4] (6,8] (6,8] (6,8] (0,2] (4,6]
[43] (2,4] (6,8] (4,6] (0,2] (8,10] (0,2] (0,2]
```

```
[50] (8,10] (6,8] (10,12] (6,8] (4,6] (4,6] (2,4]
[57] (6,8]
Levels: (0,2] (2,4] (4,6] (6,8] (8,10] (10,12]
```

Note, the function `seq(0,12,2)` creates a sequence of integers from 0 to 12 with interval length 2. When used as second argument to `cut(...)`, it creates the cut points into which the `Age` variable is to be cut. Note, the right end point is included in the class by default.

Question: In which age class is cat number 29 located? The above output will give you the answer.

To select data in a certain age class, one may use the following command:

```
youngcats<-cats[cats$Age.class=="(0,2]",]
```

If we want to do this only for the weight variable (rather than for the whole data frame), then one may use one of the following two commands:

```
# Weight is the 6th variable (or column) in cats
youngcats.weight<-cats[cats$Age.class=="(0,2]",6]
young.cats.weight<-cats$Weight[cats$Age.class=="(0,2]"]
```

When writing a script, it is useful to add comments to describe what you are doing by inserting a hashtag # in front of a line of text. R will see anything that begins with # as text instead of code, so it will not try to run it, but the text will provide valuable information about the code for whoever is reading your script (including future you!).

If you want to keep your R code, do make sure you save your script file!

3. Descriptive statistics

There is a whole range of specialized functions in R that quickly yields tables, graphs and numerical measures of data sets. Most of them can be used with only a single argument, but there usually are a number of extra options you can use to modify the output. Detailed information of each function can be obtained through the `help(...)` function. If you, for instance, want detailed information on how to use the function `read.table(...)`, you write

```
> help(read.table)
```

and a browser window opens with the manual page for `read.table(...)`. In RStudio it opens in the help window.

The command

```
summary(weight)
```

yields some of the numerical measures of location that we have discussed in class. There are also separate functions `min(...)`, `max(...)`, `mean(...)`, `median(...)`, and

`quantile(...)` (used for computing quartiles and percentiles) that calculates these measures. The first four of these functions only require one variable as an argument:

```
> min(weight)
[1] 3.9
> max(weight)
[1] 5.3
> mean(weight)
[1] 4.72
> median(weight)
[1] 5
```

while `quantile(...)` requires a second argument for the desired percentage (but expressed as a proportion):

```
> quantile(weight, 0.25)
25%
4.2
> quantile(weight, 0.75)
75%
5.2
```

The variance and standard deviation can be obtained using the functions `var(...)` and `sd(...)`:

```
> var(weight)
[1] 0.397
> sd(weight)
[1] 0.6300794
```

What do you expect to get if you type:

```
> sqrt(var(weight))
```

There is no specific function for the inter-quartile range (IQR), but it is rather straightforward to calculate it using the difference between the 75th and the 25th percentile (third and first quartile) as computed with the `quantile(...)` function above.

Use the aforementioned R functions to find the following numerical measures for the age of cats:

Mean	Sd	Median	IQR
4.964912	2.731855	5	5

Frequency tables can be created using the function `table(...)`. As long as the variable is categorical, it works fine, but if you apply the function to a numerical variable, R first converts it to a categorical variable which may create strange results.

```
> table(cats$Sex)
female   male
    27     30
```

What do you think will happen if you apply it for Breed, Age and Weight? Think about it and then run the code:

```
table(cats$Breed)
table(cats$Age)
table(cats$Weight)
```

This function can also create so-called cross tables where frequencies of combinations of two (or more) variables can be obtained. One can, for instance, get frequencies of neutered cats for each sex:

```
> table(cats$Sex,cats$Neutered)
          no        yes
female    15        12
male      0         30
```

If we want to, we can add a third variable, too:

```
table(cats$Breed, cats$Sex, cats$Neutered)
```

We might want to know if the mean age is different for neutered and unneutered cats. For doing this, we can use the `mean` function:

```
mean(cats$Age[cats$Neutered=="no"])
mean(cats$Age[cats$Neutered=="yes"])
```

A somewhat more elegant solution is to use the aggregate function:

```
aggregate(Age~Neutered, data=cats, FUN=mean)
```

If we want to compare the standard deviations rather than the means, then use:

```
aggregate(Age~Neutered, data=cats, FUN=sd)
```

For comparing both means and standard deviations, we may use the following command:

```
aggregate(Age~Neutered, data=cats, FUN=function(x) c(mean=mean(x),
sd=sd(x)))
```

Now, use the aggregate function to get the following output:

```
     Sex Weight.min Weight.Q1.25% Weight.median Weight.Q3.75% Weight.max
1 female      1.500         3.000         3.300         4.250      5.500
2   male      3.300         5.000         5.200         5.675      7.600
```

4. Plots

The standard function for creating graphs and plots is `plot(...)`. This is a very flexible function in the sense that you can design your graphs more or less exactly the way you want it. For example, it is quite easy to create frequency charts by combining `plot(...)` and `table(...)`:

```
plot(table(cats$Breed))
```

As you see, a separate graphical window appears where the graph is displayed. The graph can be saved under the "File" menu in a number of different formats.

We can use the `plot` function with `table` to plot cross tables:

```
plot(table(cats$Breed,cats$Sex))
```

For comparing male and female cats with respect to proportions of cats that belong to different breeds, we may use the plot:

```
plot(table(cats$Breed, cats$Sex))
```

or, possibly,

```
plot(table(cats$Breed,as.factor(cats$Sex)))
```

Using plot to make a graph of one numerical variable is not that useful unless the order of the data has a meaning:

```
plot(cats$Age)
```

We will now look at three graphs for the age of cats. The following code will put the next three plots in the same window:

```
par(mfrow=c(1,3))
```

We could plot the factor version of the variable Age (Age.F):

```
plot(cats$Age.F)
```

Histograms for numerical variables are easily constructed with the `hist(...)` function:

```
hist(cats$Age)
```

For histograms, R automatically tries to find suitable class limits, and is usually pretty good at it. If needed, you can change the limits yourself using the option breaks. By using the following breaks, where each bin is one age, we (almost?) get the same pattern as for the barchart of Age.F:

```
hist(cats$Age, breaks=seq(from=0.5, to=12.5, by=1))
```

For getting one graph per window, then write

```
par(mfrow=c(1,1))
```

Some of the most commonly used options for the `plot` function are `main`, `xlab` and `ylab`, which can be used to put a main title and labels for the axes in the plot:

```
hist(cats$Weight, main="Histogram of Cats", xlab="Weight")
```

The `hist` function shows you by default the frequency of a certain bin on the y-axis. However, if you want to see how likely it is that an interval of values of the x-axis occurs, you will need a probability density rather than frequency. We thus want to ask for a histogram of proportions. You can change this by setting the `freq` argument to false

```
hist(cats$Weight, main="Histogram of Cats", xlab="Weight",
ylab="Proportion", freq=FALSE)
```

If we want to compare male and female cats, it can be useful to make separate histograms for males and females:

```
par(mfrow=c(1,2))
hist(cats$Weight[cats$Sex=="female"], main="female
cats",xlab="Weight")
hist(cats$Weight[cats$Sex=="male"], main="male cats",xlab="Weight")
par(mfrow=c(1,1))
```

Boxplots are created with the `boxplot(...)` function:

```
boxplot(cats$Weight , main="Weight of cats")
```

The boxplot uses the measurements from the `summary` function, except for the mean value. If we want to, we can add the mean value to the plot:

```
points(mean(cats$Weight),col="red",pch=20)
```

Boxplots are mostly used for comparisons. For example, we may want to compare male and female cats:

```
boxplot(cats$Weight~cats$Sex, main="Weight")
```

Scatter plots are often used to observe relationships between continuous variables. For example,

```
plot(cats$Age, cats$Weight)
```

Scatter plots display two (continuous) variables. It is possible to introduce a third (categorical) variable in a scatter plot using color:

```
plot(cats$Age, cats$Weight, col=cats$Sex)
```

Possibly, you may need to change the previous command to

```
plot(cats$Age, cats$Weight, col=as.factor(cats$Sex))
```

5. Do it yourself

On Canvas, you will find home assignment 1.