EE502 Statistics and Study Designs

Lab Manual for Analysis of Biological Data

This book contains instructions for most lab exercises for EE502 Statistics and Study Designs and is based on Whitlock 2008: BIOL300, the biostatistics course at University of British Columbia (UBC), Vancouver, Canada. It is intended to complement, not to replace, the text *Analysis of Biological Data*, by Whitlock and Schluter 2015. The labs contain a mix of data collection, computer simulation, and analysis of data. The instructions were originally based on using a computer program called *JMP* (pronounced "jump"), but has for our EE502 course been adapted to using *R*.

All data described in these labs are real, taken from actual experiments and observations reported in the scientific literature. References for each paper are given at the end of the manual.

This lab manual is a work-in-progress. Suggestions for improvements are welcome.

Many of the labs use java applets, and these are to be found on the web. In order to save you from typing those URLs, all the references are provided from the BIO 300 lab page at <http://www.zoology.ubc.ca/~whitlock/bio300/LAB300pdf/LabOutline300.html>.

1. Introduction to statistics on the computer and graphics

# Goals

* Get started on the computers, learning how to start with *R* and *RStudio*
* Collect a data set on ourselves for future use
* Make graphs, such as histograms, bar charts, box plots, scatter plots, and dot plots.
* Learn to graph data as the first step in data analysis

# Quick summary from text (see Chapter 2 in Whitlock and Schluter)

* Computers make data analysis faster and easier. However, it is still the human's job to choose the right procedures.
* Variables are either numerical (measured as numbers) or categorical (describing which category an individual belongs to).
* Graphing data is an essential step in data analysis and presentation. The human mind receives information much better visually than verbally or mathematically.
* The distribution of categorical variables can be presented in a bar chart. The distribution of numerical variables can be presented in a histogram, a box plot, or cumulative frequency plot.
* The relationship between two numerical variables can be shown in a *scatter plot*. The relationship between two categorical variables can be shown in a *grouped bar chart* or a *mosaic plot*. The association between a numerical variable and a categorical variable can be shown with *multiple histograms, grouped cumulative frequency plots*, or *multiple box plots*.
* A good graph should be honest and easy to interpret, with as much information as needed to interpret the graph readily available. At the same time, the graph should be uncluttered and clear.

# Activities

# Learning the Tools

1. Downloading the right software

1. Download *R* from <http://www.r-project.org/>. Click on *download R* in the top left corner and choose your preferred mirror (e.g., from the University of Bergen, <https://cran.uib.no/>). Download the correct version for your computer.
2. Download *RStudio* from <https://www.rstudio.com/products/rstudio/>. You want to choose the free *RStudio Desktop Open Source Edition*. Download the correct version for your computer.

**2. Make your contribution to the student data set**

* <https://forms.gle/6Gpg19aJRm61ygVw8>

**3. A short introduction to R and RStudio**

R is a free programming language developed for statistical computing and graphics. Today it is widely regarded as the standard language for statistical computing in academia. If you want to continue in any form with research, chances are good that R will be a big part of your daily life.

R itself can be run in any command line interface, however, to make it more user friendly, we will be using RStudio which will provide graphical user interface that makes a lot of things a little easier to work with.

To be able to use R, we first need to introduce some of the basic concepts and the basic syntax (i.e., the language part of R). Most use of R is centred around *objects* and *functions*. **Objects** store information. This can be everything from simple numbers to more complex data and results as well as graphics. **Functions** are all the commands you can use to tell R what to do. This includes data handling and editing, calculations to complex modelling and creation of graphics.

Example: Storing the number 1 in the object x

a <- 1

Example: Storing the result of 1+1 in the object y

b <- 1+1

Now, x contains 1 and y contains 2. We could now compute a + b and would get 3 as a result. No matter what we want to store in an object, the syntax follows these easy examples:

Object name <- content

Functions are made up of two different parts, the name of the function and arguments of the function:

Function(argument1, argument2,…)

The name specifies which function R should use. The arguments available change with the different functions and are used to give details on what data to use and what settings within the function to use.

Example: Calculate the mean of the numbers 1-10

mean(1:10)

1:10 is the R way of writing 1-10. We could also combine our knowledge of objects with our new knowledge of functions to calculate the same mean:

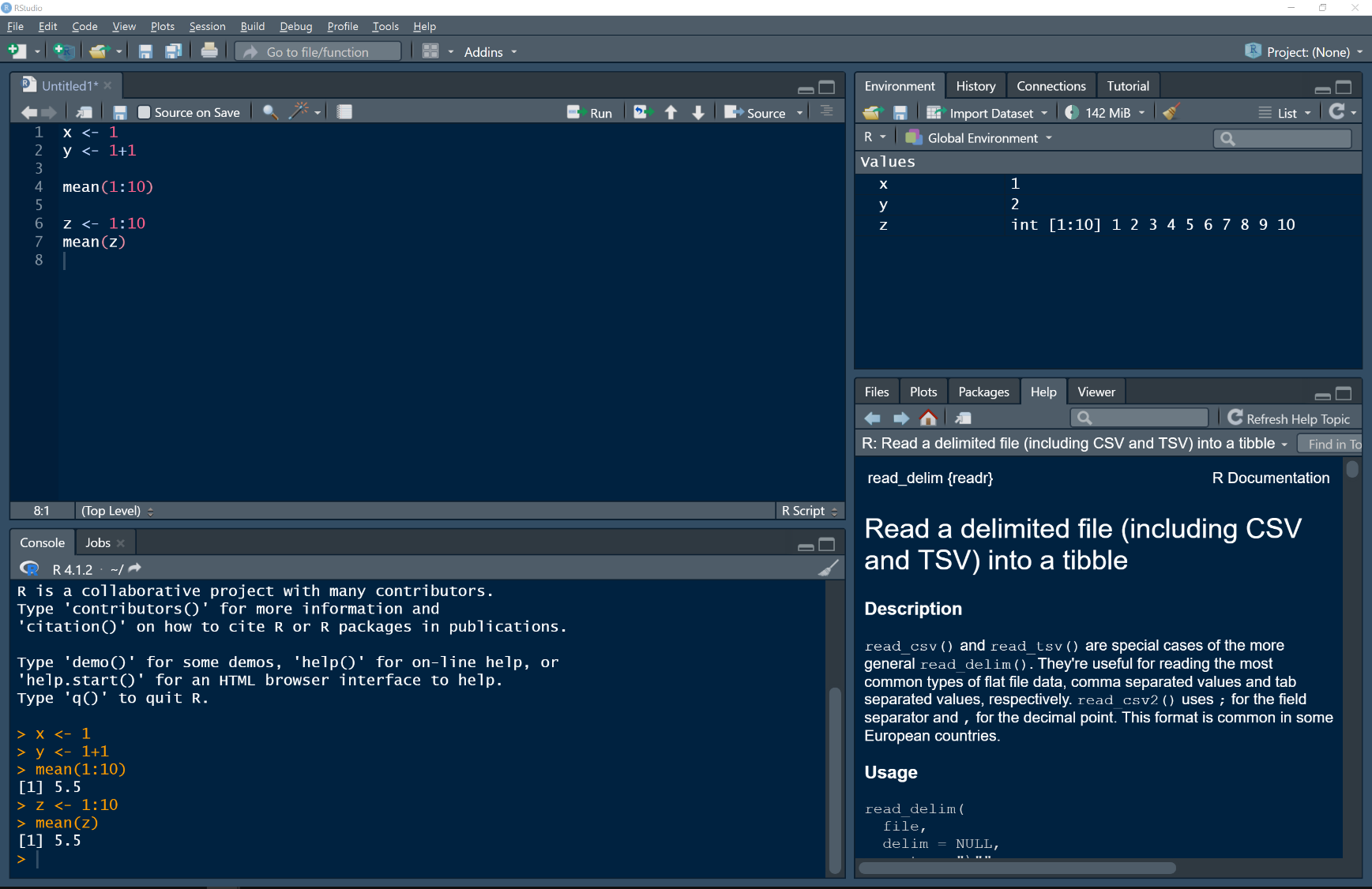
c <- 1:10

mean(c)

Both ways will output the correct result of 5.5.

One last important thing about R syntax is that it is case sensitive, meaning it will differentiate between small and capital letters! This means you need to be careful and use the correct case.

So far so good with the most basic syntax of R. All of this might sound a bit confusing, but do not worry, you will be able to understand more of the language when working through these data labs.

Before continuing to the exercises, we need to look at the interface of RStudio. The interface is made up of four different panes that are visible at a time. Note that you can rearrange the panes based on personal taste, so your RStudio may look a bit different to the screenshots presented here. In the screenshot below you see the basic RStudio window with the *script* in the top left, the *console* in the bottom left, the *environment* in the top right and the help section in the bottom right.

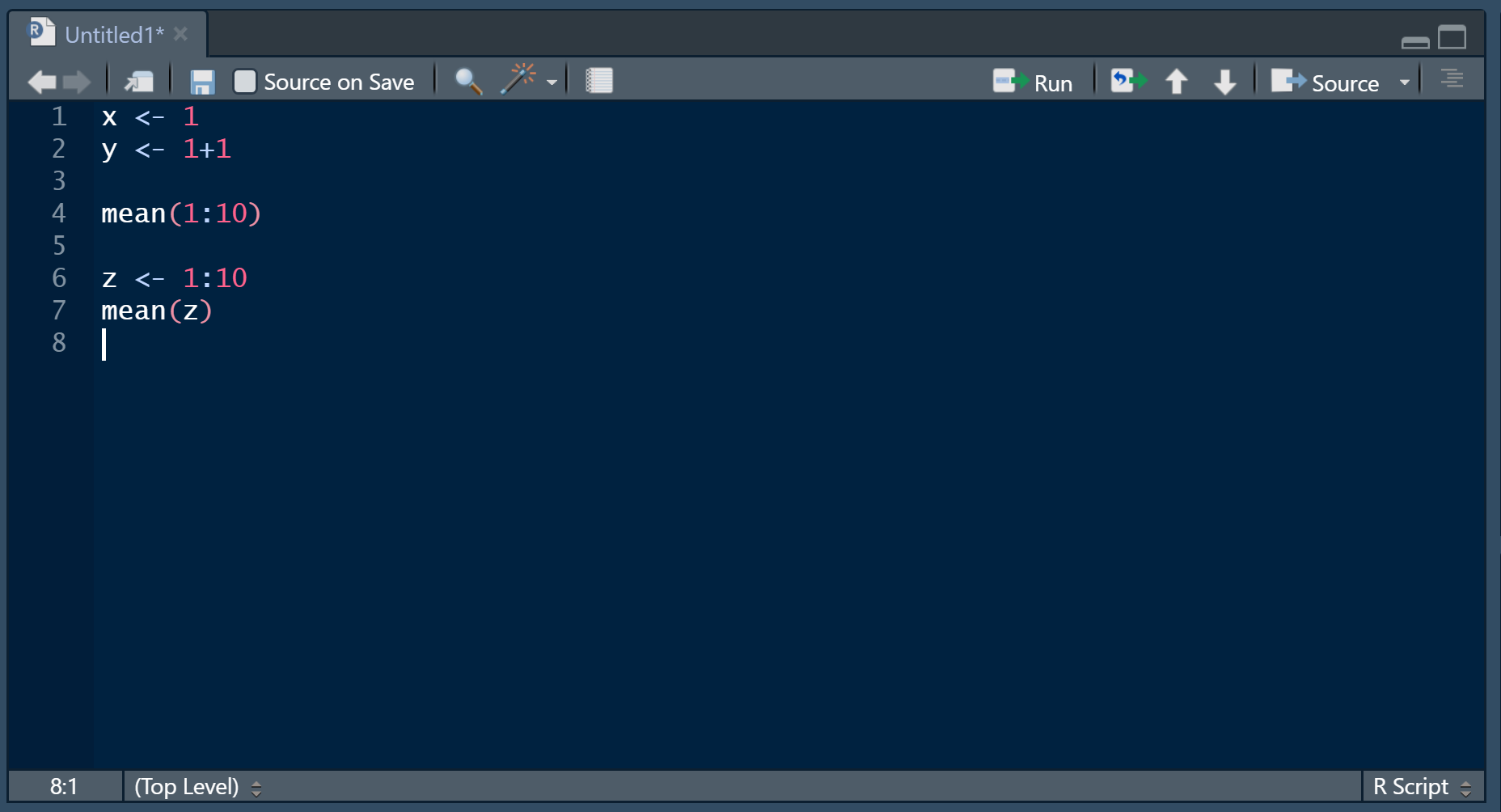
*Overview over the RStudio interface*

**The script**

An R script is the document where you store the code you are writing. It works like a normal text or word document, and you can save and reload this file. If you do not see the script when starting RStudio, you can click on the white icon in the top left corner with a green plus symbol to create a new script (Ctrl+Shift+N / Cmd+Shift+N).

When you want to run a line of code (meaning activating a line of code), you can either click the button labelled “run” above the script or use the keyboard shortcut Ctrl+Enter / Cmd+Return.

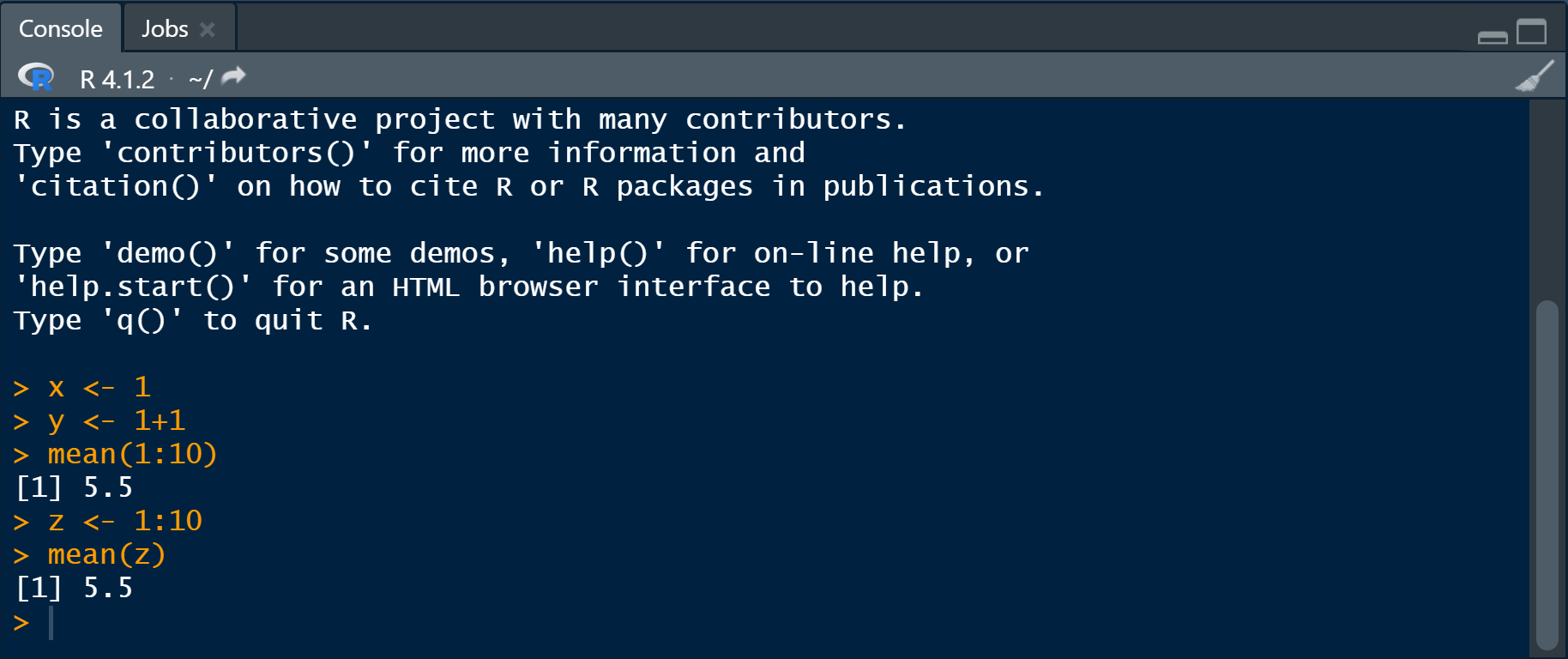
Saving an R script works very much the same as a Word document. You can use the menu (File -> Save) or keyboard shortcuts (Ctrl+S / Cmd+S)

****

*The R script in RStudio*

**The console**

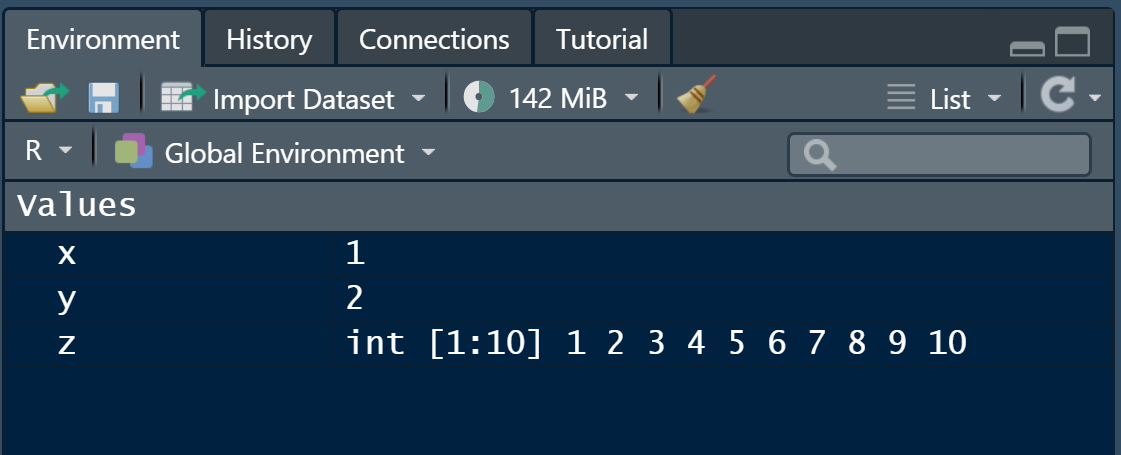
The console keeps track of the code that has been run. It also shows any outputs (apart from graphics) that are the result of your code. In this example you can see that we created an object x, an object y and an object z. Additionally, we see the outcome of our mean calculations.

****

*The console in RStudio*

**The environment pane**

The environment pane shows all the objects that have been stored. Here we can see that object x contains the number 1, object y contains the number 2 and object z contains all numbers from 1 to 10.

****

*The ‘environment’ pane in RStudio*

There are more panes, however, these three are the most important ones to start using RStudio.

4. Opening a data file

There are ways of creating data files within R but in most cases, we will have other files that we want to import into RStudio. R can read almost every kind of file that contains text, but usually data sets are entered in Excel and then exported into a format you can read in *R*, such a comma- or tab-delimited text. To do this in Excel, choose “Save as”, then select “CSV (Comma-delimited) (\*.csv)) It is a text-only format read by almost any program, including *R*.

You can import a data file by clicking on "Import Dataset" and then choosing "From Text (base)". You will be presented a dialogue that lets you change some import settings. Having the right settings is essential for a correct data import. To know which settings are correct, you need to look at the data file you want to import.

**Name:** The name your dataset will have in RStudio. It is recommended to choose a short but clear name for your datasets.

**Encoding:** Automatic should be alright in most cases. If you experience weird symbols in the dataset after the import, you can try different encodings, UTF-8 being the most common one.

**Heading:** This one is important! Does the first row in your dataset contain column headings (yes) or data (no)?

**Row names:** If your first column contains row names rather than data, you would want to choose "Use first column".

**Separator:** This one is also important! In text-based files, columns are separated by a specific character. Usually this is either a comma, a semicolon or a tab. You need to have look in the file to see which is correct.

**Decimal:** What character is used as a decimal character? Usually, data files follow the English use of a period. If you have a "Norwegian" dataset which uses a comma, you need to change this setting.

**Quote:** Usually not important, leave at standard setting

**Comment:** Usually not important for data sets, leave at standard setting

**na.strings:** This is also an important one. R uses the concept of NA values. Na stands for "not available" and is used in cases where you don't have data. For R to understand this correctly you need to state how missing data fields (NA fields) are described in your dataset. Common ways of writing this are for example "NA" or ".".

Open Canvas and navigate to our room “EE502-1 22H Statistics and Study Design”, open the “Filer” and go to “02. DataLabs”. This is where you will find the data sets needed for this class (Datalab 1), and for the Data Labs all datasets are as .csv files in this folder.

The first dataset we want to import is a file called "*titanic.csv*". This file has information on all the passengers of the RMS Titanic on its initial and disastrous run. Follow the steps outlined earlier in this section to import this file. To check the correct settings required, you may want to open the file in notepad first to get an understanding of how the file is built.

Hint: in this example you may want to pay extra attention to the *na.string* setting.

After you have dialled in the correct settings and clicked “Import”, you will see that your dataset has been opened in a new tab next to your R script. If you see anything weird here, you may want to check your import settings.

If you know that you want to import the same dataset later, you may want to write the code for importing into your script so that you can import the file the next time without the need to go through the import dialogue again. To do this open the “History” pane which is located next to the “Environment” pane. Here you see every line of code that has been run and even if you did not type any code while following the import dialogue, Rstudio has written the code for you in the background. The line of code that has imported the dataset will look something like this:

Name of dataset <- read.csv(“location of file”, import settings chosen)

titanic <- read.csv("C:/Users/tietg/Desktop/Statistikk/Datafiler/titanic.csv", na.strings=".")

Now click on that line in the history pane and then click on the button labelled “To source”. Source describes your R script, so this button will copy this exact line of code to your script. Next time you open the script you can simply run this code (Ctrl+Enter / Cmd+Return) and import the dataset without opening the import dialogue.

If you look closely in the history pane, you will see that after the import code, another line of code was run:

View(name of dataset) / e.g. View(titanic)

This code opens the dataset in the new tab next to the script. NB: Remember that R is case sensitive! Here, the function View() has a capital V, meaning writing view() with a small v is not going to open the dataset (but rather resulting in an error, you can try to see how that looks like!)!

A bit more detailed information about functions and how to understand them.

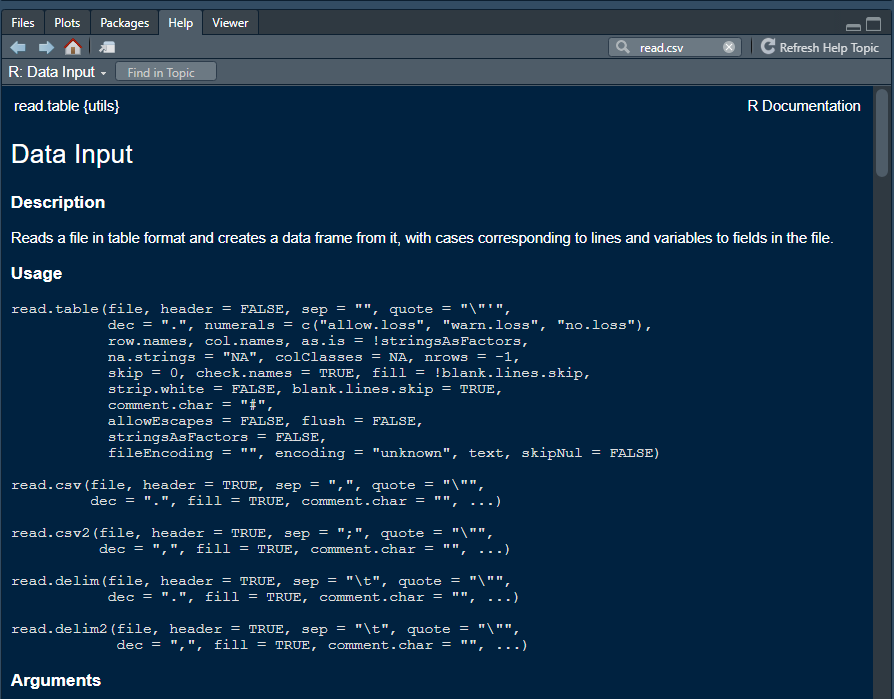
In the example of importing a file, you can see that the function that is used in the import dialogue is called read.csv(). In the import dialogue, we applied quite a few different settings, however, in the line of code visible in the history pane, only one appears (na.string). The reason for this is, that every function has a set of standard values which is used when nothing else is specified. Thus, whenever the standard setting is the one you want to use, you do not need to write code about this particular setting. How does one know what the standard settings are? This is where the “Help” pane in RStudio comes in handy. Find the help pane (usually located in the lower right corner of RStudio) and write the name of the function into the search bar (in this case: read.csv()). Alternatively, you can write ?read.csv in your script or the console and run this line of code or place the cursor where you have written the function in the script and press the f1 key on your keyboard. All three methods will open the same page in the help pane and will work with any function.

The help page of a function can look a bit intimidating at first, but you do not need to understand every single bit of this now. The first thing you notice is that there in fact are many different functions that appear in this help page. The reason is that R functions come in “families”, don’t worry about this, just find the function that you are using at the moment.

The first thing the help page will show you is what kind of arguments a function has and what their standard values are. When you scroll further down, you can read a desacription of every argument. For read.csv we see the following:

read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)

Here we can for example see that the standard value for the “Header” argument is “True”, meaning the first row of the dataset is containing column names. In the import dialogue we said that this was the case since indeed, the first row of the Titanic datset contained column names. Since “True” is the standard value, the “Header” argument did not appear in the line of code that was written earlier on. The samme applies for the “sep” argument (meaning separator”, as we had a file where columns were separated by a comma, which is the standard setting. At the end we can see “…”. This means that other arguments are the same as in the function described earlier in the help page. To find the standard value for the “na.string” argument, we therefore need to look at the first function in this help section (read.table). There we can see that the standard value for “na.strings” is “NA”. Now, if we inspected the Titanic dataset correctly, we see that missing data was described with a period (“.”), hence we changed this setting in the import dialogue. This is the reason why the “na.strings” argument showed up in our line of code earlier.



**5. Data classes**

R operates with different data classes based on the content of a variable. We will introduce the most important ones here.

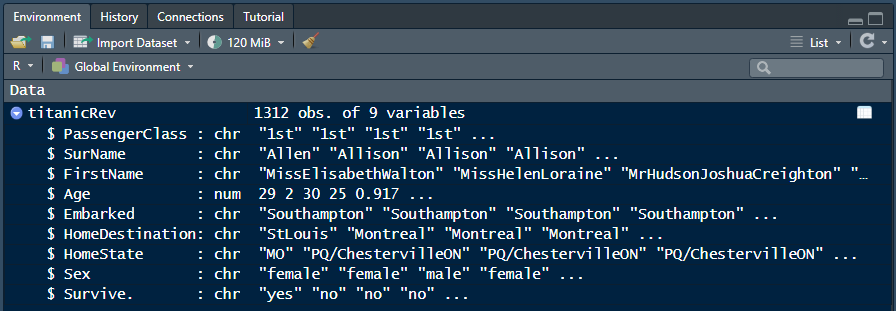
**Character (chr):** Character variables can contain any characters, meaning letters, numbers and other symbols. When numbers are stored inside a character variable, R will not be able to use the numbers as numbers and will only see them as “letters”. This means you cannot use these numbers to calculate things.

**Numeric (num):** Numeric variables contain numbers. Here, R can use the numbers to make calculations. Nothing else can be stored in numeric variables.

**Integer (int):** Integer variables are basically the same as numeric variables, however, they cannot store decimals.

In most cases, R will figure out the correct class by itself, and you don’t need to worry about this. However, there may be cases where you get error messages when doing calculations. In these cases, you may want to check that your variable is numeric or integer and not accidently set to character.

You can easily check the classes of your variables in the environment pane in RStudio. Click on the small blue button next to the dataset to get an overview of all the variables. If everything has gone well, you will see that every variable apart from “Age” has been set to “character” while “Age” correctly has been assigned the “Numeric” class.



*Checking for correct data type in the “Environment” pane*

Let’s assume that “Age” was classed as a “Character” variable. How can we change this to the numeric class?

The general code for this looks like this:

Dataset$Variable <- as.numeric(dataset$Variable)

The dollar sign ($) is always used to choose a specific variable within a dataset. Since we write the same dataset and variable on both sides of the function, the only change that is made, is the change to numeric. If we would like to convert something to characters, we can use the as.character function in the same way.

6. Making a graph of one variable

For example, using the *Titanic* data in "titanicRev.csv", let's plot a histogram of the **numeric** variable “Age” for the passengers.

To plot a histogram we can use the hist() function. Open the help page for the function to get an overview about its different arguments.

The breaks argument lets you change how many bars your histogram has. If you have fewer bars, each bar will contain a larger part of the variation shown on the x-axis. Explore how the histogram changes when you increase or decrease the number of breaks.

Distributions of categorical (character) variables may also be visualized as a bar chart, for example with the variable "*Passenger class*". Making a bar chart in R requires an extra step where we create a simple table (using the table() function) that counts the observations in each category we want to investigate.

Note also how we can use the arguments main, xlab and ylab to edit the title of the graph, the title of the x-axis and of the y-axis respectively. If you do not use these commands, R will put titles based on the dataset, however, quite often making your own titles will make a graph simpler and easier to understand.

7. Making a graph of two variables

You may also plot the relationship between two variables, a numeric and a categorical (character) variable.

First, let’s investigate the age distribution in the different passenger classes of the guests on the Titanic. To do this, we can for example plot three different histograms. Before we can start creating the plots, we need to filter (or subset) the dataset into the different passenger classes (since we want one histogram per passenger class).

In the R language, you use square brackets [ ] to subset a dataset. When referencing rows and columns in R, rows always come first. In this case, we want to reference rows first within the square brackets and the columns afterwards. The basic subsetting syntax looks like this:

Subset <- dataset[rows, columns]

Now, since this looks rather confusing, let’s look at our passenger class example. How do we want our dataset(i.e., the rows and columns) to change when we filter for first class passengers? We want to keep all the rows (=passengers) that are in first class. We also want to keep all the columns in our dataset. We can now write this code:

firstclass <- titanicRev[titanicRev$PassengerClass=="1st",]

This creates a new dataset called “firstclass” based on the subsetting we do. As said, within the brackets, rows are called first. Here we say that we want to keep all the rows where PassengerClass equals “1st”. After the comma, we do not need to write anything since we want to keep all the columns. Note however, that we must write the comma for the subsetting to work! Another important thing to note is the use of two equal signs (==). Using one single equal sign (=) works in the same way as the arrow (<-), meaning it will assign something to an object. Every time we want to say that something is equal to something else, we need to use two equal signs.

Now, with the filtering being finished, we can use the already known hist() function to plot the histograms. If we want to use the histograms to compare the age distributions in the different age classes, we should plot them side by side. We can do this with the par() function and mfrow argument. Inside the mfrow argument we can choose how many rows and columns our plot should have. As you learned when using square brackets to subset, in R rows are named first, then columns. par(mfrow=c(3,1)) gives therefore 3 rows and 1 column, meaning we will get our three histograms underneath each other. If you want them next to each other, you need to write par(mfrow=c(1,3)).

For the best comparison, the axes should be equal in all three plots. As a standard, R choses the length of the axes based on the spread in the data. Since the spread in the three histograms is slightly different, we get different axes. To manually adjust this and force R to show a certain axis, we can use the xlim and ylim arguments within the hist() function (xlim adjusting the x-axis, and ylim adjusting the y-axis).

There are other plots that we can use to explore relationships between a categorical variable (factor) and a numeric variable (response).

Firstly, we can use the package “sciplot” to create a simple line plot (or plot of means) that presents the mean and confidence intervals (mean +/- SE) of one response variable for different factors (groups). The function is called lineplot.CI().

Packages are add-ons that someone has created and are free to use like R itself. We can install a package by writing:  
install.packages(“package”) (i.e., install.packages(“sciplot”))

and then load the package by writing:   
library(packagename) (i.e., library(sciplot))

We can also use the stripchart() function to create a strip chart. Here, we have some different arguments that adjust the plot. You can play around a bit with for example the type argument to explore different ways of making this graph.

In the titanic dataset, we only have one numerical variable. If you want to plot two numeric variables in another dataset, you need to use the plot() function to produce a scatter plot.

8. Using the manuals.

These course notes will only scratch the surface of the possible techniques available in *R*.

Maybe the most important trick with R is to know that you don’t need to know every single function by hard. R has an extremely active online community and if you have a question, you can be sure that someone else has already asked the same question on the internet!

Googling will certainly be your best friend when using R in the future. There are many good web pages, discussion forums (such as stackoverflow.com) and blogs that you will come across when searching online for R questions.

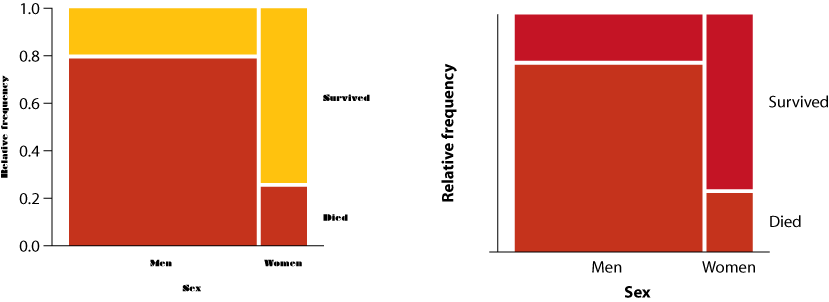
Inside RStudio you will want to learn a few different keyboard shortcuts that will make you life easier. Some of the most important ones have been introduced earlier in this document but you can find a complete list of available shortcuts here: <https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts>. You can also go to “Help” or “Tools” in the menu bar at the top of the RStudio window and click on “Keyboard Shortcuts Help” to get the shortcuts within RStudio. Under the “Tools” menu you can also change the keyboard shortcuts if you prefer other ones than the standard shortcuts.

There is also a really cool R package called swirl. This is an interactive tutorial that teaches the R language directly within RStudio. Go to <https://swirlstats.com/students.html> to find out how to install and start swirl.

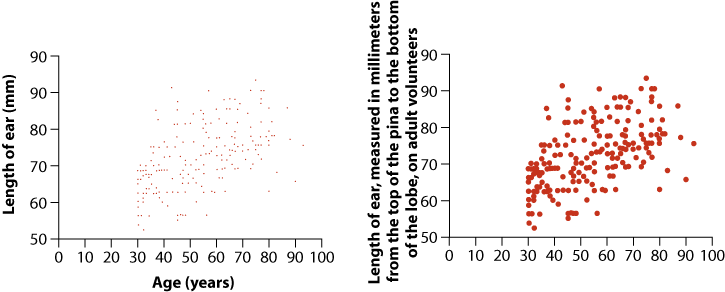
# Questions

**1**. For each of the following pairs of graphs, identify features that are better on one version that the other.

1. Survivorship as a function of sex for passengers of the *RMS Titanic*



1. Ear length in male humans as a function of age



**2**. Open the data file called "countries2005." This file gives various data from the World Bank on all countries of the world for 2005.

1. Plot a histogram for the variable "birth rate crude (per 1000 people)."
2. Look carefully at the histogram. Do you see evidence of a problem with the data file? Find and fix the problem.
3. Plot the histogram on the corrected data set.

**3**. Muchala (2006) measured the length of the tongues of eleven different species of South American bats, as well as the length of their palates (to get an indication of the size of their mouths). All of these bats feed on nectar from flowers, using their tongues. Data from their paper are given in the file "battoungues.csv."

1. Plot a scatter plot with palate length as the explanatory variable and tongue length as the response variable. Describe what you see.
2. All the data points in this graph have been double checked and verified. With that in mind, how do you interpret the outlier on the scatter plot?

**4**. Use the countries data (as corrected as per Question 2). Plot distributions for "Continent," "Prevalence of HIV," and "Physicians per 1000 people.", respectively. What kinds of variables are these (numerical or categorical)? What kinds of graphs are being drawn?

**5**. Use the countries data (as corrected in Question 2). Plot the relationship between the following sets of variables:

1. Male life expectancy and female life expectancy,
2. Continent and life expectancy,
3. Literacy rates and life expectancy
4. Personal computers and life expectancy
5. Number of physicians and life expectancy
6. Which variable seems to explain life expectancy better: number of personal computers or number of physicians? Try to explain the pattern that you see.

**6**. Use the data set collected on your class from today. (We'll return later to some of the other variables later in the term.) Plot the relationship between the following pairs of variables. For each case, describe the pattern that you observe:

1. Handedness and "footedness"
2. Handedness and dominant eye
3. Sex and height
4. Height and head circumference

**7**. (Optional) -- This last exercise will give more exercise and teach some slightly more advanced techniques for using *R*.

Here, we will use the “*MammalsLarge.csv”* dataset that consists of data from a study of body mass of large mammals during the last Ice Ages. These data were published as a data paper in Ecology and deposited in the *Ecological Archives.*

To investigate the “*MammalsLarge*.csv” data set more in detail, you are asked to read and examine the data. The data are stored in a ‘flat’ csv-text file:

**Read and examine the data**

Download the file to your computer, and open it in a spreadsheet program e.g., Excel, Calc. to explore the dataset.

Note the following important things:

1) comma is used as a data separator

2) period is used as a decimal point character

3) NA is used to denote North America. As we learned earlier, NA is commonly used to specify data that is not available, so it is important to change

4) data that is not available is specified by an empty cell -> we need to specify this upon importing the dataset

Import the dataset to RStudio using the correct settings.

Carry out the following inspections of the data:

View the first few lines of the data frame on the screen. You’ll see that every row represents the data for a different mammal species. How many character variables and how many numerical variables do we have?

Use the table() function to explore the distributions of the variables “Continent” and “Status”.

You’ll notice from the graph that there’s a typo in the data for the variable “continent”. One case is shown as being on the continent “Af” rather than “AF”. Fix this.

Create a new variable in the mammal data set: the log (base 10) of body mass. You can either do this directly in R or in Excel and import the new dataset afterwards.

**Plots of single variables**

Produce a bar graph of the number of mammal species on each “continent”. Which continent has the most mammal species? Which has the least?

Generate a histogram of the body mass of mammal species. How informative is that?

Generate a histogram of log body mass. Is this more informative?

Redo the previous histogram but use the Number of bins option to decrease the resolution to 10 units. How much detail is lost? Does it affect the interpretation of the data if you increase resolution to 20? 50? 100?

Redo the histogram but display probability density instead of frequency.

**Plots of two variables**

Use a box plot to compare the distribution of body sizes (log scale most revealing) of mammals having different extinction status. Are extinct mammals similar to, larger than, or smaller than, extant mammals?

Examine the previous box plot. How do the shapes of the body size distributions compare between extinct and extant mammals?

Calculate the median log body mass of each extinction status group of mammals. Check that these are consistent with the box plot results. You can either use the median() function to calculate the median for each group on their own, or the tapply() function to output a handy table that includes every status group.

Use procedure again to calculate the mean of log body mass of each extinction status mammal group. Why is the mean log size of extant mammals larger than the median, but the mean log size for extinct mammals smaller than the median?

Repeat the command for untransformed body mass (rather than log-transformed body mass). What is the magnitude of the difference in median body mass of the different groups?

# Stats lab student data sheet 1

Record the following measurements on yourself. These will be shared (without your name attached) with the rest of the class, so if you feel any discomfort with such sharing feel free to omit any or all of the measurements.

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Height* (cm) |  |  |
|  | *Circumference of head (cm)* | Measure around the head at eye-brow level. |  |
|  | *Sex* |  |  |
|  | *Number of siblings* | How many siblings (not counting yourself) are in your family? |  |
|  | *Handedness* | Which hand do you normally write with? |  |
|  | *Dominant foot* | Which foot do you kick a ball with? |  |
|  | *Dominant eye* | Make a triangle with your index fingers and thumbs, and hold it out at arms' length. With both eyes open, focus on a specific spot in the distance, and center that spot in the triangle of your fingers. Now hold your arms still and shut one eye after the other. The spot will be closer to the center of the triangle when viewed with your dominant eye. | |