PSY9510 - Day 1

2022-09-26

The very beginning

We first started to **set** the working directories (an *absolute path*)

```
setwd("~/OneDrive - University of Bergen/Fag/PROGRAMMERING/R/PSY9510-R")
```

To find out which working directory we´re in:

```
getwd ()
```

```
## [1] "/Users/evgeniataranova/OneDrive - University of Bergen/Fag/PROGRAMMERING/R/PSY9510-R"
```

We then proceeded with installing a library package (such as tidyverse):

```
#install.packages("tidyverse")
```

Each time we open R, we need to open the desired package. This is done in the following way:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.0      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Vectors

We then played with vectors:

```
c(10,20,30)
```

```
## [1] 10 20 30
```

```r
B<- c(10,20,20)                          #Assigned the vector to a variable
length(B)                                #Checking the length of the vector
```

```
## [1] 3
```

Logical operators

We also looked at some logical operators:

```r
TF <- c(TRUE, T, TRUE, FALSE, F)  #Playing with logical operators and vectors
typeof(TF)
```

```
## [1] "logical"
```

```r
length(TF)
```

```
## [1] 5
```

```r
assign("A", 10)
A < 10
```

```
## [1] FALSE
```

```r
TF == 1:5
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE
```

```r
TF == c(1, 1, 1, 0, 0)            #TRUE corresponds to 1, FALSE corresponds to 0
```

```
## [1] TRUE TRUE TRUE TRUE TRUE
```

R markdown

We then proceeded to working with and exploring R markdown.

`#Heading level 1`

or

`Heading level 1 ================`

`##Heading level 2`

or

`Heading level 2 ----------------`

etc.

Creating a list

`* item 1`

`+ sub-item1`

`+ sub-item2`

! Make sure to have a clean line after each list-line

Working with dynamic R tables

R markdown´s tables are a bit difficult, and we want a smarter way to do so. We therefore installed the R package `pander`.

(OBS! We now code directly in markdown via the chunks)

To create a table, we need a dataset. For this example we use the dataset `cars`.

```
library(pander)
data(cars) #Opens the dataset `cars`
#pander(cars) #This makes a table of the dataset - but this was a bit stupid

pander(head(cars)) #Head takes only takes a specified amount of data -
```

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

```
#Tail taks the last part of an object
pander(tail(cars,n=3L)) #L just means an integer, we can simply remove it (i.e. the same as L. It doesn
```

	speed	dist
48	24	93
49	24	120
50	25	85

Creating a dataset in R

Data frame are fundamental data structures used by most of R´s modelling software.

We create a data frame using the `data.frame`-function

First,we can look at what data-frame really is:

```
?data.frame()
```

Subsetting data in R

(i.e. taking a subset of a vector)

We can specify specific rows and specific columns = and thus create a new dataset

Example):

Now we test what LETTER does. All the english letters are defined in R automatically and we do not need to define it ourselves:

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

If we want only some letters, we define which one we want by speicfying the interval:

```
Three_first_letter <- LETTERS [1:3]
print(Three_first_letter)
```

```
## [1] "A" "B" "C"
```

1 dimension: needs only 1 index: `LETTERS[index1]`

2 dimensions: need 2 indexes: `LETTERS[index1, index2]`

If we want only some letters, we define which one we want by speicfying the interval:

```
Three_first_letter <- LETTERS [1:3]
print(Three_first_letter)
```

```
## [1] "A" "B" "C"
```

An example: subset only line 2 and 4, and all the columns:

```
data(iris)
#print(iris) #Iris is a dataset
print(iris[2:4, ])
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
```

select the rows 12:14 and columns 2:5

```
pander(iris[12:14, 2:5])
```

	Sepal.Width	Petal.Length	Petal.Width	Species
12	3.4	1.6	0.2	setosa
13	3	1.4	0.1	setosa
14	3	1.1	0.1	setosa

You can also be very selective in indexing, i.e. not a subset but specific data!

```
pander(iris[c(1,2,3,7,9), c(1,2)])
```

	Sepal.Length	Sepal.Width
1	5.1	3.5
2	4.9	3
3	4.7	3.2
7	4.6	3.4
9	4.4	2.9

```
#The first vector is rows, the second is the columns
```

Check how many dimensions a R object has:

```
dim(iris)
```

```
## [1] 150   5
```

....And now we´ve manipulated a lot with the dataset iris - perhaps it´s time to **clean it**:

```
remove(iris)
```

(A little bir more on lists:)

```
#Creating a list:
list_test <- list(a = 1:3, b= "example_string", c = pi, d= list (1,2,3))
```

Now, let´s examine what this vector list consists of:

1. `list_test` - the name of the vector that holds a list
2. `list()` - the functions to construct a list
3. `a=1:3` - a list called `a` holding the integers `1`, `2`and `3`.
4. `b="example_string"` - a list called `b` holding a string
5. `c = pi` - a list called `c`holding the value of pi
6. `d = list(1,2,3)`- a list called `d` holding a list (i.e. thus the list `list_test` holds a list itself) that holds the integers `1`, `2` and `3`, i.e. the same as list `a`, just "written out".

Creating a data.frame

Let´s now create a dataframe.How to do this is chowed in the following example:

```
name <- c("Emily", "Julie", "Stina", "Eirik", "Nadine", "Sara", "Ole", "Anders", "Nikles", "Fredrik")
age <- c(27,25,31,26, 31, 22, 27,37, 44, 45)
gender <- c("female", "female", "female", "male", "female", "female", "male", "male", "male", "male")

df <- data.frame (name, age, gender) #Defining a dataframe

pander(df) #Using pander to print the df
```

name	age	gender
Emily	27	female
Julie	25	female
Stina	31	female
Eirik	26	male
Nadine	31	female
Sara	22	female
Ole	27	male
Anders	37	male
Nikles	44	male
Fredrik	45	male

Now, we try to subset some data:

Q: What is the mean of age for female participants of the class PSY9510?

A:

We can use the method `mean` to calculate the average

OBS! To get a speicific object type use: `data_frame_name$variable_name`

- The `$` operator is used to extract or subset a specific part of a data object in R. For instance, this can be a data frame object or a list.

```
print(df[df$gender =="female", "age"])
```

```
## [1] 27 25 31 31 22
```

```
mean(df[df$gender =="female", "age"]) #The function for average
```

```
## [1] 27.2
```

Let´s assign the mean to an R object:

```
female_mean <- mean(df[df$gender == "female", "age"])
```

The analysis showed that the mean age of females in this dataset is 27.2.

(Btw, by typing `"r "variable_name""` , we bring R objects inside text) (Such as using {} in formatted strings in Python)

Let´s calculate the SD for male´s age:

```r
#help(sd)
sd(df[df$gender == "male", "age"], na.rm = TRUE)
```

```
## [1] 9.038805
```

BTW: We add an argument by placing a **,**

BTW: We **remove missing values/data by: `na.rm=TRUE` or `na.remove = TRUE`**

How can I evaluate if the age of the participants are above or equal to 35 or not?:

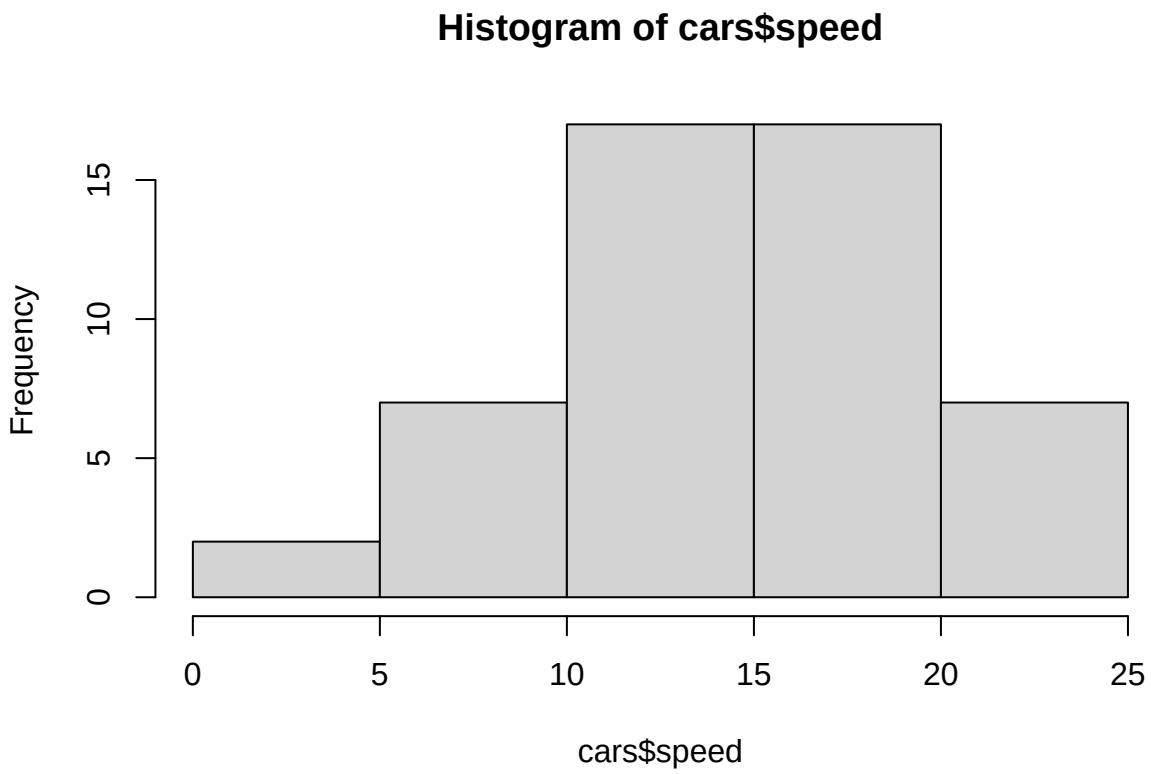
```r
val <- df$age >= 35

#Now I define a new variable

df$age35 <-val

pander(df) #Print the data taht we now have
```

name	age	gender	age35
Emily	27	female	FALSE
Julie	25	female	FALSE
Stina	31	female	FALSE
Eirik	26	male	FALSE
Nadine	31	female	FALSE
Sara	22	female	FALSE
Ole	27	male	FALSE
Anders	37	male	TRUE
Nikles	44	male	TRUE
Fredrik	45	male	TRUE

Adding figures

We can add figures to our markdown. Here we will add a histogram using the `hist`function.

```r
hist (cars$speed)
```


Histogram of cars$speed
Frequency
0
5
10
15
0
5
10
15
20
25
cars$speed

```
#?hist
```