PSY9510 - Day 2

2022-09-27

Setting our working directory as an **absolute path**:

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

(See that this is a *absolute path* - as it is stored in your, and only your, home directory. When working with R (especially data analysis) preferably use a *relative path* in order to be able to share with others)

`~` - a convenient shortcut to your homedirectory (in your case, a shortcut to "OneDrive..")

`"."` - specifying a *relative path*

Setting our working directory in a **relative path**:

- NB! Could be a good idea to install the `here`-package for relative paths! Read about this package here: https://cran.r-project.org/web/packages/here/here.pdf

```
#setwd("./"PSY9510-R)
```

Repetition

To see which working directory we´re in, use `getwd()` To set a working directory, use `setwd(absolute OR relative path)`

- A relative path is noted by `"."` before

For loops

For loops are written in the following form:

```
for (b in 1:5){
  print (b)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Something more complicated - subsetting of a vector (a variable) and use that subsetted value in a for loop.

Two methods:

1)

```
S <- c("some", "character", "for", "the", "example")

for (indiancurry in S) {
  print (indiancurry)
}
```

```
## [1] "some"
## [1] "character"
## [1] "for"
## [1] "the"
## [1] "example"
```

2) This is more complicated, with the use of indexing `variable_name( [index of the element] )`

```
for (b in 1:5){
  print (S[b])
}
```

```
## [1] "some"
## [1] "character"
## [1] "for"
## [1] "the"
## [1] "example"
```

Cleaning our workspace.

First, check what´s in your workspace by (display the workspace - the name of most of objects currently stored within R):

```
ls ()
```

```
## [1] "b"           "indiancurry" "S"
```

```
#or
objects ()
```

```
## [1] "b"           "indiancurry" "S"
```

Cleaning our workspace:

```
#rm("a", "b", "indiancurry") #Remove particular objects
```

Working with Data (part 1)

Repetition:

Data frames - matrix-like structures, in which the columns can be of different types. Think of data frames as "data matrices" with *one row per observational unit but with (possibly) both numerical and categorical values*

- Many experiments are best described by data frames since *treatments are categorical but the response is numeric*

A dataframe has n dimensions (if a table, 2 dimensions - columns and rows)

To index: (e.g. index 1 is column, index2 is row)

```
#df [index1, index2]
```

!! Important to distinguish between `df[[...]]`and `df[...]`.

- `[[...]]`is used for *subsetting*

- `[...]`is used for *indexing*

Typing data in R

- Define data using the `c`function.
- `readRDS` and `saveRDS` are both functions for saving the dataset. Look at help documentation for what arguments this function takes in
  - Can only save one file at a time

```
data(iris) #Loads the dataset "iris"
saveRDS(iris, "testdata.rds") #Saving the dataset "iris" in the file "testdata.rds"
```

.csv files in R

About .csv-files

Every row of the dataset ends with a new line

Delimiters in .csv:

- ;

- :

- ,

” ” in a .csv-file is a NA (missing observation)

read.csv- function

From the “R for Data Science”- book:

- read_csv() reads comma delimited files, read_csv2() reads semicolon separated files (common in countries where , is used as the decimal place), read_tsv() reads tab delimited files, and read_delim() reads in files with any delimiter.

See help-file for `read.csv`

Takes in the following arguments: `read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)`

Where `header` set to `TRUE` indicates the file contains the names of the variables as its first line.

- If missing (if we don not specify it), the value is determnined form the file format:
    - `header` is set to `TRUE` IF AND ONLY IF the first row contains one fewer field than the number of columns

You can get the data directly from the internet by placing the url.

writing/creating a .csv-file in R

Reading the dataset "iris" in to a .csv-file

```
data("iris") #Lading the dataset iris
write.csv(iris, file = "iris_test.csv") #Saves the dataset iris in the file iris_test.csv in the curren
```

Deleting a file or directory:

```
unlink ("iris_test.csv") #Deletes the
```

Reading a .fwf

From the "R for Data Science"-book:

- read_fwf() reads fixed width files. You can specify fields either by their widths with fwf_widths() or their position with fwf_positions(). read_table() reads a common variation of fixed width files where columns are separated by white space.

Reading from internet

To do so, we first create an .rds file

Exercise Loading a dataset and viewing it:

```
data (airquality)
#View(airquality) #Views the data in a separate vindow
dim(airquality) #Tells how many dimensions the dataset has
```

```
## [1] 153   6
```

```
str(airquality) #Tells the structure of the data (it´s characteristics, e.g. dataframe, n of observatio
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Subsetting some data:

*Using the function subset()

From the help-file:

```
subset(airquality, Temp > 80, select = c(Ozone, Temp)) #Specifies which temperature to whow and to only
```

```
##     Ozone Temp
## 29     45   81
## 35     NA   84
## 36     NA   85
## 38     29   82
## 39     NA   87
## 40     71   90
## 41     39   87
## 42     NA   93
## 43     NA   92
## 44     23   82
## 61     NA   83
## 62    135   84
## 63     49   85
## 64     32   81
## 65     NA   84
## 66     64   83
## 67     40   83
## 68     77   88
## 69     97   92
## 70     97   92
## 71     85   89
## 72     NA   82
## 74     27   81
## 75     NA   91
## 77     48   81
## 78     35   82
## 79     61   84
## 80     79   87
## 81     63   85
## 83     NA   81
## 84     NA   82
## 85     80   86
## 86    108   85
## 87     20   82
## 88     52   86
## 89     82   88
## 90     50   86
## 91     64   83
## 92     59   81
## 93     39   81
## 94      9   81
## 95     16   82
## 96     78   86
## 97     35   85
## 98     66   87
## 99    122   89
## 100    89   90
## 101   110   90
## 102    NA   92
## 103    NA   86
## 104    44   86
## 105    28   82
## 117   168   81
## 118    73   86
## 119    NA   88
## 120    76   97
## 121   118   94
## 122    84   96
## 123    85   94
## 124    96   91
## 125    78   92
## 126    73   93
## 127    91   93
## 128    47   87
## 129    32   84
## 134    44   81
## 143    16   82
## 146    36   81
```

To **filter (drop)** away a column: use a `-` as a prefix before the specified column/row

```
subset(airquality, Day == 1, select = -Temp) #Views only the data from Day 1 and shows every column of
```

```
##     Ozone Solar.R Wind Month Day
## 1      41     190  7.4     5   1
## 32     NA     286  8.6     6   1
## 62    135     269  4.1     7   1
## 93     39      83  6.9     8   1
## 124    96     167  6.9     9   1
```

To filter/drop away several columns, place them together in a vector (`-c(column_name, column_name)`)

```
subset(airquality, Day == 1, select = -c(Temp, Wind))
```

```
##     Ozone Solar.R Month Day
## 1      41     190     5   1
## 32     NA     286     6   1
## 62    135     269     7   1
## 93     39      83     8   1
## 124    96     167     9   1
```

Reading Stata data

```
#install.packages("readstata13") #Installing a package for reading STATA-data (readstata13 is recommend
#install.packages("foreign") #Other package
#install.packages("haven") #Other package

library(readstata13) #Opening the recommended package readstata13
```

Reading and writing SPSS datasets in R

Important:

- **BE CAREFUL WITH VARIABLES THAT HAVE LABELS (string/character labels)**. They might be converted to *factor (nominal/categorical) variables* , even if they are stored as numerics on SPSS. I.e. On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.
  - Said in another way:
    - Default option of R: **convert strings to factors** (i.e. character value/variable labels in orirginal dataset → categorical data (gender, color, types) in R).
    - Implication: if your data is NOT categorical → set `stringsAsFactor = FALSE` when importing your data.
  - **An R factor** - the data objects used to categorize the data (i.e. a categorical variable (either integers/numerics or scharacters/strings). The factors - the categorized data - is stored as *levels* (i.e. categories; always a character/string)
    - **Summarized**:
      - A factor (data object for categorization) has *levels* as attributes (a *level* is always a *character/string*)
      - Not possible to do arithmetics on factors -> need to convert back to numeric values
    - The `use.value.labels = TRUE` converts non-categorical valuables in SPSS to R (categorical) factors
    - `use.value.labels = FALSE` does not

Check whether a vector is a factor:

```
str(airquality$Temp)
```

```
##  int [1:153] 67 72 74 62 56 66 65 59 61 69 ...
```

```
is.factor(airquality$Temp) #Cheks whether the variable/column Temp in the data airquality is a factor
```

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

Create a factor - the `factor (data_name)` -function

```
data_1 <- c (1,2,3,3,4,5,6,3,5,2)
factor_of_data_1 <- factor(data_1)
print(factor_of_data_1)
```

```
##  [1] 1 2 3 3 4 5 6 3 5 2
## Levels: 1 2 3 4 5 6
```

Label or index the levels:

```
levels(factor_of_data_1) <- c("One", "Two", "Three", "Four", "Five", "Six")

print(factor_of_data_1)
```

```
##  [1] One   Two   Three Three Four  Five  Six   Three Five  Two  
## Levels: One Two Three Four Five Six
```

See that both the factors and the levels change in accordance!

Some checking of the factors:

```
#Class and mode
class(factor_of_data_1)
```

```
## [1] "factor"
```

```
mode(factor_of_data_1)
```

```
## [1] "numeric"
```

Converting factors: Needed if to do arithmetic operations (bc can´t do arithmetics on factors)

Convert factors back to numeric or character in 2 methods - **Only if the factor has numeric values, not possible if it has character values** :

1. by applying the `as.numeric()`-function on the *level* (and not the factors, because if applied to the factor it returns only how R stores the varibales)
2. by converting the factors to character variable and then converting the character variable to numeric variable: `as.numeric(as.chararcter(data_name))`

1)

```
data_2 <- c(100,200,300,400,400,100, 200)
data_2_factor <- factor(data_2)

#See what happens if we use character labels:
levels(data_2_factor) <- c("Hundred", "Two hundred", "3", "4") #Assigned character and numeric labels t

#print(data_2_factor)
#Hundred     Two hundred 300         400         400         Hundred     Two hundred
#Levels: Hundred Two hundred 300 400

as.numeric(levels(data_2_factor)) #Trying to convert all the values back to numeric
```

```
## Warning: NAs introduced by coercion

## [1] NA NA  3  4
```

```
#Warning: NAs introduced by coercion[1]  NA  NA 300 400

#The error message is because it is not possible to convert character labels ("Hundred" and "Two hundre
```

2)

```
#2)

as.numeric(as.character(factor_of_data_1))
```

```
## Warning: NAs introduced by coercion

##  [1] NA NA NA NA NA NA NA NA NA NA
```

To read SPSS data, load the `foreign`-package

- `read.spss` from this package reads a file stored by the SPSS `save` or `export` commands.
- See the help-file for `read.spss` how you want to handle the data in SPSS format in exporting it to R

```
#install.packages("foreign") #A package
#install.packages("haven") #A package
#library(foreign) #Installing the "foreign"package that has the relevant functions
#?read.spss
#read.spss (file_name, use.value.labels = FALSE, ....)
```

See that setting `use.value.labels` to `FALSE` does not convert the SPSS‘ value labels to R´s special factors.

Write a SPSS file from an R object

```
data("airquality")
haven::write_sav(data=airquality, path="airquality.sav") #Here, we only load the subpackage "write_sav"
```

Read a SPSS file into R

```
df <- haven::read_sav(file = "airquality.sav")  #Here, we only load the subpackage "read_sav" from the
str(df)
```

```
## tibble [153 x 6] (S3: tbl_df/tbl/data.frame)
##  $ Ozone  : num [1:153] 41 36 12 18 NA 28 23 19 8 NA ...
##   ..- attr(*, "format.spss")= chr "F8.0"
##  $ Solar.R: num [1:153] 190 118 149 313 NA NA 299 99 19 194 ...
##   ..- attr(*, "format.spss")= chr "F8.0"
##  $ Wind   : num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##   ..- attr(*, "format.spss")= chr "F8.2"
##  $ Temp   : num [1:153] 67 72 74 62 56 66 65 59 61 69 ...
##   ..- attr(*, "format.spss")= chr "F8.0"
##  $ Month  : num [1:153] 5 5 5 5 5 5 5 5 5 5 ...
##   ..- attr(*, "format.spss")= chr "F8.0"
##  $ Day    : num [1:153] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "format.spss")= chr "F8.0"
```

```
library(pander)
pander(head(df))
```

Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

Playing with some data:

```
data(attenu)
#Viev(attenu)
colnames(attenu)
```

```
## [1] "event"   "mag"     "station" "dist"    "accel"
```

```
vec <- colnames(attenu)
length(vec)
```

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

```
vec[3]
```

```
## [1] "station"
```

```
vec2 <- vec[3:1] #Subsets the columna From 3 to 1, 3 and 1 included
```

Summary descriptive statistics:

```
summary(attenu, digits=4) #Using the summary-funciton for SUMMARY STATISTICS. "digits = 4" indicates on
```

```
##      event            mag          station         dist       
##  Min.   : 1.00   Min.   :5.000   117    :  5   Min.   :  0.50  
##  1st Qu.: 9.00   1st Qu.:5.300   1028   :  4   1st Qu.: 11.32  
##  Median :18.00   Median :6.100   113    :  4   Median : 23.40  
##  Mean   :14.74   Mean   :6.084   112    :  3   Mean   : 45.60  
##  3rd Qu.:20.00   3rd Qu.:6.600   135    :  3   3rd Qu.: 47.55  
##  Max.   :23.00   Max.   :7.700   (Other):147   Max.   :370.00  
##                                  NA’s   : 16                   
##      accel        
##  Min.   :0.00300  
##  1st Qu.:0.04425  
##  Median :0.11300  
##  Mean   :0.15422  
##  3rd Qu.:0.21925  
##  Max.   :0.81000  
## 
```

subset the missing functions: In the example above, R tells us that there are 16 missing values (NA=16). We want to check if this is correct.

`is.na` checks whether an observation is NA or no (the output is TRUE or FALSE)

```
is.na(attenu$station) #checks whether an observation is NA or no (the output is TRUE or FALSE)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
##  [97] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [109] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
## [121] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [157] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [181] FALSE FALSE
```

```
length(is.na(attenu$station)) #Checks the amount of factors (elements, in R-language "the length of vec
```

```
## [1] 182
```

Check how many missing values:

`sum(is.na(attenu$station))` counts how many missing values we have.

```r
length(is.na(attenu$station)) #Checks the amount of factors (elements, in R-language "the length of vec
```

```
## [1] 182
```

Reading Excel-files

The `readxl` package makes it easy to get data out of Excel into R

```r
#install.packages("readxl")
library(readxl)
```

If - statements

Exercise:

Check whether the variable "event" in the dataset "attenu" is of the type "numeric":

```r
data(attenu)

if (is.numeric(attenu$event)){
  mean(attenu$event, na.rm=FALSE) #na.rm=FALSE drops the missing values
} else {

}
```

```
## [1] 14.74176
```

Another example:

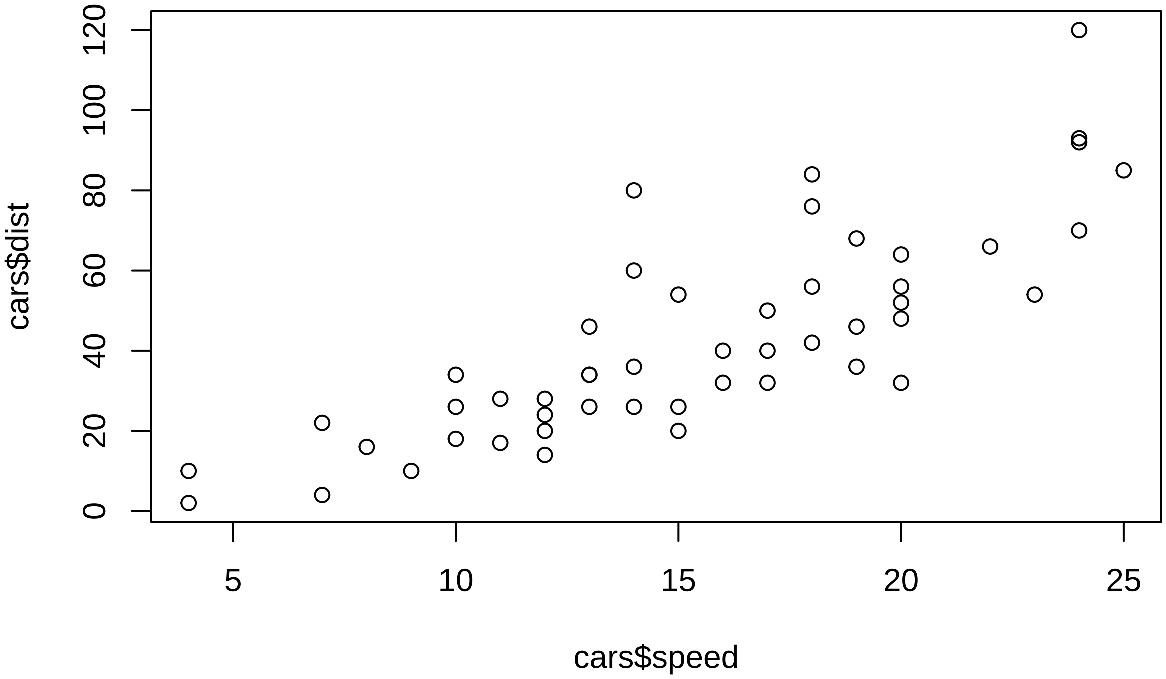
```r
data(iris)

if (is.numeric(iris$Species)){
  mean(iris$Species)
} else {
  str(iris$Species)
}
```

```
##  Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Basic plotting (using R base - without tidyverse)

One can plot using R base with the help of the `plot`- function.

```r
data(cars)
#View(cars)
plot(cars$speed, cars$dist) #Creates a plot where speed is on the x-axis and distance on the y-axis
```



Working with character vectors (strings)

Important printing functions:

- `print()`
- `paste()`
  - concatenates vectors after converting to character
  - read help-file for input arguments
- `cat()`

Manipulating/working with strings/text data:

- Using R base
- Using `stringr`package

Using R base

The function `substr()` extracts or replaces substrings in a character vector:

```
food <- "tofu"
nchar (food) #Number of characters in this vector of one length (NOT `length()`as this is for a string)
```

```
## [1] 4
```

```
substr("cooked tofu", start=1, stop=2) #Takes the first two (start =1, stop =2) letters.
```

```
## [1] "co"
```

To search for a word using *R base*:

1. Split the string based on something common, e.g. a space, in order to treat each word of a text as a separate element of a string vector

```
split_food <- "cooked tofu"

strsplit (split_food, split = " ")
```

```
## [[1]]
## [1] "cooked" "tofu"
```

2. By using the `which`-function, gives the `TRUE` indiced of a logical object. Example:

```
LETTERS #Printing all the letters in the English alphabet
```

```
##  [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"
```

```
which(LETTERS == "R") #Asks which index does the letter "R" pose?
```

```
## [1] 18
```

To search for a word using *stringr -packages* :

(Stringr is part of tidyverse)

Has 7 main verbs:

- `str_detetct`
- `str_count`
- `str_subset`
- `str_locate`
- `str_extract`
- `str_match`
- `str_replace`
- `str_c` (concatenates the elements of a vector if those are of string-type)

(MERK! Tidyverse´s metagrammar is `_`)

```
library(stringr)

x <- c("Why", "Video", "Cross")

str_length(x)
```

How to do nchar(), concatenate, and substring() with stringr:

```
## [1] 3 5 5
```

```
str_c(x)
```

```
## [1] "Why"   "Video" "Cross"
```

!!

Check out `regexxplain`- package (RegEx-library) - a good package to practise expressions of the `stringr`-package