Practical 1 - Processing data

Diego Aguilar-Ramirez

2022-05-12

Contents

In this practical, you will learn how to import your data into R. You will also learn how to explore your data and understand what type of variables are within the data set. Then, you wil use some basic tools for visualizing the distribution of variables. Finally, you'll be introduced to basic ways for transforming variables in R. A brief description of how to select variables is included at the end of this practical.

DISCLAIMER This teaching material is still in development. Please contact the author to report mistakes and for general feedback at diego.aguilar-ramirez@ndph.ox.ac.uk

Set up

STEPS training data set

For this practical, we will be using the data set `WHO-STEPS-training.csv`.

`WHO-STEPS-training.csv` is a modified version of the data set used for training on the WHO-developed software **Epi Info**. It contains data from 1289 participants of the STEPS survey, which collects data on key risk factors for non-communicable diseases.

`WHO-STEPS-training.csv` includes the following variables:

NEED TO COMPLETE TABLE

Name	Description

Please save this data set in your computer's **Desktop**, in a folder called **data**.

The path to this file should look like this: `C:/User/Desktop/data/WHO-STEPS-training.csv`

Prepare your script for the practical

In the previous practical (pre-course material), you created an **R Project** and an **R script**.

As mentioned in the lecture, it is important to keep track of all the data cleaning steps. The best way to do this is using a script, so let's create a new one for this practical.

Open **RStudio** if it is not already open.

Create a new R Script (you can use the short-cut `Ctrl`+`Shift`+`N`)

Or go to File > New File > R Script

Annotate the script:

```
# Practical 1: Processing data
# Author:
# Date:
```

Now save your script with a descriptive file-name. For example: `01.prac_data-proc.R`

Help in R

R includes extensive facilities for accessing documentation and searching for help. The `help()` function and `?` help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages. Also, an HTML-based global help can be called with `help.start()`.

Cheat sheets are a great resource for learners. RStudio as a bunch available and you can find them here. This one covers basic R functions and this one is helpful to get your head around the RStudio IDE.

> Use help documentation regularly!

---

1. Importing data

In this section, you will learn how to import your data set into R within an R Project

For this practical, we will place the training data set within the project directory in a folder called `/data`. However it is also acceptable to import the data set directly from its original location. We will explore both options.

Before importing a data set, it is a good idea to have a clear expectation of what is included in the data. Usually, data sets come with data dictonaries that describe the contents of the data. You want to make sure you have a clear idea of how many variables (columns) your data should have and a general idea of how many individuals or observations (rows) the data should have.

Be aware that extremely large data sets (thousands to millions for rows or columns) might need to be approached more carefully than shown here and could require large computing resources.

NOTE: The decision to create copies of data sets and move these data into other directories largely depend on the standards of information governance and data security within institutions.

1.1 Manually move the data into project directory and then import into R

1.1.1 Move dataset

1. Open your R Project directory `C:/path/to/file/2022_Intro-to-statistics` in your File Explorer
2. Create a folder called `data`
3. Save a new copy of the `WHO-STEPS-training.csv` data set in the newly-created folder `data`

1.1.2 Import to R

For this step, we are going to use two : ***rio*** and ***here***. These packages facilitate defining relative file paths and importing (and exporting) data. Additionally, they work nicely with RStudio projects.

- **rio** functions `import()` and `export()` handle many different file types (.csv, .tsv, .xlsx)
- **here** function `here()` allows description of location of files in a given R project in relation to the project's *root directory*

Let's see what they do:

Calling the function `here()` returns (to the **Console**) the file path to your current **working directory**.

```
pacman::p_load(here) #Install and load the package here() using pacman
```

```
here()
```

```
## [1] "J:/01_Teaching/Short-courses/2022_Intro-to-statistics/scripts"
```

Adding the arguments `"data"` and `WHO-STEPS-training.csv` return the file name of the data set.

```
here("data","WHO-STEPS-training.csv")
```

```
## [1] "J:/01_Teaching/Short-courses/2022_Intro-to-statistics/scripts/data/WHO-STEPS-training.csv"
```

Now let's use the function `import()` to open the data set in the filepath returned by the `here()` function and assign it to the object `steps.data`:

```
pacman::p_load(rio)
steps.data <- import(here("data", "WHO-STEPS-training.csv"), na="") #In this data, missing values are i
```

You are probably asking what is an **object**.

Well, when instructed to, R treats any datum or data that we want to temporarily hold on to (for instance, to use for analyses) as an **object**.

The way to instruct R to do this is with the assignment operator `<-` and in the form:

```
x <- y
```

where `x` is the object being assigned whatever value `y` is. `y` can be many things including a number, a string, a vector (of numbers or strings or both), the result returned by a function, a matrix, or a table).

In this case, we are asking R to read the data set `WHO-STEPS-training.csv` and assign it to an object called `steps.data` which will "live" in the R workspace (i.e. temporarily until the R session ends).

Now that you have imported the data into R, you should be able to see it on the Environment pane (top right). It should show that `steps.data` has 1289 obs. of 26 variables.

1.2 Do it all with R

1.2.1 Import directly from original location

You can also use the absolute path to import the data (not using `here()`)

```
steps.data <- import("C:/User/Desktop/data/WHO-STEPS-training.csv", na="")
```

Notice that by importing the data set this was, the data is only temporarily loaded into R, but no copy is placed in your R Project directory.

NOTE: see the **advanced** section to learn how to create a new directory with R and then export a your data set

> Now were are ready to explore our data!

---

2. Understanding & Exploring data

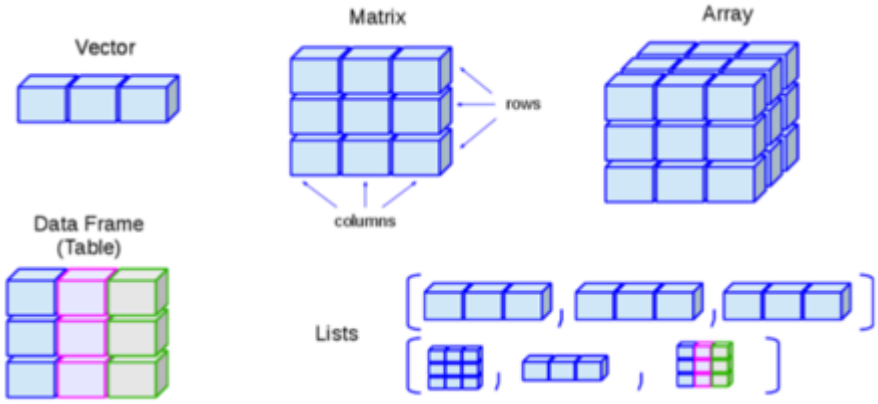
2.1 How does R understand data

Often, epidemiological data comes from **multiple sources**:

- Questionnaire data from recruitment
- Biological sample results
- Linkage to clinical/mortality records

How do data usually look like in `R`?

`R` is an object-oriented language where **objects** are anything (constants, data structures, functions, graphs) that can be assigned to a variable



As described in the Epidemiologist R Handbook, in epidemiology, data sets are usually in the format of a **data frame**, where rows are individuals and columns are variables with different measurements for each individual. This is the data structure that will be most relevant for this course insofar as it most simply depicts **"the data"**.

However, **vectors** are also commonly used throughout R coding and most of the functions used for statistical models output their results as **lists**. Therefore, mastering the use of other data formats besides the **dataframe** is important in order to take full advantage of R.

2.2 Exploring the data

There are multiple commands or functions in R that are useful to have a glimpse at your data. Use combinations of the following functions to understand the contents of the data set.

Use `dim()` to obtain the dimensions of the data frame (number of rows and number of columns). The output is a vector.

```
dim(steps.data)
```

```
## [1] 1289   12
```

Use `nrow()` and `ncol()` to get the number of rows and number of columns, respectively. You can get the same information by extracting the first and second element of the output vector from `dim()`.

```
nrow(steps.data)
```

```
## [1] 1289
```

```
ncol(steps.data)
```

```
## [1] 12
```

Notice how

```
dim(steps.data)[1]
```

```
## [1] 1289
```

```
# same as
nrow(steps.data)
```

```
## [1] 1289
```

And how

```
dim(steps.data)[2]
```

```
## [1] 12
```

```
# same as
ncol(steps.data)
```

```
## [1] 12
```

Use `head()` to obtain the first n observations and `tail()` to obtain the last n observations; by default, $n = 6$. These are good commands for obtaining an intuitive idea of what the data look like without revealing the entire data set, which could have millions of rows and thousands of columns.

```
head(steps.data,n=5)
```

```
tail(steps.data,n=5)
```

The `str()` function returns many useful pieces of information, including the above useful outputs and the types of data for each column.

```
str(steps.data)
```

```
## 'data.frame':    1289 obs. of  12 variables:
##  $ gender      : chr  "Women" "Men" "Men" "Men" ...
##  $ age         : int  39 43 37 60 43 32 61 62 42 56 ...
##  $ curr_smkr   : int  2 2 1 2 2 1 2 2 1 2 ...
##  $ daily_smkr  : int  NA NA 1 NA NA 1 NA NA 1 NA ...
##  $ curr_alc    : int  NA 1 1 NA NA NA 1 1 NA 1 ...
##  $ freq_alc    : int  NA 2 2 NA NA NA 3 3 NA 1 ...
##  $ bp_last_meas: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ height_cm   : int  162 169 168 179 159 171 168 175 175 172 ...
##  $ weight_kg   : int  72 77 73 96 80 96 77 77 94 83 ...
##  $ sbp_1       : int  136 123 117 150 120 136 165 141 159 134 ...
##  $ sbp_2       : int  147 118 129 147 104 125 158 138 139 144 ...
##  $ sbp_3       : int  133 117 116 138 110 129 153 146 138 136 ...
```

In this example, `num` denotes that the variable is numeric (continuous) and `chr` that it is a string character.

2.3 Summarising data

Summarising data can help draw out patterns.

2.3.1 Generate basic data descriptive statistics

When applied to a data frame, the `summary()` function returns summary statistics for all columns, according to the class of each column.

```
summary(steps.data)
```

```
##     gender               age          curr_smkr       daily_smkr
##  Length:1289        Min.   : 0.00   Min.   :1.000   Min.   :1.000
##  Class :character   1st Qu.:35.00   1st Qu.:1.000   1st Qu.:1.000
##  Mode  :character   Median :44.00   Median :2.000   Median :1.000
##                     Mean   :44.28   Mean   :1.616   Mean   :1.246
##                     3rd Qu.:54.00   3rd Qu.:2.000   3rd Qu.:1.000
##                     Max.   :65.00   Max.   :2.000   Max.   :2.000
##                     NA's   :6       NA's   :6       NA's   :801
##     curr_alc        freq_alc      bp_last_meas     height_cm
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   : 78.0
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:163.0
##  Median :1.000   Median :2.000   Median :1.000   Median :169.0
##  Mean   :1.161   Mean   :2.578   Mean   :1.272   Mean   :168.8
```

```
##  3rd Qu.:1.000   3rd Qu.:3.000   3rd Qu.:1.000   3rd Qu.:175.0
##  Max.   :2.000   Max.   :4.000   Max.   :3.000   Max.   :201.0
##  NA's   :355     NA's   :505     NA's   :9       NA's   :88
##    weight_kg          sbp_1           sbp_2           sbp_3
##  Min.   : 39.00   Min.   : 58.0   Min.   : 89.0   Min.   : 88.0
##  1st Qu.: 78.00   1st Qu.:124.0   1st Qu.:119.0   1st Qu.:118.0
##  Median : 91.00   Median :137.0   Median :132.0   Median :129.0
##  Mean   : 93.74   Mean   :139.6   Mean   :133.9   Mean   :132.4
##  3rd Qu.:107.00   3rd Qu.:153.0   3rd Qu.:146.0   3rd Qu.:144.0
##  Max.   :232.00   Max.   :237.0   Max.   :226.0   Max.   :227.0
##  NA's   :88       NA's   :89      NA's   :89      NA's   :89
```

You can also summarise a single variable using

```
summary(steps.data$sbp_1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    58.0   124.0   137.0   139.6   153.0   237.0      89
```

The package `skimr` is also useful for summarising data. It's function `skim()` provides a wealth of data set information and summary statistics. It also provides relative and absolute measures of missingness.

```
pacman::p_load(skimr)
```

```
skim(steps.data)
```

Table 2: Data summary

Name	steps.data
Number of rows	1289
Number of columns	12
Column type frequency:	
character	1
numeric	11
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
gender	0	1	0	5	1	3	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
age	6	1.00	44.28	11.23	0	35	44	54	65	
curr_smkr	6	1.00	1.62	0.49	1	1	2	2	2	
daily_smkr	801	0.38	1.25	0.43	1	1	1	1	2	
curr_alc	355	0.72	1.16	0.37	1	1	1	1	2	
freq_alc	505	0.61	2.58	0.93	1	2	2	3	4	
bp_last_meas	9	0.99	1.27	0.54	1	1	1	1	3	
height_cm	88	0.93	168.81	8.88	78	163	169	175	201	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
weight_kg	88	0.93	93.74	22.28	39	78	91	107	232	
sbp_1	89	0.93	139.65	22.78	58	124	137	153	237	
sbp_2	89	0.93	133.85	21.12	89	119	132	146	226	
sbp_3	89	0.93	132.40	20.60	88	118	129	144	227	

Question 2.3.1 Complete the table bellow.
HINT: use the functions `mean()`, `sd()`, and `quantile()`

	sbp_1	age	height_cm	weight_kg
mean				
median				
SD				
range				
NA's				

TIP: If a variable has missing values `NA`, functions such as `mean()` or `sd()` will yield missing values (The result will show `NA`). The option `na.rm` indicates whether `NA` values should be stripped before the calculation proceeds.

```
mean(steps.data$height_cm)
```

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

```
mean(steps.data$height_cm, na.rm=TRUE) # na.rm removes missings
```

```
## [1] 168.8093
```

Result to question 2.3.1

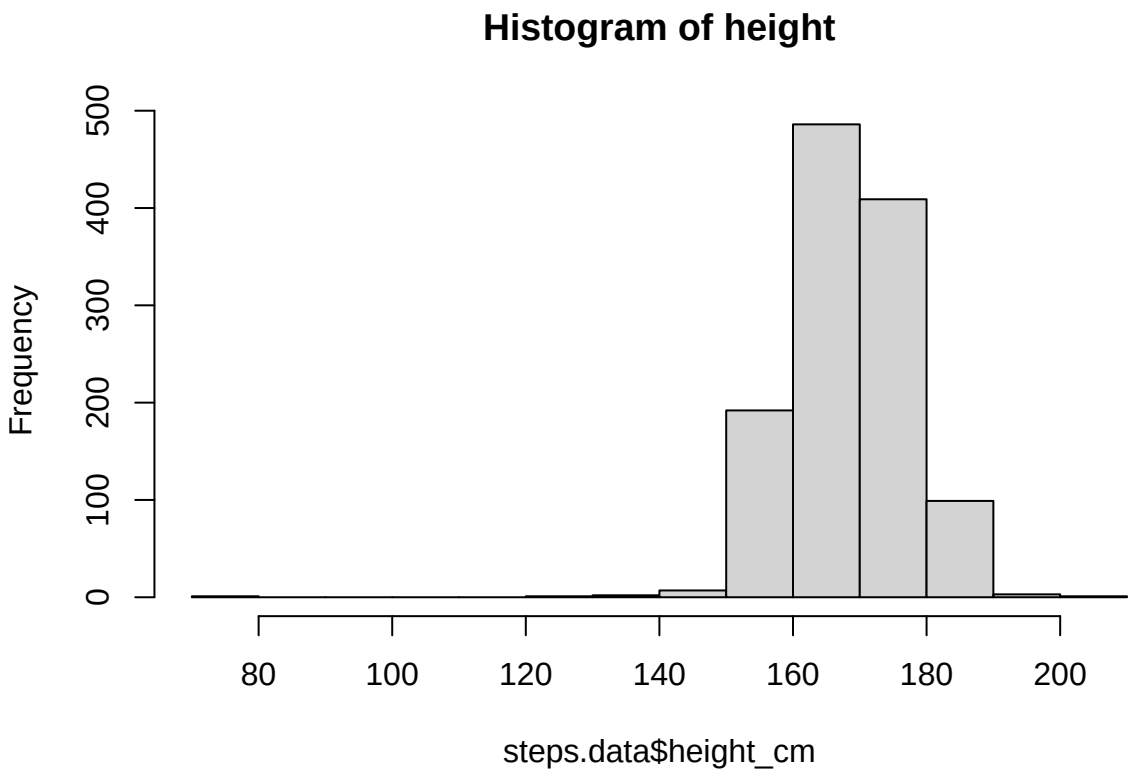
NOTE -> NEED TO ADD POPULATED TABLE AS ANSWER TO QUESTION 2.3.1

2.4 Distributions and outliers

Question 2.3.1 Plot a histogram of `height_cm` and describe the distribution.

HINT: look at the histrogram but also check the mean and median

```
hist(steps.data$height_cm, main="Histogram of height")
```



If you are unsure on skewness, remember to use the summary statistics.

Rules of thumb:

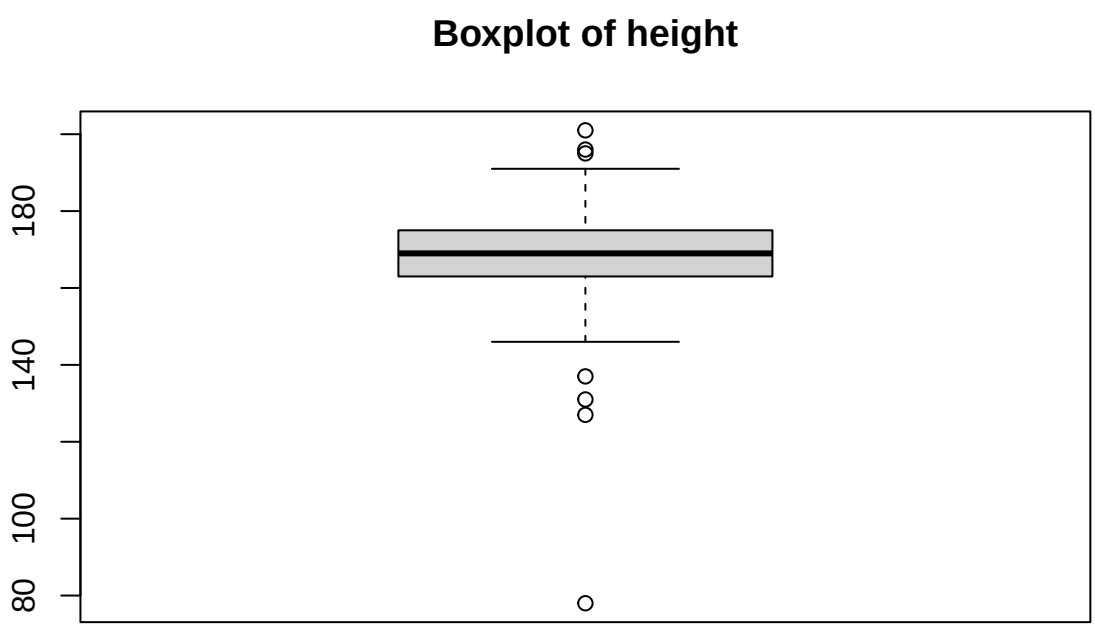
- If median = mean, it is usually normally distributed
- If data are right-skewed, mean is bigger than median
- If data are left-skewed, mean is smaller than median

In a histogram, the height of the bars on the vertical axis needs to be proportional to the probability distribution of that continuous variable (this depicts the parts of the range that have more data points). If the class (i.e. bins) widths are equal, then a frequency histogram can be used. However, if the class widths are unequal (for example, if height is truncated at 200+), then the height of the bars in a frequency histogram will misrepresent the probability distribution. With non-uniform bin size, histograms constructed with densities help maintain an accurate visualisation of the data distribution.

An important step of data exploration is to identify **outliers**, which are observations that are substantially outside the range of the majority of the data. Outliers may be the result of **measurement error**, i.e. if the values were recorded incorrectly, although sometimes they may result from extreme cases (which are often seen in small data sets). How far does a value have to be outside the main distribution to be considered an outlier? There are no concrete rules, and often it is a **judgement call** depending on how much they are likely to influence your analyses. Inspecting the distributions graphically first can provide insight into odd values.

Question 2.3.2 Make boxplot of height

```
boxplot(steps.data$height_cm, main="Boxplot of height")
```

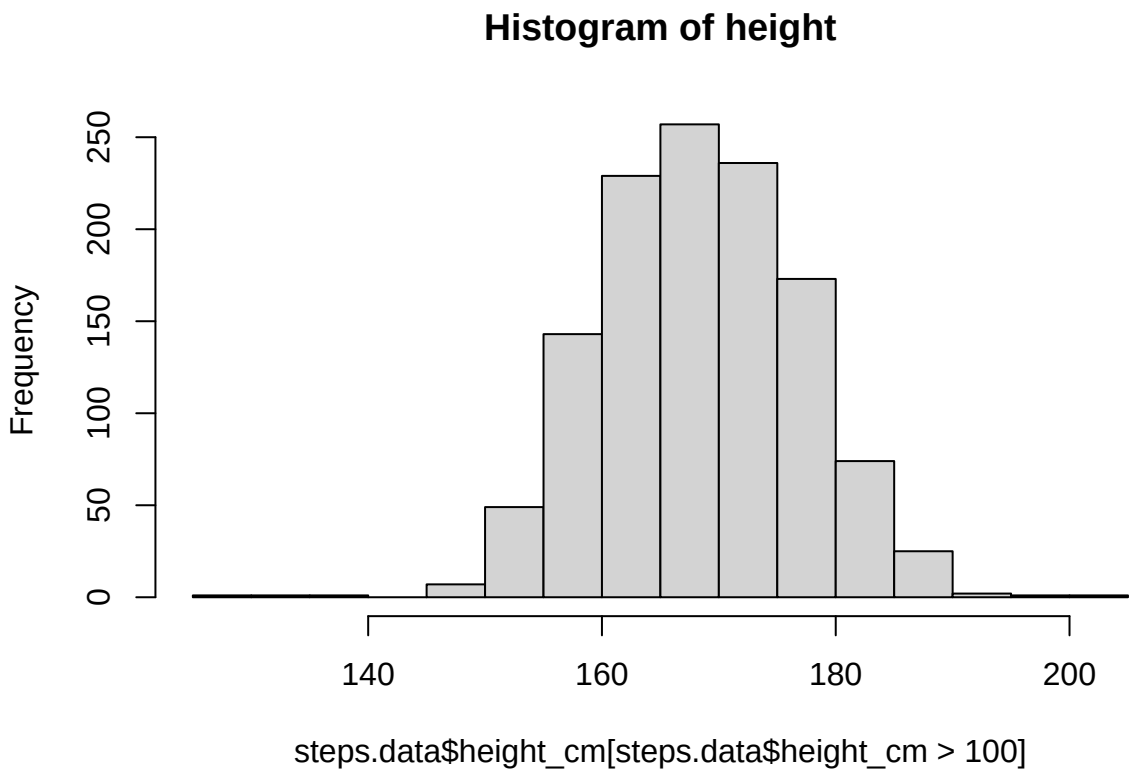


The horizontal line in the box indicates the median, with the upper edges of the box representing the upper and lower quartiles. We can see that the median is around 170, and the IQR is quite narrow and lies approximately between the values of 162-175. The bars off the box are $1.5 * IQR$. Any values that are greater or lower than $1.5 * IQR$ are represented as dots, and these can be thought of as **outliers**.

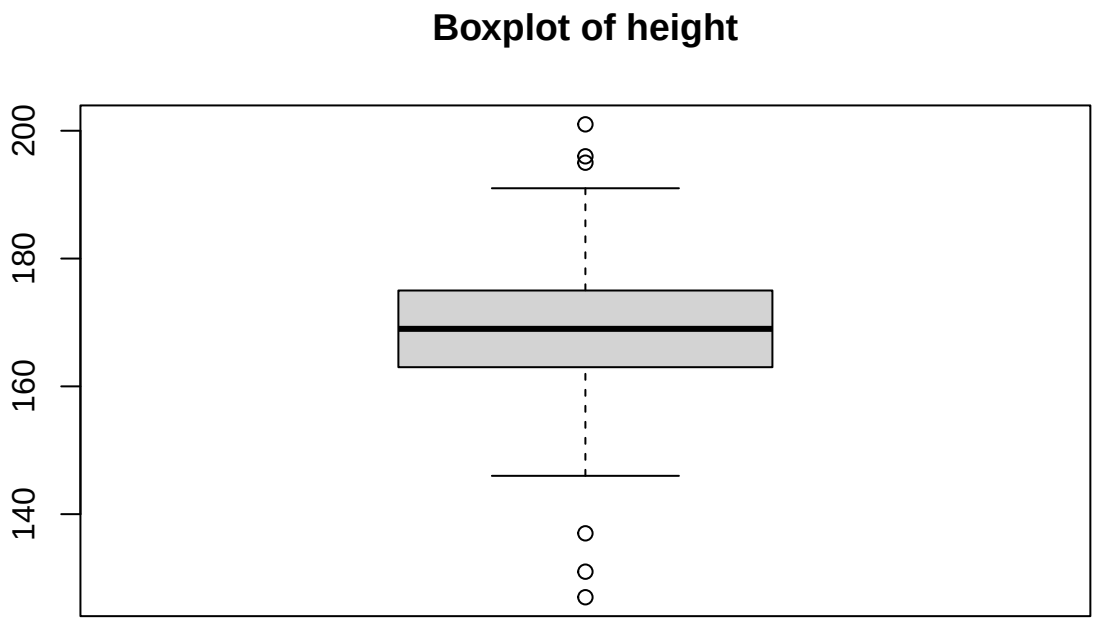
We can see that there are about 4 outliers in the upper range and 4 in the lower range. Of these, there seems to be 1 individual with a height of about 80 cm. Perhaps it ought to be 180 cm? Difficult to tell…

See how limiting our plots to those with height > 100 cm look:

```
hist(steps.data$height_cm[steps.data$height_cm>100], main="Histogram of height")
```




```
boxplot(steps.data$height_cm[steps.data$height_cm>100], main="Boxplot of height")
```



In the **advanced** section you can learn how to produce histograms, boxplots, and violin plots using **ggplot2**

---

3. Transforming data in R

In this section you will recode original variables and make them ready for analysis

An essential part of data cleaning is getting variables ready for analysis. Computing new variables such as BMI from weight and height or the average of SBP out of three different BP measures from the same participant are good examples. While doing this, you will want to check for implausible or missing values. You might also want to categorise a continuous variable into clinically meaningful groups, such as BMI categories.

3.1 Calculate new variables

Question 3.1.1 Calculate the body-mass index for all participants.

HINT: You can use mathematical symbols to create new variables

```
30+50 #Just a sum
```

```
## [1] 80
```

```
a <- 30 #Assign values to objects
b <- 50

a+b #And then use the objects for calculations
```

```
## [1] 80
```

```
c <- a+b
c
```

```
## [1] 80
```

You can also make operations with vectors

```
vec.a <- c(1,2,4,5,6,7,8,3)
vec.b <- c(2,3,6,7,8,9,5,4)
vec.c <- vec.a+vec.b
```

Try to guess the result before making the calculation

```
vec.c
```

```
## [1]  3  5 10 12 14 16 13  7
```

Finally, notice that you can use columns in data frames in the same way as vectors above, by using `$` to indicate the column

```
steps.data$height_cm # If you run this, the console will display a lot of values...
```

Now, try to calculate BMI.

Result to question 3.1.1

```
steps.data$bmi <- NA # It is a good idea to start a new variable by defining the column with all observ
# use indentations and lines to facilitate reading your code
steps.data$bmi <- # Assign to the new variable bmi
  steps.data$weight_kg / # weight in kg
  ((steps.data$height_cm)/100)^2 # height in m squared
```

Question 3.1.2 How does the newly created variable `bmi` look? Look out for **implausible** or **extreme** values.

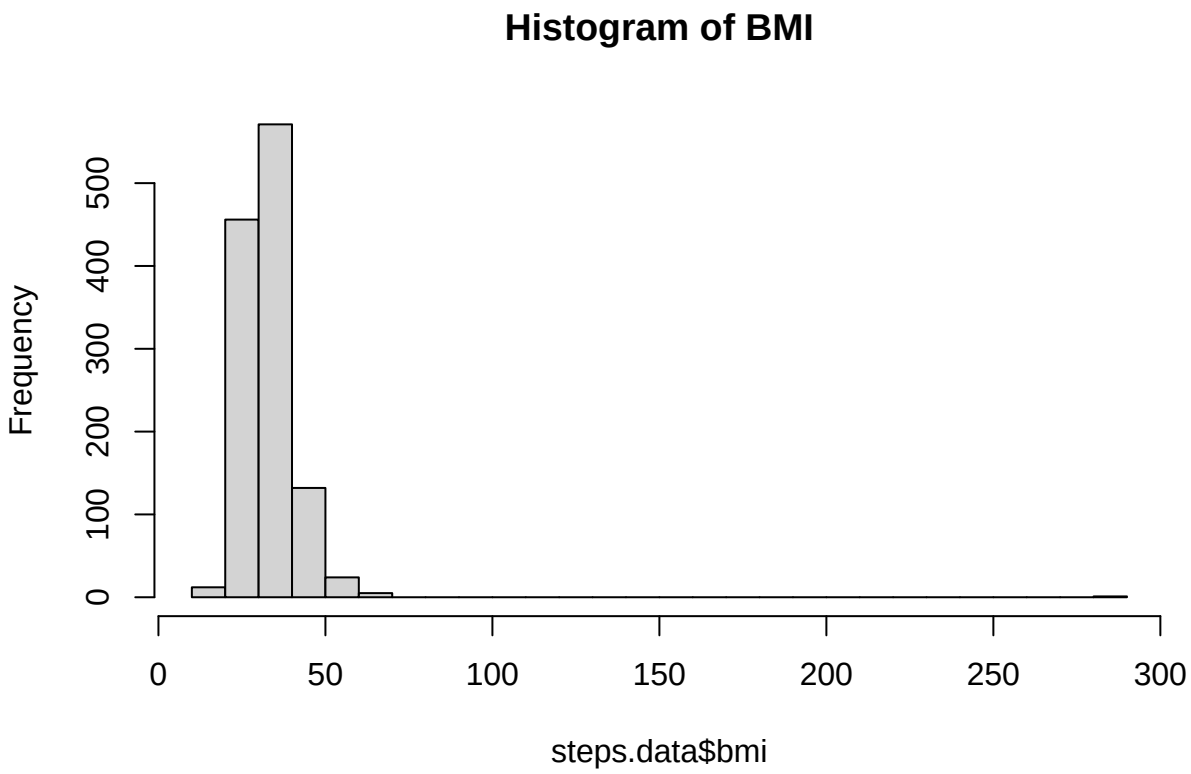
Hint, use the commands learned above `summary()` or `hist()`.

Result 3.1.2

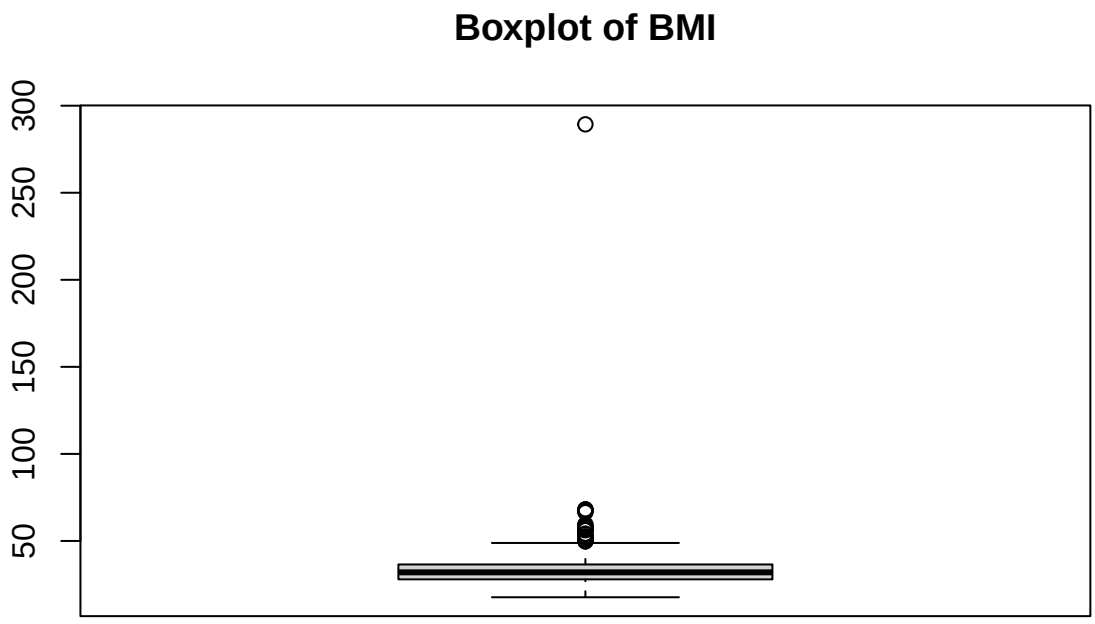
```
summary(steps.data$bmi)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   17.67   27.94   31.98   33.02   36.51  289.28      88
```

```
hist(steps.data$bmi, breaks=30, main="Histogram of BMI")
```




```
boxplot(steps.data$bmi, main="Boxplot of BMI")
```



It looks like one participant has a BMI of nearly $300 kg/m^2$. This is certainly an **implausible** value. Moreover, there are possibly quite a few extreme values or outliers as well, but these are not easy to identify at the moment.

For a moment, let's consider that any BMI values above 60 are implausible and set them as missing.

NOTE: This decision should be (clinically, biologically, etc) informed and appropriately documented in your analyses.

Use the following code:

```
steps.data$bmi[steps.data$bmi>60] <- NA #Setting implausible values (i.e. BMI >60) as missing
```

Notice how `[ ]` are used to identify the rows that meet the condition.

Question 3.1.3 How does the variable `BMI` look like after dropping these implausible values?
HINT: use the functions learned above.

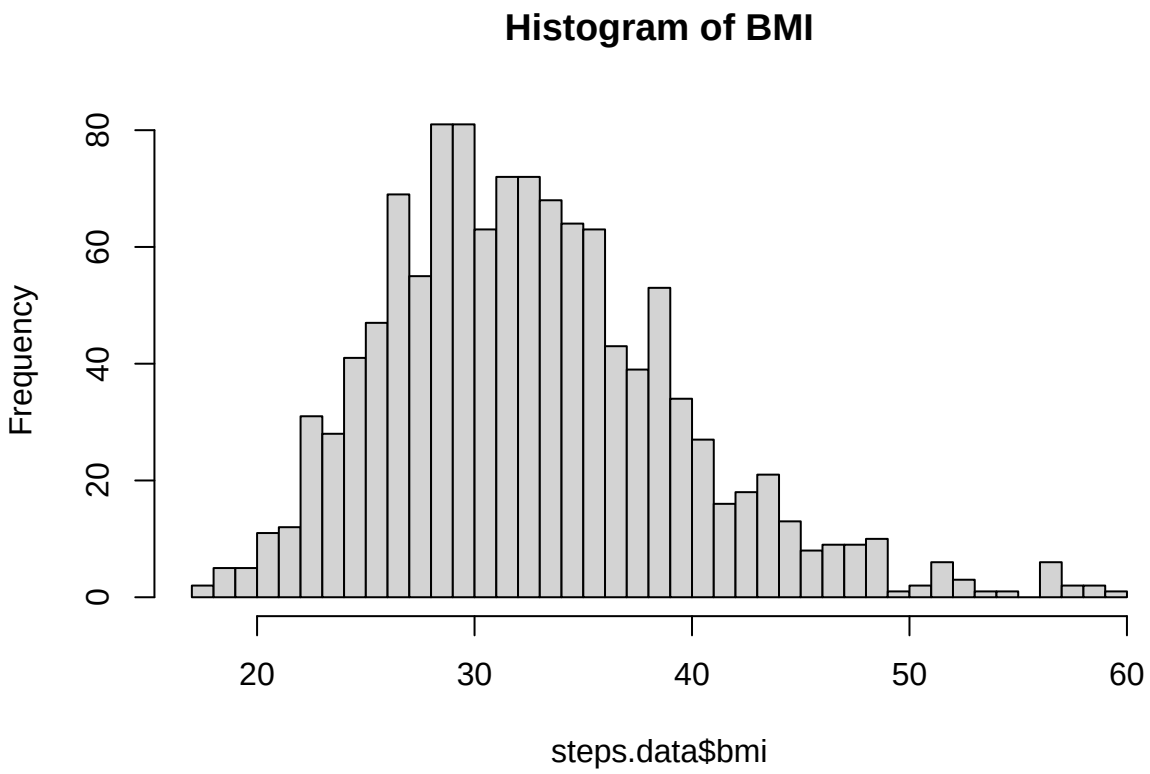
Result to question 3.3

```
summary(steps.data$bmi)
```

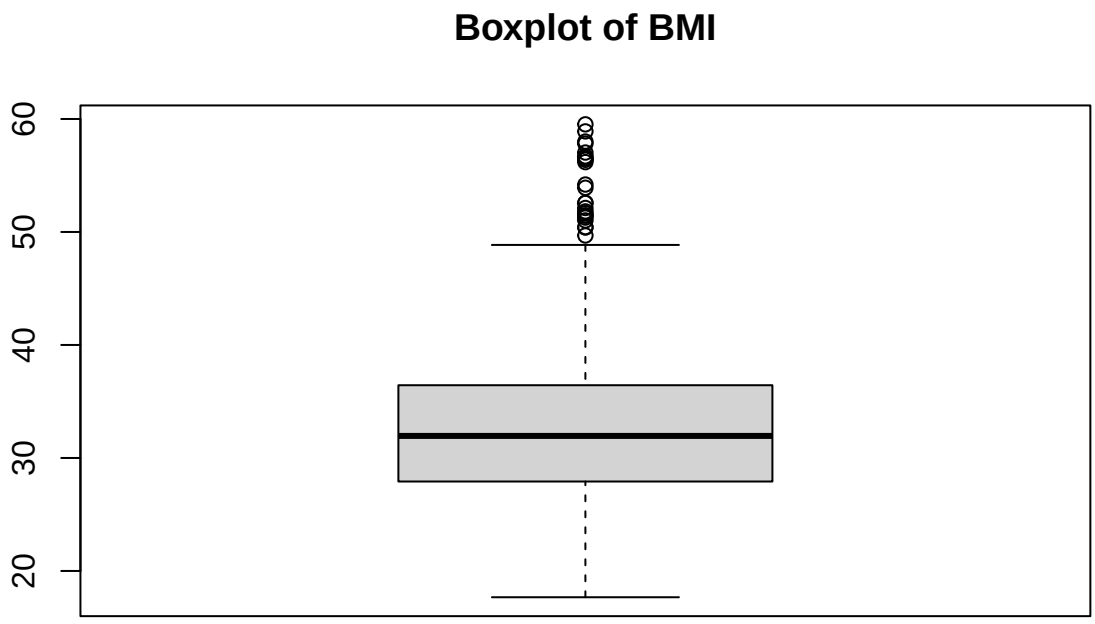
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   17.67   27.92   31.95   32.66   36.44   59.52      94
```

Note how the number of `NA`s increased from 88 to 94. Also, the mean changed from 33.02 to 31.95 but the median was nearly unaltered.

```
hist(steps.data$bmi, breaks=30, main="Histogram of BMI")
```




```
boxplot(steps.data$bmi, main="Boxplot of BMI")
```



After setting implausible values as missing, the distribution of the newly-created BMI variable now looks much more like a normally-distributed trait. There seems to be a few **outliers** that might need to be addressed at a later stage (or maybe not at all, as these could be informative).

Question 3.1.4 Consider how the implausible, and extreme values and outliers identified in section 2.4 for `height_cm` might impact the presence of implausible and extreme values in the newly-creted BMI.

Result to question 3.1.4

The implausible values in BMI are highly likely to come from implausible values in either weight, height or both. It is therefore best to check for implausible values in the original variables first and then in the new variable.

3.2 Group a continuous variable into categories

There are many reasons why you would want to categorise a continuous variable, and cut-offs need to be defined for different categories. BMI is a skewed variable and different parts of BMI's distribution may have different relationships with disease.

Question 3.2.1 Group BMI into this categories:

bmi_who: <18.5, 18.5-24,25-29, 30+

HINT Try to adapt the code from above to define implausible values as missing but be careful in using the correct columns

Result to Question 3.2.1

```
steps.data$bmi_who<-NA #Define the row as all missing
steps.data$bmi_who[steps.data$bmi<18.5]<-0 #Those with bmi <18.5 assigned with a 0 in column bmi_who
```

```
steps.data$bmi_who[steps.data$bmi>=18.5 & steps.data$bmi<25]<-1 #Those with bmi 18.5-24.9 assigned with
steps.data$bmi_who[steps.data$bmi>=25 & steps.data$bmi<30]<-2 #Those with bmi 25-29.9 assigned with a 3
steps.data$bmi_who[steps.data$bmi>=30]<-3 #Those with bmi <18.5 assigned with a 0 in column bmi_who
```

Now check the new variable

```
summary(steps.data$bmi_who)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   2.000   3.000   2.493   3.000   3.000      94
```

It seems like R still thinks this new variable is numeric. You can use the `class()` function to check this.

```
class(steps.data$bmi_who)
```

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

Because this new categorical variable is ordered, it is a good idea change the format. **Factor** is the class of variable that is useful for ordered categories.

Question 3.2.2 Change the format of this variable to factor.

HINT: Use the function `factor()` and then explore the changes.

Result to question 3.2.2

```
steps.data$bmi_who <- factor(steps.data$bmi_who, # define as factor
                             labels=c("<18.5", "18.5-24.9", "25-29.9", ">30")) #add labels
```

```
summary(steps.data$bmi_who)
```

```
##     <18.5 18.5-24.9   25-29.9       >30      NA's
##         4       130       334       727        94
```

Another option using the function `cut()`

```
steps.data$bmi_who2 <- cut(steps.data$bmi, # defines as factor
                           breaks=c(0,18.5,25,30,100), # defines breaks
                           labels=c("<18.5", "18.5-24.9", "25-29.9", ">30")) # adds label
```

It is always a good idea to compare methods and see if you reach the same result.

```
summary(steps.data$bmi_who2)
```

```
##     <18.5 18.5-24.9   25-29.9       >30      NA's
##         4       131       333       727        94
```

Or even better, use the `table()` function.

```
table(steps.data$bmi_who,steps.data$bmi_who2)
```

```
##
##             <18.5 18.5-24.9 25-29.9 >30
##   <18.5         4         0       0   0
##   18.5-24.9     0       130       0   0
##   25-29.9       0         1     333   0
##   >30           0         0       0 727
```

It seems there is 1 individual who was categorised differently by these methods. Use `help()` to investigate how `cut()` and it's argument `breaks` work to find out.

3.3 Transformations for skewed data

NEED TO ADD section but interested in feedback of whether it's useful

Will only briefly mention, and will include reference material on how to do this in R (i.e., explore skewness, visualize, transform)

---

4. Selecting

> Here you will learn how to select the rows and columns you want to use for further data cleaning and/or for data analyses

Based on the main objective of your research project, your pre-specified analyses likely have outlined what variables you will be using and what subset of participants you are interested in (if at all).

It is therefore useful to limit your "downstream" analyses to only the data that is necessary for the project.

Suppose we are only interested in exploring if `BMI` varies by `gender` and by `age`. It would make sense not including any other variable in the data set we are using.

The best way to do this, to ensure the format of the columns is preserved is using the `data.frame()` function

```
new.steps.data<-c() #Create an empty object
new.steps.data<-data.frame(
  "age"=steps.data$age,        #Create column "age" using the column age from object steps.data
  "bmi"=steps.data$bmi,        #Create column "bmi" using the column bmi from object steps.data
  "gender"=steps.data$gender)  #Create column "gender" using the column gender from object steps.data
```

If, for some reason, you are only interested in those aged above 40 years, you can use the function `subset()` which can help you do both steps at once.

```
new.steps.data.2<-subset(steps.data,
                         age > 40,
                         select = c(age, bmi, gender))
```

Summary

NEED TO ADD SUMMARY OF LEARNING OBJECTIVES

Advanced

> This section is still under construction (probably 50% way through)

Would really appreciate your feedback on what to include here.

Export to folder within R project

Create new folder with R

R can create folders for you with the `dir.create` function. To avoid overwriting folders (and files!) it is important to ask R to check if the directory exists *before* asking it to create it.

This can be accomplished with the following:

```
if (file.exists(here::here("data"))) { #check
 cat("The folder already exists")
} else {
 dir.create(here::here("data"))
}
```

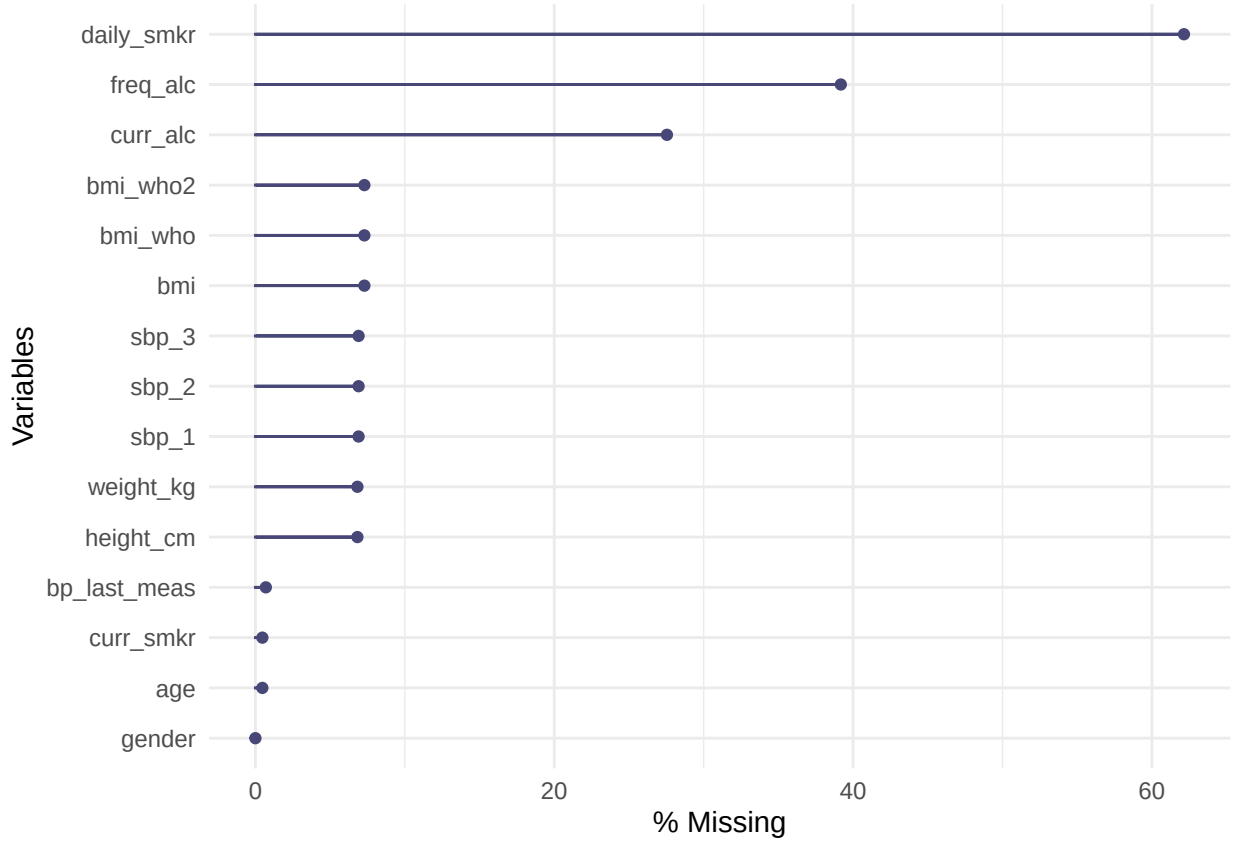
Export to data folder

```
export(steps.data, here("data","test-STEPS-export.csv"))
```

Advanced tools to visualise missing data

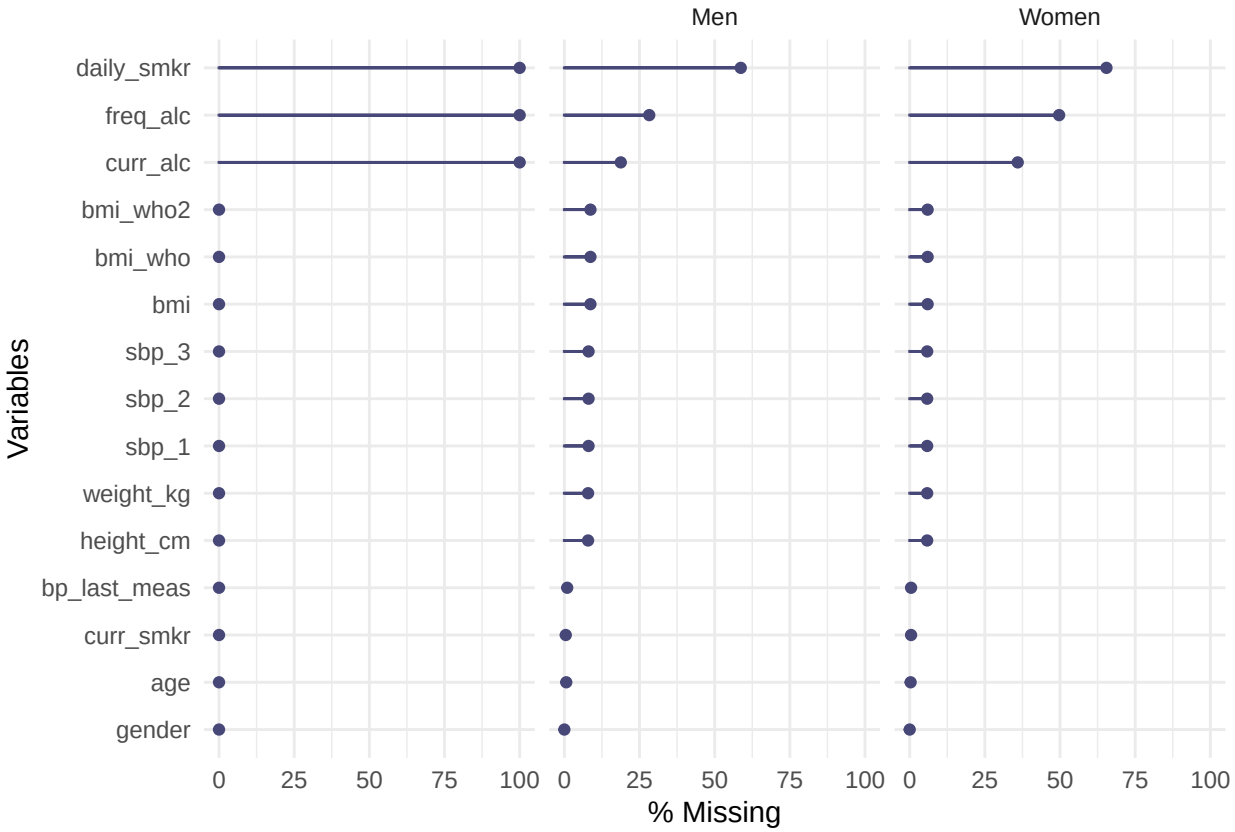
```
pacman::p_load(naniar)
gg_miss_var(steps.data, show_pct=T)
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a guide. Please
## use `guide = "none"` instead.
```




```
steps.data %>%
  mutate(sex = as.factor(gender)) %>%
  gg_miss_var(show_pct=T, facet = sex)
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a guide. Please
## use `guide = "none"` instead.
```



The workflow for data cleaning using pipes

```
plot2.df <- steps.data %>%  # Make a new data frame for plotting
  mutate(sbp = (sbp_1+sbp_2+sbp_3)/3) %>% # Compute sbp as the average of the three sbp measures availa
  filter(sbp < 250) %>% # Filter out those with implausibly high levels of sbp
  select(one_of("sbp","gender")) %>% # Only keep the variables sbp and gender
  drop_na() # Drop missing observations
```

Plots with ggplot2

From EpiRHandbook:

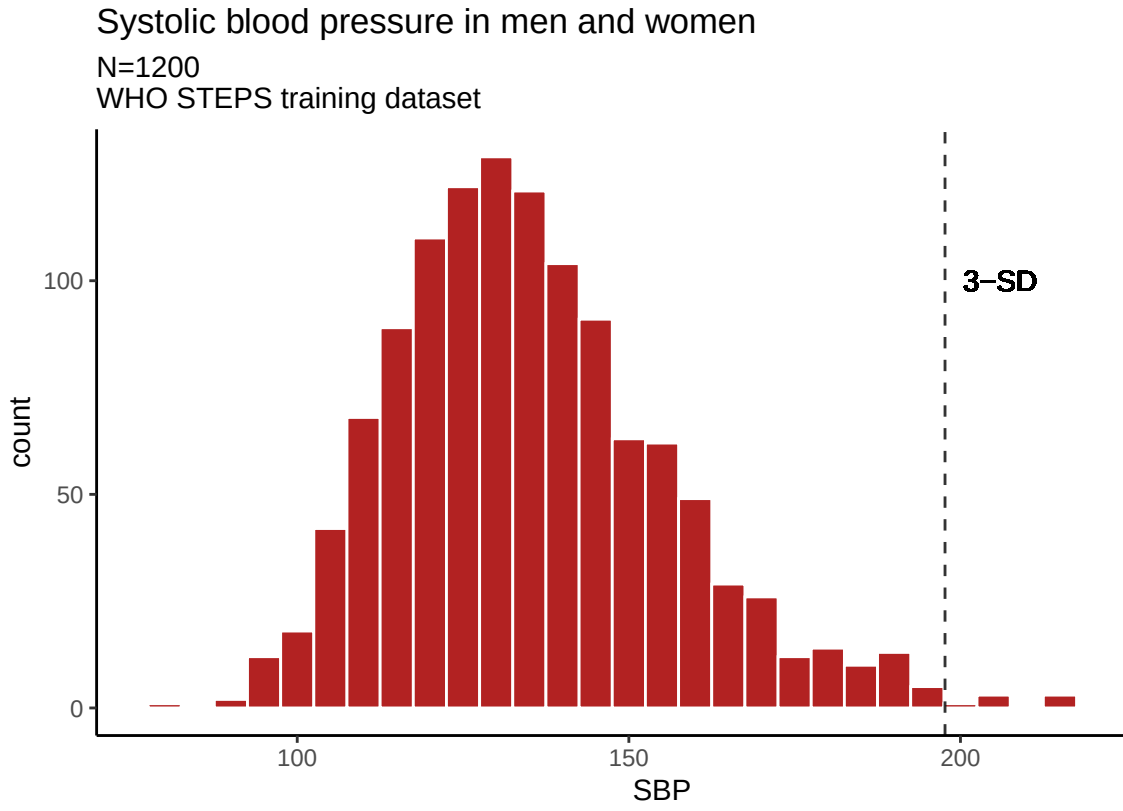
`ggplot2` is the most popular data visualisation R package. Its `ggplot()` function is at the core of this package, and this whole approach is colloquially known as "ggplot" with the resulting figures sometimes affectionately called "ggplots". The "**gg**" in these names reflects the "**grammar of graphics**" used to construct the figures. `ggplot2` benefits from a wide variety of supplementary R packages that further enhance its functionality.

The syntax is **significantly** different from base `R` plotting, and has a learning curve associated with it. Using `ggplot2` generally requires the user to format their data in a way that is highly `tidyverse` compatible, which ultimately makes using these packages together very effective.

SBP in men & women (histogram)

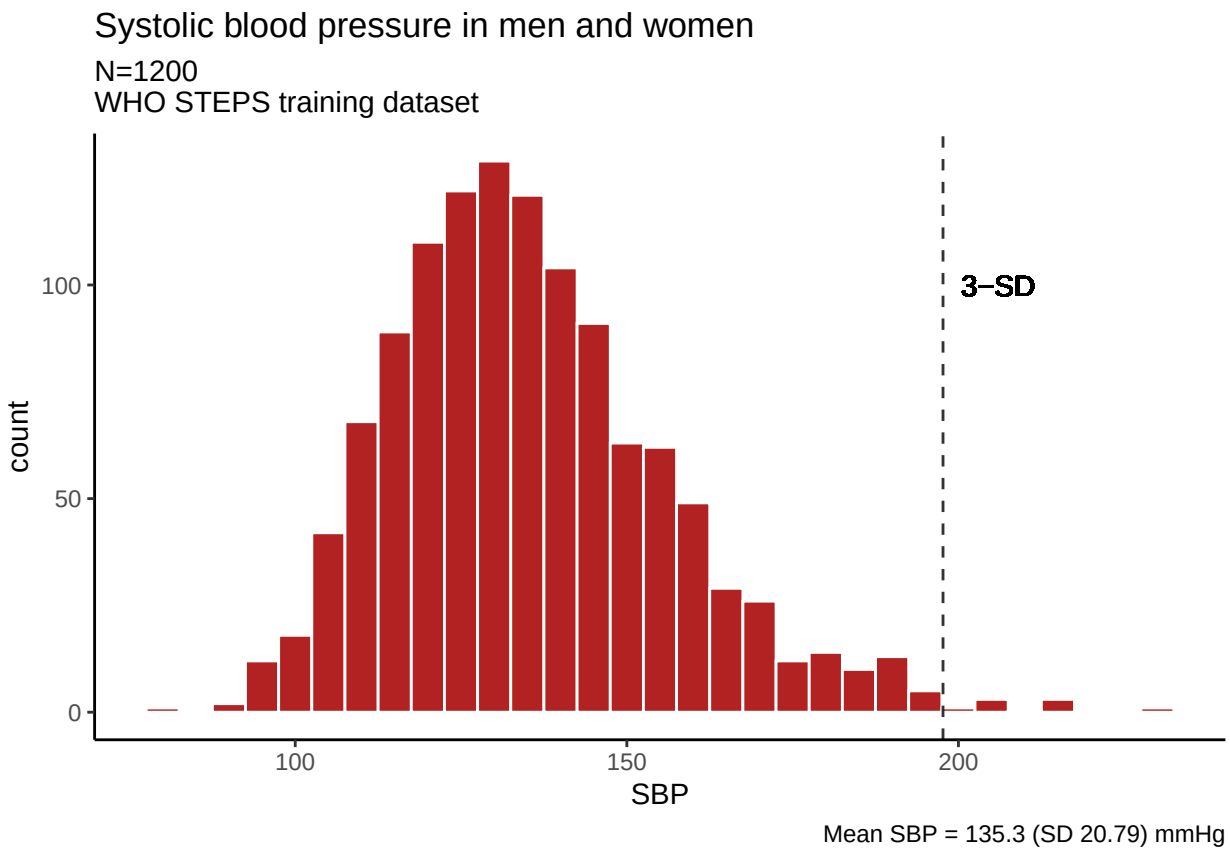
```
  plot2<- # Assign the result of yout ggplot to an object called plot2
    ggplot(data = plot2.df, mapping = aes(x = sbp)) + # Map your graph to sbp in the x axis using data
```

```
    geom_histogram(binwidth = 5, color = "white", fill = "firebrick") +

    labs(
      title = "Systolic blood pressure in men and women",
      subtitle = stringr::str_glue("N={dim(plot2.df)[1]}\nWHO STEPS training dataset"),
      x = "SBP",
      caption = stringr::str_glue("Mean SBP = {round(mean(plot2.df$sbp),2)} (SD {round(sd(plot2.df$sbp
    ) +

    theme_classic() +

    geom_vline(
      xintercept = mean(plot2.df$sbp)+3*sd(plot2.df$sbp),
      color = "black",
      linetype = 2,
      alpha = 0.8) +

    geom_text(
      x = mean(plot2.df$sbp)+3.4*sd(plot2.df$sbp),
      y = 100,
      label = "3-SD"
    )

plot2
```



Draw basic histogram

```r
plot2<- # Assign the result of yout ggplot to an object called plot2
  ggplot(data = plot2.df, mapping = aes(x = sbp)) + # Map your graph to sbp in the x axis using data
    geom_histogram(binwidth = 5, color = "white", fill = "firebrick") +

    labs(
      title = "Systolic blood pressure in men and women",
      subtitle = stringr::str_glue("N={dim(plot2.df)[1]}\nWHO STEPS training dataset"),
      x = "SBP",
      caption = stringr::str_glue("Mean SBP = {round(mean(plot2.df$sbp),2)} (SD {round(sd(plot2.df$sbp
    ) +

    theme_classic() +

    geom_vline(
      xintercept = mean(plot2.df$sbp)+3*sd(plot2.df$sbp),
      color = "black",
      linetype = 2,
      alpha = 0.8) +

    geom_text(
      x = mean(plot2.df$sbp)+3.4*sd(plot2.df$sbp),
      y = 100,
      label = "3-SD"
    )

plot2
```

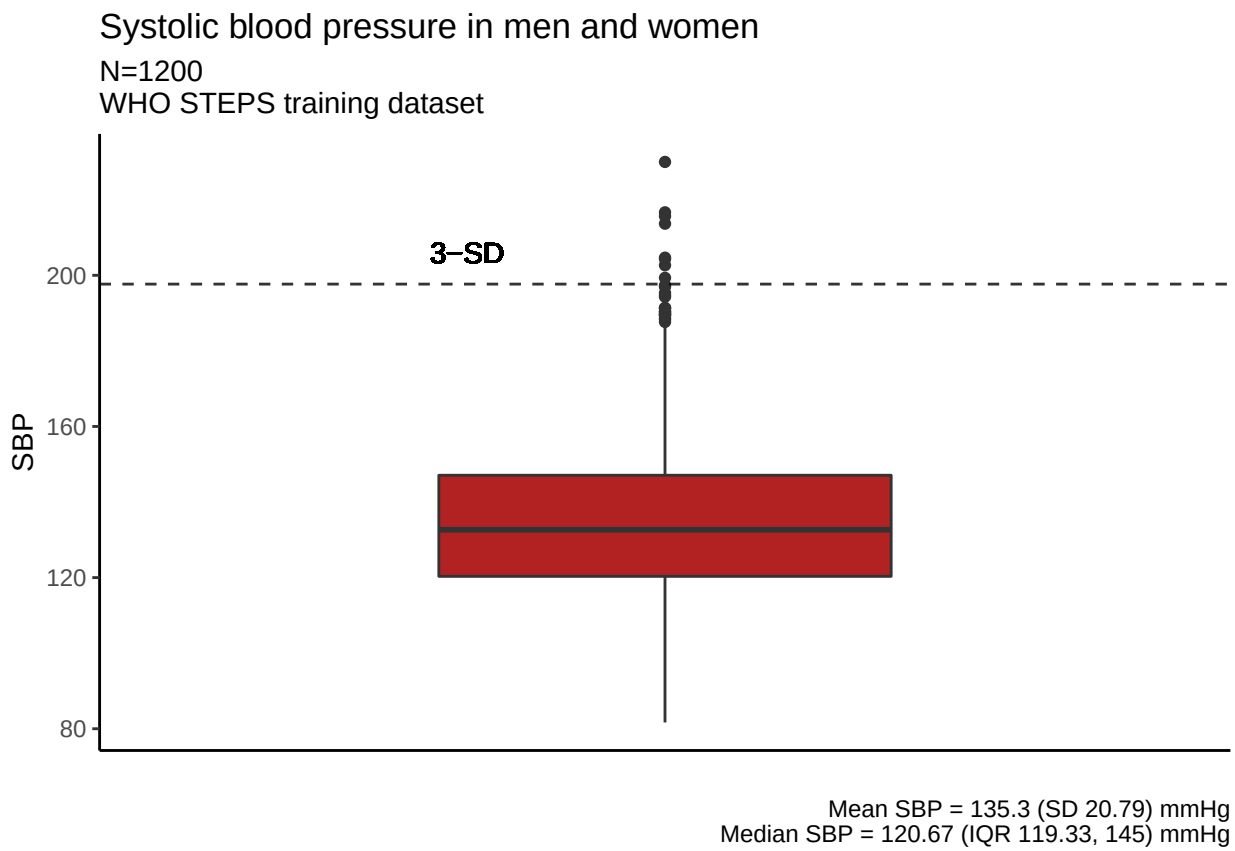


Save

```
ggsave(plot2, filename = here("figs","plot2.png"), width = 5, height = 4)
```

```
plot3<-
    ggplot(data = plot2.df, mapping = aes(y = sbp)) +
      geom_boxplot(fill = "firebrick", width = 0.8) +
      #geom_density() +
      labs(
        title = "Systolic blood pressure in men and women",
        subtitle = stringr::str_glue("N={dim(plot2.df)[1]}\nWHO STEPS training dataset"),
        y = "SBP",
        x = "",
        caption = stringr::str_glue(
          "Mean SBP = {round(mean(plot2.df$sbp),2)} (SD {round(sd(plot2.df$sbp),2)}) mmHg\nMedian SBP =
      ) +
      theme_classic() +
      #facet_wrap(~sex_fct)
      geom_hline(
        yintercept = mean(plot2.df$sbp)+3*sd(plot2.df$sbp),
        color = "black",
        linetype = 2,
        alpha = 0.8) +
      geom_text(
        y = mean(plot2.df$sbp)+3.4*sd(plot2.df$sbp),
        x = -0.35,
        label = "3-SD") +
```

```
    scale_x_discrete(breaks = NULL)
plot3
```



Save

```
ggsave(plot3, filename = here("figs","plot3.png"), width = 5, height = 4)
```