Exercise 7. Data preprocessing and exploratory analysis

Adéla Vrtková, Martina Litschmannová, Michal Béreš

1. Packages - installation and loading

In [1]:
```
# You only need to install packages once(if you don't already have them)
# install.packages("readxl")
# install.packages("dplyr")
# install.packages("openxlsx")
# install.packages("rstatix")
```

In [2]:
```
# Loading the package(must be repeated every time you restart R, it is advisable to have it
# at the beginning of the script)
library(readxl)
library(dplyr)
library(openxlsx)
library(rstatix)
# contains notifications of overwritten functions or older versions of the package
```

```
Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union



Attaching package: ‘rstatix’


The following object is masked from ‘package:stats’:

    filter
```

2. Working directory - where we load and where we store data

- **Attention, the current open folder in Rstudio, or the location of the Rskcript is not automatically a working directory**

In [3]:
```
# Working directory listing
getwd()
```

'/home/ber0061/Repositories/PS_eng_2022/Exercise 7'

In [4]:
```
# Working directory setting -> in quotation marks, full path(relative or absolute)
setwd("./data")
```

In [5]:
```
getwd() # Where are we now?
```

'/home/ber0061/Repositories/PS_eng_2022/Exercise 7/data'

In [6]:
```
setwd("./..") # back again
```

In [7]:
```
getwd() # for control
```

'/home/ber0061/Repositories/PS_eng_2022/Exercise 7'

3. Load data file

From CSV file

Basic functions - read.table, read.csv, read.csv2,... It depends mainly on the file format(.txt,.csv), the so-called separator of individual values, decimal point/dot

In [8]:
```
# Load and save a data file in csv2 format from the working directory
data = read.csv2(file="aku.csv")
```

In [9]:
```
head(data)
```

A data.frame: 6 × 8

	A5	B5	C5	D5	A100	B100	C100	D100
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1946.5	2006.5	1881.8	1806.9	1780.4	1654.2	1663.3	1668.4
2	1963.5	1991.5	1890.4	1788.1	1751.4	1663.1	1641.1	1641.9
3	1934.3	1988.8	1865.7	1775.0	1743.5	1633.3	1621.5	1620.0
4	1934.8	1975.4	1880.7	1805.4	1727.4	1642.2	1610.7	1685.8
5	1939.9	1998.4	1861.1	1775.7	1728.8	1656.7	1624.6	1610.5
6	1925.9	2012.3	1887.3	1807.3	1767.5	1664.4	1604.6	1670.6

In [10]:
```
data = read.csv2(file="aku.csv", sep=";", quote="", skip=0, header=TRUE)
head(data)
```

A data.frame: 6 × 8

	A5	B5	C5	D5	A100	B100	C100	D100
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1946.5	2006.5	1881.8	1806.9	1780.4	1654.2	1663.3	1668.4
2	1963.5	1991.5	1890.4	1788.1	1751.4	1663.1	1641.1	1641.9
3	1934.3	1988.8	1865.7	1775.0	1743.5	1633.3	1621.5	1620.0
4	1934.8	1975.4	1880.7	1805.4	1727.4	1642.2	1610.7	1685.8
5	1939.9	1998.4	1861.1	1775.7	1728.8	1656.7	1624.6	1610.5
6	1925.9	2012.3	1887.3	1807.3	1767.5	1664.4	1604.6	1670.6

In [11]:
```
# Load and save a csv2 data file from the local disk to the data frame
data = read.csv2(file="./data/aku.csv")
```

In [12]:
```
# Load and save a csv2 data file from the Internet to the data frame
data = read.csv2(file="http://am-nas.vsb.cz/lit40/DATA/aku.csv")
```

From Excel(xlsx file)

Loading and saving a data file in xlsx format from the local disk to the data frame We use the function from the readxl package, which we expanded in the introduction

In [13]:
```
data = read_excel("./data/aku.xlsx",
                  sheet = "Data",          # worksheet specification in xlsx file
                  skip = 3)                # lines to be skipped
```

```
New names:
* `` -> ...1
* `Manufacturer A` -> `Manufacturer A...2`
* `Manufacturer B` -> `Manufacturer B...3`
* `Manufacturer C` -> `Manufacturer C...4`
* `Manufacturer D` -> `Manufacturer D...5`
* ...
```

In [14]:
```
head(data)
```

A tibble: 6 × 9

...1	Manufacturer A...2	Manufacturer B...3	Manufacturer C...4	Manufacturer D...5	Manufacturer A...6	Manufacturer B...7	Manufacturer C...8	Manufacturer D...9
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1946.5	2006.5	1881.8	1806.9	1780.4	1654.2	1663.3	1668.4
2	1963.5	1991.5	1890.4	1788.1	1751.4	1663.1	1641.1	1641.9
3	1934.3	1988.8	1865.7	1775.0	1743.5	1633.3	1621.5	1620.0

...1	Manufacturer A...2	Manufacturer B...3	Manufacturer C...4	Manufacturer D...5	Manufacturer A...6	Manufacturer B...7	Manufacturer C...8	Manufacturer D...9
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
4	1934.8	1975.4	1880.7	1805.4	1727.4	1642.2	1610.7	1685.8
5	1939.9	1998.4	1861.1	1775.7	1728.8	1656.7	1624.6	1610.5
6	1925.9	2012.3	1887.3	1807.3	1767.5	1664.4	1604.6	1670.6

Remove unnecessary rows/columns and name rows/columns for easier data addressing

In [15]:

```
# Indexing with negative indexes returns everything except the index value
# do not mix negative and positive indices!
data = data[,-1] # delete the first column with indexes
head(data)
```

A tibble: 6 × 8

Manufacturer A...2	Manufacturer B...3	Manufacturer C...4	Manufacturer D...5	Manufacturer A...6	Manufacturer B...7	Manufacturer C...8	Manufacturer D...9
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1946.5	2006.5	1881.8	1806.9	1780.4	1654.2	1663.3	1668.4
1963.5	1991.5	1890.4	1788.1	1751.4	1663.1	1641.1	1641.9
1934.3	1988.8	1865.7	1775.0	1743.5	1633.3	1621.5	1620.0
1934.8	1975.4	1880.7	1805.4	1727.4	1642.2	1610.7	1685.8
1939.9	1998.4	1861.1	1775.7	1728.8	1656.7	1624.6	1610.5
1925.9	2012.3	1887.3	1807.3	1767.5	1664.4	1604.6	1670.6

In [16]:

```
# Rename columns - if necessary
colnames(data)=c("A5","B5","C5","D5","A100","B100","C100","D100")
head(data)
```

A tibble: 6 × 8

A5	B5	C5	D5	A100	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1946.5	2006.5	1881.8	1806.9	1780.4	1654.2	1663.3	1668.4
1963.5	1991.5	1890.4	1788.1	1751.4	1663.1	1641.1	1641.9
1934.3	1988.8	1865.7	1775.0	1743.5	1633.3	1621.5	1620.0
1934.8	1975.4	1880.7	1805.4	1727.4	1642.2	1610.7	1685.8
1939.9	1998.4	1861.1	1775.7	1728.8	1656.7	1624.6	1610.5
1925.9	2012.3	1887.3	1807.3	1767.5	1664.4	1604.6	1670.6

Note(which is good to read until the end....)

in Rstudio) it is possible to import using "Import Dataset" from the Environment window without having to write the code. In this case, however, there must be no special characters(hooks, commas) in the "path" to the file. Otherwise, an error will appear. The object imported this way will be in the new RSstudio as type "tibble". This is a more modern "data.frame" and in some features it can cause problems and throw errors! You can easily convert this object to type data.frame using **as.data.frame()**

4. Pre-processing data + Dplyr library

Overview of Dplyr library functions

- **%>%** is a so-called pipe operator, typical usage is "res=data %>% operation", where the result is a operation calibrated to data
- **select(...)** is one of the operations that we can insert into the "pipe" operator - it is used to select data
  - select(1) - selects the first column
  - select(A5) - selects the column named A5
  - select(1,3,5) - selects columns 1,3,5
- **mutate(new_column=...)** is an operation that produces a new data column in the data frame using the specified calculation over the current columns
  - data %>% mutate(C=A-B) produces a new column named "C" in the "data" data frame as the difference of the values in the existing columns "A" and "B"
- **filter(...)** filters values from the data that meet the specified requirements

- data %>% filter(manufacturer=="A"|manufacturer=="B") returns a data file that has only "A" or "B" values in the "manufacturer" column
- data %>% filter(manufacturer=="A", values>1000) if we write the requirements one after the other(separated by a comma) we understand it as logical **and**

- **summarize(...)** calculate the prescribed numerical characteristics within the specified columns(suitable for combination with group.by)
  - data %>% summarize(prum=mean(kap5), median=median(kap5))
- **arrange(...)** ascending or descending row order
  - data %>% arrange ascending
  - data %>% arrange(desc) descending
- **group_by(...)** grouping of data according to unique values in the specified column
  - data %>% group_by(manufacturer)

Very useful "cheat sheet" can be found here: https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

Column/row selections

In [17]:
```r
# Display of the first six lines
head(data)
```

A tibble: 6 × 8

A5	B5	C5	D5	A100	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1946.5	2006.5	1881.8	1806.9	1780.4	1654.2	1663.3	1668.4
1963.5	1991.5	1890.4	1788.1	1751.4	1663.1	1641.1	1641.9
1934.3	1988.8	1865.7	1775.0	1743.5	1633.3	1621.5	1620.0
1934.8	1975.4	1880.7	1805.4	1727.4	1642.2	1610.7	1685.8
1939.9	1998.4	1861.1	1775.7	1728.8	1656.7	1624.6	1610.5
1925.9	2012.3	1887.3	1807.3	1767.5	1664.4	1604.6	1670.6

In [18]:
```r
# Display of the last six lines
tail(data)
```

A tibble: 6 × 8

A5	B5	C5	D5	A100	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1962.8	2000.8	NA	NA	1803.3	1664.3	NA	NA
NA	2001.1	NA	NA	NA	1627.6	NA	NA
NA	2000.4	NA	NA	NA	1655.5	NA	NA
NA	1998.6	NA	NA	NA	1634.4	NA	NA
NA	2000.1	NA	NA	NA	1645.7	NA	NA
NA	1993.6	NA	NA	NA	1673.9	NA	NA

In [19]:
```r
# Display of line 10
data[10,]
```

A tibble: 1 × 8

A5	B5	C5	D5	A100	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1944	2002	1887	1872.2	1740.6	1634.7	1630.1	1709.8

In [20]:
```r
# Display of the 3rd column - several ways
tmp = data[,3]
head(tmp)
```

A tibble: 6 × 1

C5
<dbl>
1881.8
1890.4
1865.7
1880.7

C5
<dbl>
1861.1
1887.3

In [21]:

```
# or(if we know the name of the variable written in the 3rd column)
data$C5
```

1881.8 · 1890.4 · 1865.7 · 1880.7 · 1861.1 · 1887.3 · 1922 · 1926.1 · 1898.8 · 1887 · 1908.5 · 1890.7 · 1914.2 · 1899.4 · 1932 · 1862 · 1873.5 · 1878.5 · 1898.4 · 1903.2 · 1887.8 · 1884.7 · 1902 · 1929.4 · 1900.3 · 1872.8 · 1893.2 · 1929.9 · 1884.3 · 1901 · 1899 · 1885.8 · 1892.7 · 1893.3 · 1886.1 · 1892 · 1908 · 1904.4 · 1925.9 · 1924.2 · 1928.4 · 1920.9 · 1942.5 · 1928.8 · 1901.2 · 1905.5 · 1890.8 · 1923.4 · 1891.3 · 1927 · 1891.8 · 1904.7 · 1887.3 · 1887.4 · 1899.7 · 1910.5 · 1891.2 · 1882 · 1929.7 · 1911.9 · 1877.2 · 1874.1 · 1877.2 · 1901.7 · 1933.1 · 1848.4 · 1905.1 · 1901.4 · 1936.1 · 1889.3 · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA>

In [22]:

```
# or using the dplyr package select function, which selects the selected columns
tmp = data %>% select(C5)
head(tmp)
```

A tibble:
6 × 1

C5
<dbl>
1881.8
1890.4
1865.7
1880.7
1861.1
1887.3

In [23]:

```
# Save the first and fifth columns of data frame
data_1_5 = data[,c(1,5)]
head(data_1_5)
```

A tibble: 6 × 2

A5	A100
<dbl>	<dbl>
1946.5	1780.4
1963.5	1751.4
1934.3	1743.5
1934.8	1727.4
1939.9	1728.8
1925.9	1767.5

In [24]:

```
# or using the dplyr function
data_1_5 = data %>% select(1,5)
head(data_1_5)
```

A tibble: 6 × 2

A5	A100
<dbl>	<dbl>
1946.5	1780.4
1963.5	1751.4
1934.3	1743.5
1934.8	1727.4
1939.9	1728.8
1925.9	1767.5

In [25]:

```
# or by name
data_1_5 = data %>% select(A5, "A100")
head(data_1_5)
```

A tibble: 6 × 2

A5	A100
<dbl>	<dbl>
1946.5	1780.4
1963.5	1751.4
1934.3	1743.5
1934.8	1727.4
1939.9	1728.8
1925.9	1767.5

Exclude data from the file.

In [26]:

```
# Exclude the first and fifth columns from the data. data frames and data storage. framework attempt
temp_data = data[,-c(1,5)]
head(temp_data)
```

A tibble: 6 × 6

B5	C5	D5	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2006.5	1881.8	1806.9	1654.2	1663.3	1668.4
1991.5	1890.4	1788.1	1663.1	1641.1	1641.9
1988.8	1865.7	1775.0	1633.3	1621.5	1620.0
1975.4	1880.7	1805.4	1642.2	1610.7	1685.8
1998.4	1861.1	1775.7	1656.7	1624.6	1610.5
2012.3	1887.3	1807.3	1664.4	1604.6	1670.6

In [27]:

```
# or using dplyr
temp_data = data %>% select(-1, -5)
head(temp_data)
```

A tibble: 6 × 6

B5	C5	D5	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2006.5	1881.8	1806.9	1654.2	1663.3	1668.4
1991.5	1890.4	1788.1	1663.1	1641.1	1641.9
1988.8	1865.7	1775.0	1633.3	1621.5	1620.0
1975.4	1880.7	1805.4	1642.2	1610.7	1685.8
1998.4	1861.1	1775.7	1656.7	1624.6	1610.5
2012.3	1887.3	1807.3	1664.4	1604.6	1670.6

In [28]:

```
# or by name
temp_data = data %>% select(-A5,-A100)
head(temp_data)
```

A tibble: 6 × 6

B5	C5	D5	B100	C100	D100
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2006.5	1881.8	1806.9	1654.2	1663.3	1668.4
1991.5	1890.4	1788.1	1663.1	1641.1	1641.9
1988.8	1865.7	1775.0	1633.3	1621.5	1620.0
1975.4	1880.7	1805.4	1642.2	1610.7	1685.8
1998.4	1861.1	1775.7	1656.7	1624.6	1610.5
2012.3	1887.3	1807.3	1664.4	1604.6	1670.6

Basic conversion of a simple data matrix into a standard data format - stack(...)

In [29]:

```
data5 = data[,1:4] # from the data we select those columns that correspond to measurements after 5 cycles
colnames(data5) = c("A","B","C","D") # Rename columns
head(data5)
```

A tibble: 6 × 4

A	B	C	D
<dbl>	<dbl>	<dbl>	<dbl>

A	B	C	D
<dbl>	<dbl>	<dbl>	<dbl>
1946.5	2006.5	1881.8	1806.9
1963.5	1991.5	1890.4	1788.1
1934.3	1988.8	1865.7	1775.0
1934.8	1975.4	1880.7	1805.4
1939.9	1998.4	1861.1	1775.7
1925.9	2012.3	1887.3	1807.3

In [30]:

```
data5S = stack(data5)         # and transfer to st. data format
colnames(data5S) = c("kap5","manufacturer") # and edit the column names once more
head(data5S)
```

A data.frame: 6 × 2

	kap5	manufacturer
	<dbl>	<fct>
1	1946.5	A
2	1963.5	A
3	1934.3	A
4	1934.8	A
5	1939.9	A
6	1925.9	A

In [31]:

```
# We do the same for measurements performed after 100 cycles
data100 = data[,5:8] # we select from the data those columns that correspond to measurements after 100 cycles
colnames(data100) = c("A","B","C","D") # Rename columns
data100S = stack(data100)         # and transfer to st. data format
colnames(data100S) = c("kap100","manufacturer") # and edit the column names once more
```

If we want standard data type with both measurements, we should use reshape function:

In [32]:

```
dataS=reshape(data=as.data.frame(data),
              direction="long", # means we are going from data matrix (wide format)
                                # into standard data format - long formant
              varying=list(c("A5","B5","C5","D5"), # list of vectors with values for each
                           c("A100","B100","C100","D100")), # resulting column
              v.names=c("cycles5","cycles100"), # name of columns in the result
              times=c("A","B","C","D"),  # values of sorting variable
              timevar="manufacturer")
head(dataS)
# you can use na.omit(dataS) to remove NaN values from data frame
```

A data.frame: 6 × 4

	manufacturer	cycles5	cycles100	id
	<chr>	<dbl>	<dbl>	<int>
1.A	A	1946.5	1780.4	1
2.A	A	1963.5	1751.4	2
3.A	A	1934.3	1743.5	3
4.A	A	1934.8	1727.4	4
5.A	A	1939.9	1728.8	5
6.A	A	1925.9	1767.5	6

!!! Handle the na.omit function extremely carefully so that you do not inadvertently lose data !!!

Defining new columns in a data frame

In [33]:

```
# Defining a new variable of the drop in the capacity
dataS$drop = dataS$cycles5 - dataS$cycles100
```

In [34]:

```
head(dataS)
```

A data.frame: 6 × 5

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
1.A	A	1946.5	1780.4	1	166.1
2.A	A	1963.5	1751.4	2	212.1

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
3.A	A	1934.3	1743.5	3	190.8
4.A	A	1934.8	1727.4	4	207.4
5.A	A	1939.9	1728.8	5	211.1
6.A	A	1925.9	1767.5	6	158.4

In [35]:

```
# or using a function from the dplyr package
dataS = dataS %>% mutate(drop=cycles5-cycles100)
```

Select data from standard data format

In [36]:

```
dataS$cycles5
```

1946.5 · 1963.5 · 1934.3 · 1934.8 · 1939.9 · 1925.9 · 2023 · 1952.5 · 1894.7 · 1944 · 1946.7 · 1903.7 · 1967.2 · 1949.9 · 1938.5 · 1986.4 · 1962.5 · 1931.7 · 1979 · 1944.3 · 1919.3 · 1966.3 · 1884.4 · 1934 · 1992.6 · 1996.4 · 1920.8 · 1951.2 · 1896.5 · 1933.8 · 1947.9 · 1952.6 · 1940.5 · 1973 · 1952 · 2015 · 1975.4 · 1955.7 · 1927.3 · 1921.2 · 1923 · 1951.1 · 1907.6 · 1930.4 · 1981.4 · 1943.2 · 1940 · 1904.5 · 1950.2 · 1929.6 · 1966.1 · 1940.3 · 1962.1 · 1954.7 · 1960.2 · 1976 · 1937.3 · 1965.1 · 1933.2 · 1900.8 · 1923.1 · 2003.3 · 1984.2 · 1985 · 1964.4 · 1899.4 · 1932.2 · 1966.1 · 1962.8 · 1915.1 · 1987 · 1951.3 · 1972.7 · 1955.4 · 1926.8 · 1963.4 · 1938.6 · 1975.5 · 1938.8 · 1972.5 · 2030 · 1955.5 · 1925.3 · 1939.2 · 1995.3 · 1931.1 · 1923.1 · 2003.3 · 1984.2 · 1985 · 1964.4 · 1899.4 · 1932.2 · 1966.1 · 1962.8 · <NA> · <NA> · <NA> · <NA> · <NA> · 2006.5 · 1991.5 · 1988.8 · 1975.4 · 1998.4 · 2012.3 · 1995.4 · 2011.6 · 2010.5 · 2002 · 1998 · 2012.9 · 2003.1 · 1999.3 · 2001.1 · 2003.7 · 1992.7 · 1979.5 · 2009.7 · 2003.1 · 1979 · 1997.2 · 2010.9 · 1984.7 · 2020.5 · 1992.3 · 2007.7 · 2009.9 · 1987.1 · 2001.3 · 1993.4 · 1974.1 · 1994.7 · 1994.7 · 2023.7 · 1982.8 · 2005.5 · 2013 · 1998.9 · 1995.5 · 2017.5 · 1987.3 · 2007.2 · 2030.9 · 2001.5 · 1992.2 · 2004.7 · 1997.4 · 2000.6 · 1996.5 · 2014.8 · 2011.8 · 2015.7 · 2015.1 · 2003.4 · 2014.9 · 2003.4 · 1985.6 · 2016.2 · 1981.8 · 2014 · 1996.8 · 1994 · 2005.3 · 2003.2 · 1998.6 · 2005.1 · 1994.8 · 1993.5 · 2003.7 · 1974.3 · 2001 · 1983.6 · 1999.2 · 2010.1 · 1995.3 · 1999.8 · 1995.9 · 1990.3 · 2018.3 · 2014.9 · 2006.5 · 1989.3 · 2002.5 · 2006 · 2010.3 · 1991.7 · 1996.9 · 1991.4 · 1991.3 · 2003.3 · 2009.5 · 2006.2 · 2007.5 · 2000.8 · 2001.1 · 2000.4 · 1998.6 · 2000.1 · 1993.6 · 1881.8 · 1890.4 · 1865.7 · 1880.7 · 1861.1 · 1887.3 · 1922 · 1926.1 · 1898.8 · 1887 · 1908.5 · 1890.7 · 1914.2 · 1899.4 · 1932 · 1862 · 1873.5 · 1878.5 · 1898.4 · 1903.2 · 1887.8 · 1884.7 · 1902 · 1929.4 · 1900.3 · 1872.8 · 1893.2 · 1929.9 · 1884.3 · 1901 · 1899 · 1885.8 · 1892.7 · 1893.3 · 1886.1 · 1892 · 1908 · 1904.4 · 1925.9 · 1924.2 · 1928.4 · 1920.9 · 1942.5 · 1928.8 · 1901.2 · 1905.5 · 1890.8 · 1923.4 · 1891.3 · 1927 · 1891.8 · 1904.7 · 1887.3 · 1887.4 · 1899.7 · 1910.5 · 1891.2 · 1882 · 1929.7 · 1911.9 · 1877.2 · 1874.1 · 1877.2 · 1901.7 · 1933.1 · 1848.4 · 1905.1 · 1901.4 · 1936.1 · 1889.3 · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · 1806.9 · 1788.1 · 1775 · 1805.4 · 1775.7 · 1807.3 · 1789.9 · 1846 · 1820.3 · 1872.2 · 1752.4 · 1830.5 · 1760.2 · 1793.4 · 1783.1 · 1846.7 · 1814.4 · 1832.2 · 1776.1 · 1823.7 · 1824 · 1810.4 · 1814.1 · 1753.3 · 1802.1 · 1768.1 · 1784.4 · 1843.2 · 1813.1 · 1789.1 · 1793.3 · 1819.9 · 1776.4 · 1808.8 · 1801.9 · 1789.4 · 1777.3 · 1778.4 · 1779.2 · 1839.9 · 1742.6 · 1766.3 · 1803 · 1769.2 · 1781.2 · 1788.2 · 1799.3 · 1823.5 · 1774.8 · 1818.2 · 1793.2 · 1810.6 · 1777.5 · 1817.6 · 1769.9 · 1748.5 · 1770.8 · 1832 · 1764.7 · 1764 · 1785.6 · 1822.9 · 1650.3 · 1820 · 1825.8 · 1869.8 · 1812.2 · 1792.3 · 1760.8 · 1732.6 · 1804.3 · 1828.4 · 1825.2 · 1865 · 1831.9 · 1770 · 1788.2 · 1853.1 · 1747.2 · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA>

In [37]:

```
# May be useful - create separate variables
a5 = dataS$cycles5[dataS$manufacturer=="A"] # Class(type) numeric
a5
```

1946.5 · 1963.5 · 1934.3 · 1934.8 · 1939.9 · 1925.9 · 2023 · 1952.5 · 1894.7 · 1944 · 1946.7 · 1903.7 · 1967.2 · 1949.9 · 1938.5 · 1986.4 · 1962.5 · 1931.7 · 1979 · 1944.3 · 1919.3 · 1966.3 · 1884.4 · 1934 · 1992.6 · 1996.4 · 1920.8 · 1951.2 · 1896.5 · 1933.8 · 1947.9 · 1952.6 · 1940.5 · 1973 · 1952 · 2015 · 1975.4 · 1955.7 · 1927.3 · 1921.2 · 1923 · 1951.1 · 1907.6 · 1930.4 · 1981.4 · 1943.2 · 1940 · 1904.5 · 1950.2 · 1929.6 · 1966.1 · 1940.3 · 1962.1 · 1954.7 · 1960.2 · 1976 · 1937.3 · 1965.1 · 1933.2 · 1900.8 · 1923.1 · 2003.3 · 1984.2 · 1985 · 1964.4 · 1899.4 · 1932.2 · 1966.1 · 1962.8 · 1915.1 · 1987 · 1951.3 · 1972.7 · 1955.4 · 1926.8 · 1963.4 · 1938.6 · 1975.5 · 1938.8 · 1972.5 · 2030 · 1955.5 · 1925.3 · 1939.2 · 1995.3 · 1931.1 · 1923.1 · 2003.3 · 1984.2 · 1985 · 1964.4 · 1899.4 · 1932.2 · 1966.1 · 1962.8 · <NA> · <NA> · <NA> · <NA> · <NA>

In [38]:

```
# using dplyr with a data frame result
a5.df = dataS %>%
  filter(manufacturer=="A") %>%  # filters rows corresponding to manufacturer A
  select(cycles5)    # Selects only the values in column kap5,
head(a5.df)
```

A data.frame: 6 × 1

	cycles5
	<dbl>
1.A	1946.5
2.A	1963.5
3.A	1934.3

	cycles5
	<dbl>
4.A	1934.8
5.A	1939.9
6.A	1925.9

More detailed window for Dplyr library functions - work on data in standard data format

It is necessary to apply to data in st. data format !!! Pipe operator %>% - helps with chaining functions - in the new RSstudio shortcut key Ctrl + Shift + M

filter - applies a filter to the given column

In [39]:

```
# filter - selects/filters rows based on given conditions
# Selection of products from the manufacturer
tmp = dataS %>% filter(manufacturer=="A")
head(tmp)
```

A data.frame: 6 × 5

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
1.A	A	1946.5	1780.4	1	166.1
2.A	A	1963.5	1751.4	2	212.1
3.A	A	1934.3	1743.5	3	190.8
4.A	A	1934.8	1727.4	4	207.4
5.A	A	1939.9	1728.8	5	211.1
6.A	A	1925.9	1767.5	6	158.4

In [40]:

```
# Selection of products from manufacturer A or B
# separating conditions correspond to the logical "or"
tmp = dataS %>% filter(manufacturer=="A" | manufacturer=="B")
head(tmp)
```

A data.frame: 6 × 5

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
1.A	A	1946.5	1780.4	1	166.1
2.A	A	1963.5	1751.4	2	212.1
3.A	A	1934.3	1743.5	3	190.8
4.A	A	1934.8	1727.4	4	207.4
5.A	A	1939.9	1728.8	5	211.1
6.A	A	1925.9	1767.5	6	158.4

In [41]:

```
# Selection of all products with a decrease of 200 mAh and more from the manufacturer C
# comma separating conditions corresponds to logical "and at the same time"
tmp = dataS %>% filter(drop>=200, manufacturer=="C")
head(tmp)
```

A data.frame: 6 × 5

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
1.C	C	1881.8	1663.3	1	218.5
2.C	C	1890.4	1641.1	2	249.3
3.C	C	1865.7	1621.5	3	244.2
4.C	C	1880.7	1610.7	4	270.0
5.C	C	1861.1	1624.6	5	236.5
6.C	C	1887.3	1604.6	6	282.7

mutate - produce a new column

In [42]:

```
# mutate - adds a new variable or transforms an existing one
# Creating a new column drop_Ah, which indicates the capacity drop in Ah(original data in mAh, 1 Ah=1000 mAh)
tmp = dataS %>% mutate(drop_Ah=drop/1000)
head(tmp)
# Attention! if we do not save the result with the new column, it will only be printed and disappear
```

A data.frame: 6 × 6

	manufacturer	cycles5	cycles100	id	drop	drop_Ah
	<chr>	<dbl>	<dbl>	<int>	<dbl>	<dbl>
1.A	A	1946.5	1780.4	1	166.1	0.1661
2.A	A	1963.5	1751.4	2	212.1	0.2121
3.A	A	1934.3	1743.5	3	190.8	0.1908
4.A	A	1934.8	1727.4	4	207.4	0.2074
5.A	A	1939.9	1728.8	5	211.1	0.2111
6.A	A	1925.9	1767.5	6	158.4	0.1584

summarize - generates summary characteristics of various variables

In [43]:

```
# Calculation of the mean and median of all values of the variable cycles5
dataS %>% summarise(average=mean(cycles5),median=median(cycles5))
```

A data.frame: 1 × 2

average	median
<dbl>	<dbl>
NA	NA

If the results contain NaNs, it means that the original data contained NaNs. There are two options, either drop NaNs from data, or set the function to ignore them. Be carefull with droping NaN values, you can loose data you want to keep. E.g. of you have data for capacity for 5 cycles, but not for 100, na.omit(...) will drop the whole line.

In [44]:

```
tmp = na.omit(dataS)
tmp %>% summarise(average=mean(cycles5),median=median(cycles5))
```

A data.frame: 1 × 2

average	median
<dbl>	<dbl>
1919.419	1932.65

In [45]:

```
dataS %>% summarise(average=mean(cycles5,na.rm = TRUE),median=median(cycles5,na.rm = TRUE))
```

A data.frame: 1 × 2

average	median
<dbl>	<dbl>
1919.419	1932.65

arrange - sorts rows according to the selected variable

In [46]:

```
# Ascending and descending order of rows according to the decrease value
tmp = dataS %>% arrange(drop)
head(tmp)
```

A data.frame: 6 × 5

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
63.D	D	1650.3	1659.7	63	-9.4
93.A	A	1932.2	1866.9	93	65.3
29.D	D	1813.1	1712.2	29	100.9
7.D	D	1789.9	1683.9	7	106.0
25.D	D	1802.1	1690.0	25	112.1
51.D	D	1793.2	1676.8	51	116.4

In [47]:

```
tmp = dataS %>% arrange(desc(drop))
head(tmp)
```

A data.frame: 6 × 5

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
91.B	B	2003.3	1620.9	91	382.4
92.B	B	2009.5	1630.7	92	378.8
55.B	B	2003.4	1625.0	55	378.4

	manufacturer	cycles5	cycles100	id	drop
	<chr>	<dbl>	<dbl>	<int>	<dbl>
66.B	B	1998.6	1620.9	66	377.7
67.B	B	2005.1	1630.7	67	374.4
14.B	B	1999.3	1625.5	14	373.8

group_by - groups values into groups according to the selected variable

In [48]:

```
# the table is "virtually" divided into groups for later processing, eg summarize
head(dataS %>% group_by(manufacturer))
```

A grouped_df: 6 × 5

manufacturer	cycles5	cycles100	id	drop
<chr>	<dbl>	<dbl>	<int>	<dbl>
A	1946.5	1780.4	1	166.1
A	1963.5	1751.4	2	212.1
A	1934.3	1743.5	3	190.8
A	1934.8	1727.4	4	207.4
A	1939.9	1728.8	5	211.1
A	1925.9	1767.5	6	158.4

In [49]:

```
# Ideal for calculating summary characteristics for each manufacturer separately, eg average
dataS %>%
  group_by(manufacturer) %>%
  summarise(average=mean(cycles5,na.rm = TRUE), "st.dev."=sd(cycles5,na.rm = TRUE))
```

A tibble: 4 × 3

manufacturer	average	st.dev.
<chr>	<dbl>	<dbl>
A	1950.486	29.23079
B	2000.596	10.92109
C	1899.396	20.23397
D	1797.044	34.72579

Final note on dplyr(which is good to finish until the end...)

Some operations may throw a "tibble" object. This is a more modern data.frame, however it can cause problems and cause error messages in some functions! You can easily convert this "tibble" object to data.frame using as.data.frame().

5. Data conversion to the standard data format (for the two most common data formats)

From data in Data Matrix format (already seen before)

In [50]:

```
data_DM = read_excel("./data/datova_matice.xlsx")
head(data_DM)
```

```
New names:
* `` -> ...1
```

A tibble: 6 × 9

...1	Amber Světelný tok při teplotě 22 °C (lm)	Amber Světelný tok při teplotě 5 °C (lm)	Bright Světelný tok při teplotě 22 °C (lm)	Bright Světelný tok při teplotě 5 °C (lm)	Clear Světelný tok při teplotě 22 °C (lm)	Clear Světelný tok při teplotě 5 °C (lm)	Dim Světelný tok při teplotě 22 °C (lm)	Dim Světelný tok při teplotě 5 °C (lm)
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0	784.9	786.6	849.7	850.9	822.4	823.4	773.8	772.6
1	782.0	783.8	831.9	833.2	783.5	787.2	809.3	812.9
2	782.6	785.3	828.5	825.0	790.0	787.7	826.0	830.0
3	777.7	772.2	795.5	790.5	792.7	795.3	778.1	781.9
4	824.7	825.3	815.2	817.9	829.4	831.5	777.3	772.4
5	759.1	759.1	804.9	801.1	799.0	798.7	797.6	795.7

In [51]:

```
data_DM = data_DM[,-1]
colnames(data_DM) = c("A22", "A5", "B22", "B5", "C22", "C5", "D22", "D5")
```

```
head(data_DM)
```

A tibble: 6 × 8

A22	A5	B22	B5	C22	C5	D22	D5
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
784.9	786.6	849.7	850.9	822.4	823.4	773.8	772.6
782.0	783.8	831.9	833.2	783.5	787.2	809.3	812.9
782.6	785.3	828.5	825.0	790.0	787.7	826.0	830.0
777.7	772.2	795.5	790.5	792.7	795.3	778.1	781.9
824.7	825.3	815.2	817.9	829.4	831.5	777.3	772.4
759.1	759.1	804.9	801.1	799.0	798.7	797.6	795.7

Reshape function

Its parameters:

- **data** - data to be converted must be fe format data.frame(as.data.frame(data))
- **direction** - which direction we want to transform
  - "long" - to standard format
  - "wide" - back to the data matrix
- **varying** - column names that indicate the same data for different categories
  - it is a sheet of vectors
  - each sheet item is one measurement
  - each vector is then a list of columns
- **v.names** - column names in st. give. format
  - The number of names must match the number of vectors in varying
- **times** - names of individual categories
  - ATTENTION !! must be in the same order as the varying variable
- **timevar** - column name with categories

In [52]:

```
data_DM_S=reshape(data=as.data.frame(data_DM),
                  direction="long",
                  varying=list(c("A5", "B5", "C5", "D5"),
                               c("A22","B22","C22","D22")),
                  v.names=c("5 C","22  C"),
                  times=c("Amber","Bright","Clear","Dim"),
                  timevar="vyrobce")
head(data_DM_S)
```

A data.frame: 6 × 4

	vyrobce	5 C	22 C	id
	<chr>	<dbl>	<dbl>	<int>
1.Amber	Amber	786.6	784.9	1
2.Amber	Amber	783.8	782.0	2
3.Amber	Amber	785.3	782.6	3
4.Amber	Amber	772.2	777.7	4
5.Amber	Amber	825.3	824.7	5
6.Amber	Amber	759.1	759.1	6

In [53]:

```
# and if we want, we can convert the data back
data_DM_2=reshape(data=data_DM_S,
                  direction="wide",
                  varying=list(c("A5", "B5", "C5", "D5"),
                               c("A22","B22","C22","D22")),
                  v.names=c("5 C","22  C"),
                  times=c("Amber","Bright","Clear","Dim"),
                  timevar="vyrobce")
head(data_DM_2)
```

A data.frame: 6 × 9

	id	A5	A22	B5	B22	C5	C22	D5	D22
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1.Amber	1	786.6	784.9	850.9	849.7	823.4	822.4	772.6	773.8
2.Amber	2	783.8	782.0	833.2	831.9	787.2	783.5	812.9	809.3

	id	A5	A22	B5	B22	C5	C22	D5	D22
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
3.Amber	3	785.3	782.6	825.0	828.5	787.7	790.0	830.0	826.0
4.Amber	4	772.2	777.7	790.5	795.5	795.3	792.7	781.9	778.1
5.Amber	5	825.3	824.7	817.9	815.2	831.5	829.4	772.4	777.3
6.Amber	6	759.1	759.1	801.1	804.9	798.7	799.0	795.7	797.6

From a data file where the categories are in individual Excel sheets

In [54]:

```
data_A = read_excel("./data/po_listech.xlsx", sheet=1)
head(data_A)
data_B = read_excel("./data/po_listech.xlsx", sheet=2)
data_C = read_excel("./data/po_listech.xlsx", sheet=3)
data_D = read_excel("./data/po_listech.xlsx", sheet=4)
```

```
New names:
* `` -> ...1
```

A tibble: 6 × 3

...1	Světelný tok při teplotě 22 °C (lm)	Světelný tok při teplotě 5 °C (lm)
<dbl>	<dbl>	<dbl>
0	825.2	828.9
1	855.4	847.4
2	823.3	813.3
3	826.1	815.2
4	785.5	781.1
5	835.0	828.3

```
New names:
* `` -> ...1

New names:
* `` -> ...1

New names:
* `` -> ...1
```

In [55]:

```
data_A$vyrobce = "Amber"
data_B$vyrobce = "Bright"
data_C$vyrobce = "Clear"
data_D$vyrobce = "Dim"
head(data_A)
```

A tibble: 6 × 4

...1	Světelný tok při teplotě 22 °C (lm)	Světelný tok při teplotě 5 °C (lm)	vyrobce
<dbl>	<dbl>	<dbl>	<chr>
0	825.2	828.9	Amber
1	855.4	847.4	Amber
2	823.3	813.3	Amber
3	826.1	815.2	Amber
4	785.5	781.1	Amber
5	835.0	828.3	Amber

In [56]:

```
data_PL_S = rbind(data_A, data_B, data_C, data_D)
head(data_PL_S)
```

A tibble: 6 × 4

...1	Světelný tok při teplotě 22 °C (lm)	Světelný tok při teplotě 5 °C (lm)	vyrobce
<dbl>	<dbl>	<dbl>	<chr>
0	825.2	828.9	Amber
1	855.4	847.4	Amber
2	823.3	813.3	Amber
3	826.1	815.2	Amber
4	785.5	781.1	Amber

...1	Světelný tok při teplotě 22 °C (lm)	Světelný tok při teplotě 5 °C (lm)	vyrobce
<dbl>	<dbl>	<dbl>	<chr>
5	835.0	828.3	Amber

6. Exploratory analysis and visualization of a categorical variable

Notes on graphics in R

the basis are the so-called high-level functions, which create a graph(ie open the graphics window and draw according to the specified parameters) followed by the so-called low-level functions, which add something to the active graphics window, do not open new low-level functions - eg abline, points, lines, legend, title, axis... which add a line, points, legend... ie. before using the "low-level" function it is necessary to call the "high-level" function(eg plot, boxplot, hist, barplot, pie,...) Further graphic parameters can be found in the help or eg here http://www.statmethods.net/advgraphs/parameters.html or here https://flowingdata.com/2015/03/17/r-cheat-sheet-for-graphical-parameters/or http://bcb.dfci.harvard.edu/~aedin/courses/BiocDec2011/2.Plotting.pdf Colors in R http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf Saving graphs is possible using the function dev.print, jpeg, pdf and others. More easily in the Plots ->Export window

In [57]:
```
# Table of absolute frequencies of the manufacturer's categorical variable...
freq = table(dataS$manufacturer)
freq # listing - object of type "table" - mostly more suitable, but more difficult conversion to type data.frame
```

```
  A   B   C   D
100 100 100 100
```

Looks weird, we should remember, that we had NaNs and we converted it from data matrix, thats why we got same numbers!

In [58]:
```
tmp = na.omit(dataS)
freq = table(tmp$manufacturer)
freq
```

```
  A   B   C   D
 95 100  70  79
```

In [59]:
```
# .. and using dplyr functions(more complex)
freq_df = tmp %>%  group_by(manufacturer) %>%
                  summarise(freq = n())  # number of products for each manufacturer
freq_df # listing - object type "tibble" - useful when we need to simply convert to type data.frame
```

A tibble: 4 × 2

manufacturer	freq
<chr>	<int>
A	95
B	100
C	70
D	79

Relative frequency table

In [60]:
```
# By direct calculation
rel.freq=100*freq/sum(freq)
rel.freq
```

```
       A        B        C        D
27.61628 29.06977 20.34884 22.96512
```

In [61]:
```
# or using the prop.table function
rel.freq=prop.table(freq)*100
rel.freq # statement
```

```
       A        B        C        D
27.61628 29.06977 20.34884 22.96512
```

In [62]:
```
# or using the dplyr functions, where absolute frequencies will also be included
freq_all = tmp %>%  group_by(manufacturer) %>%
                    summarise(freq = n()) %>%
                    mutate(rel_freq = 100*(freq/sum(freq)))
freq_all
t(freq_all) # maybe more elegant in transpose form
```

A tibble: 4 × 3

manufacturer	freq	rel_freq

manufacturer	freq	rel_freq
<chr>	<int>	<dbl>
A	95	27.61628
B	100	29.06977
C	70	20.34884
D	79	22.96512

A matrix: 3 × 4 of type chr

manufacturer	A	B	C	D
freq	95	100	70	79
rel_freq	27.61628	29.06977	20.34884	22.96512

In [63]:

```
# For relative frequencies tables, rounding must be included,
# and summation to 1 (or 100 in case of %) kept
rel.freq=round(rel.freq,digits=1) # rounded to 1 decimal place
rel.freq[4]=100-sum(rel.freq[1:3]) # rounding error monitoring
rel.freq
```

```
   A    B    C    D 
27.6 29.1 20.3 23.0 
```

In [64]:

```
# The procedure for table_abs_rel is different due to a different format(tibble)
freq_all[1:4,3] = round(freq_all[1:4,3],digits=1) # rounded to 1 decimal place
freq_all[4,3] = 100-sum((freq_all[1:3,3]))
freq_all
```

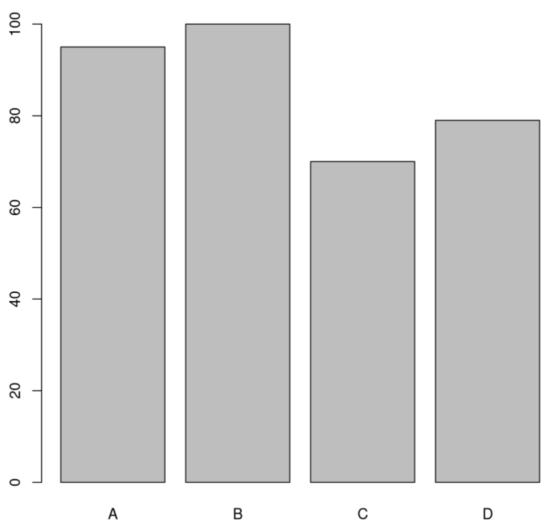
A tibble: 4 × 3

manufacturer	freq	rel_freq
<chr>	<int>	<dbl>
A	95	27.6
B	100	29.1
C	70	20.3
D	79	23.0

Visualization using graphs

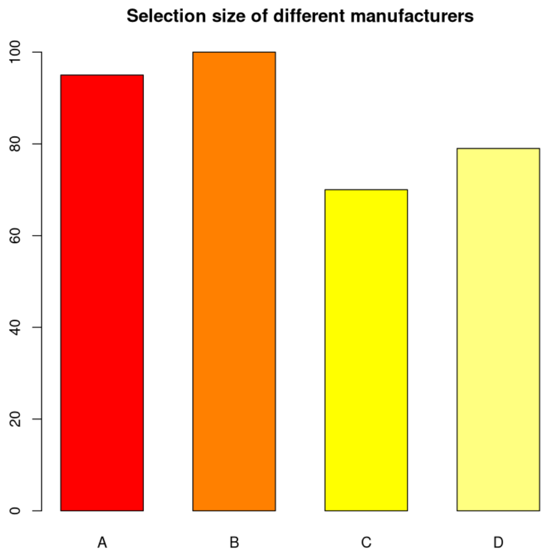
In [65]:

```
# Bar graph
# The basic R functionality (i.e. no package required) bar graph is based on the frequency table we have prepared
barplot(freq)
```



In [66]:

```
# Change colors, add name
barplot(freq,
        col=heat.colors(4), # alt. a vector of specific colors can be chosen, eg c("blue", "yellow," red "," green ")
        # or other scales(heat.colors, topo.colors, terrain.colors and many others)
        main="Selection size of different manufacturers",
        space=0.6) # The space parameter creates a space between columns
```



7. Exploratory analysis and visualization of a quantitative variable

In [67]:

```
# Descriptive statistics
summary(dataS$cycles5)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
   1650    1876    1933    1919    1993    2031      56 
```

In [68]:

```
# Beware of missing values
# Calculation of the average of one variable
mean(dataS$cycles5, na.rm = TRUE)
```

1919.41860465116

In [69]:

```
# Calculation of the median of one variable
quantile(dataS$cycles5, probs=0.5, na.rm = TRUE)
```

50%: 1932.65

In [70]:

```
# Range determination
length(dataS$cycles5)
```

400

In [71]:

```
# beware NaNs
length(na.omit(dataS$cycles5))
```

344

Other characteristics ->var(), sd(), min(), max(),...

Attention! The functions for calculating skewness and kurtosis are not part of the basic R, you will find them in the package moments. sharpness in the interval(1,5) To standardize the sharpness, it is necessary to subtract 3 from the calculated value. If you write the package name and "::" before the function name, you will ensure that the function from the given package will be used. packages have different functions under the same name

In [72]:

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

In [73]:

```
library(moments)
```

In [74]:

```
skewness(a5,na.rm = TRUE)
```

0.225070152973696

In [75]:

```
kurtosis(a5,na.rm = TRUE)-3
```

-0.00792805262342489

In [76]:

```
# If we want to calculate the given characteristic for variable capacity after 5 cycles
```

```
# according to the manufacturers, we can use the tapply function
tapply(dataS$cycles5, dataS$manufacturer, mean, na.rm=TRUE)
```

A: 1950.48631578947 **B:** 2000.596 **C:** 1899.39571428571 **D:** 1797.04430379747

In [77]:

```
# or using dplyr - here pay attention to automatic(not always correct rounding)
dataS %>%
  group_by(manufacturer) %>%
  summarise(mean(cycles5,na.rm=TRUE))
```

A tibble: 4 × 2

manufacturer	mean(cycles5, na.rm = TRUE)
<chr>	<dbl>
A	1950.486
B	2000.596
C	1899.396
D	1797.044

In [78]:

```
# To simplify the work, we can use the dplyr function and put all the characteristics in one table
dataS %>%    # without using group_by for the whole kap5 variable
  summarise(size=length(na.omit(cycles5)),
            min=min(cycles5,na.rm=TRUE),     # preventive na.rm=T
            Q1=quantile(cycles5,0.25,na.rm=TRUE),
            average=mean(cycles5,na.rm=TRUE),
            median=median(cycles5,na.rm=TRUE),
            Q3=quantile(cycles5,0.75,na.rm=TRUE),
            max=max(cycles5,na.rm=TRUE),
            variance=var(cycles5,na.rm=TRUE),
            st.dev.=sd(cycles5,na.rm=TRUE),
            variation_coeff=(100*(st.dev./average)),  # coefficient of variation in percent
            skewness=(moments::skewness(cycles5,na.rm=TRUE)),       # moments package precaution
            kurtosis=(moments::kurtosis(cycles5,na.rm=TRUE)-3))
```

A data.frame: 1 × 12

size	min	Q1	average	median	Q3	max	variance	st.dev.	variation_coeff	skewness	kurtosis
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
344	1650.3	1876.425	1919.419	1932.65	1992.625	2030.9	6344.696	79.6536	4.149882	-0.6735159	-0.5551027

In [79]:

```
# Don't forget to round correctly!
# We use group_by and get the characteristics for the capacity after 5 cycles according to the manufacturers
result = dataS %>%
            group_by(manufacturer) %>%
            summarise(size=length(na.omit(cycles5)),
            min=min(cycles5,na.rm=TRUE),     # preventive na.rm=T
            Q1=quantile(cycles5,0.25,na.rm=TRUE),
            average=mean(cycles5,na.rm=TRUE),
            median=median(cycles5,na.rm=TRUE),
            Q3=quantile(cycles5,0.75,na.rm=TRUE),
            max=max(cycles5,na.rm=TRUE),
            variance=var(cycles5,na.rm=TRUE),
            st.dev.=sd(cycles5,na.rm=TRUE),
            variation_coeff=(100*(st.dev./average)),  # coefficient of variation in percent
            skewness=(moments::skewness(cycles5,na.rm=TRUE)),       # moments package precaution
            kurtosis=(moments::kurtosis(cycles5,na.rm=TRUE)-3))
```

In [80]:

```
t(result) # more favourable looks as transposed
```

A matrix: 13 × 4 of type chr

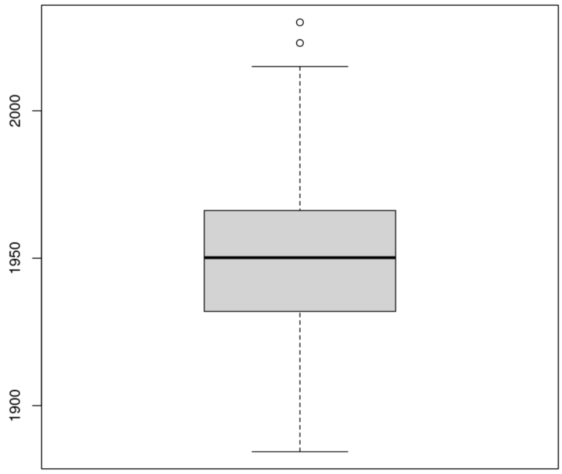
manufacturer	A	B	C	D
size	95	100	70	79
min	1884.4	1974.1	1848.4	1650.3
Q1	1931.950	1993.900	1887.075	1775.900
average	1950.486	2000.596	1899.396	1797.044
median	1950.2	2000.9	1898.9	1793.4
Q3	1966.20	2007.55	1911.55	1820.15
max	2030.0	2030.9	1942.5	1872.2
variance	854.4389	119.2703	409.4135	1205.8804
st.dev.	29.23079	10.92109	20.23397	34.72579
variation_coeff	1.498641	0.545892	1.065284	1.932384
skewness	0.22507015	-0.13191305	0.08402198	-0.70087863

kurtosis -0.007928053 0.103477086 -0.425052294 2.741260383

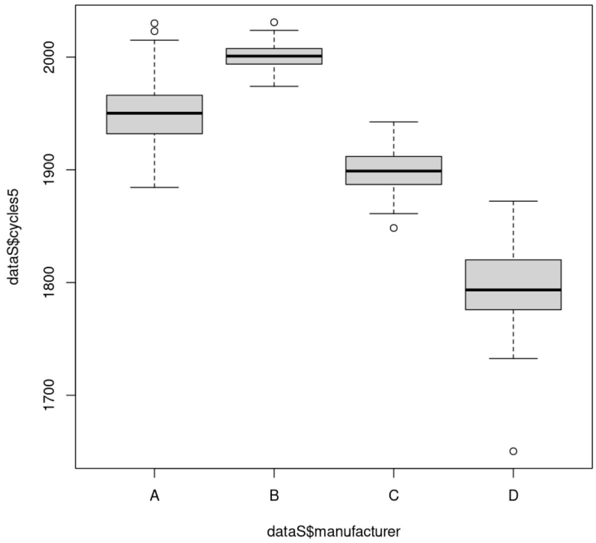
Box chart

We always plot for the original data and observe the outliers.

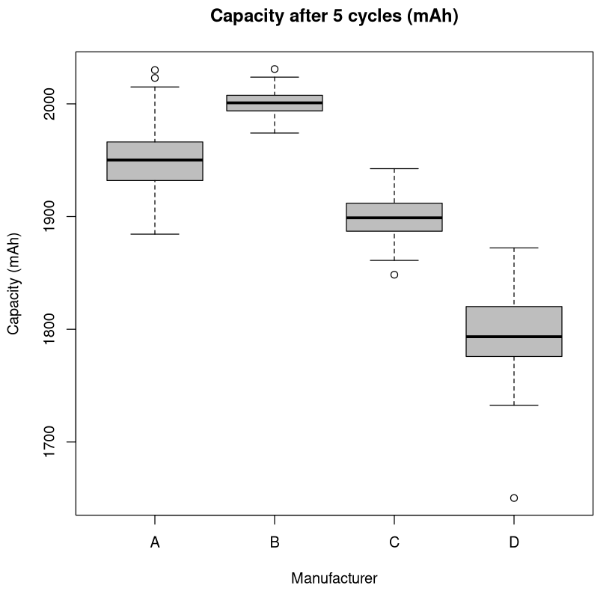
In [81]:
```
# Simple and fast rendering using the basic function only for manufacturer A and 5 cycles
boxplot(a5)
```



In [82]:
```
# And draw a multiple box graph
boxplot(dataS$cycles5~dataS$manufacturer) # graphic parameters can be set similarly to the previous ones
```



In [83]:
```
# Further modification of the graph, use of the points function to display the average
boxplot(dataS$cycles5~dataS$manufacturer,
        main="Capacity after 5 cycles (mAh)",
        xlab="Manufacturer",
        ylab="Capacity (mAh)",
        col="grey")
```



Removing outliers

In [84]:
```
outliers_cycles5 =
  dataS %>%
  group_by(manufacturer) %>%
  identify_outliers(cycles5)
outliers_cycles5
```

A tibble: 5 × 7

manufacturer	cycles5	cycles100	id	drop	is.outlier	is.extreme
<chr>	<dbl>	<dbl>	<int>	<dbl>	<lgl>	<lgl>
A	2023.0	1838.7	7	184.3	TRUE	FALSE
A	2030.0	1783.8	81	246.2	TRUE	FALSE
B	2030.9	1678.2	44	352.7	TRUE	FALSE
C	1848.4	1593.7	66	254.7	TRUE	FALSE
D	1650.3	1659.7	63	-9.4	TRUE	FALSE

Important!! - we need a column with unique indentifier - if we dont have it we can add it. By default during e.g. reshape it is added.

In [85]:
```
dataS$id2 = 1:length(dataS$manufacturer)
head(dataS)
```

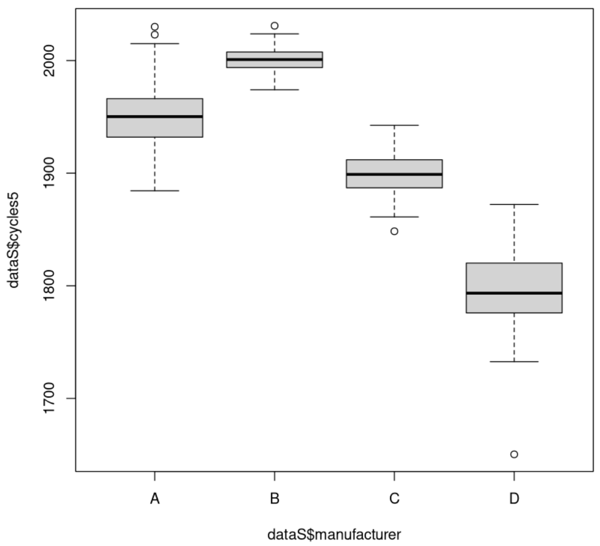
A data.frame: 6 × 6

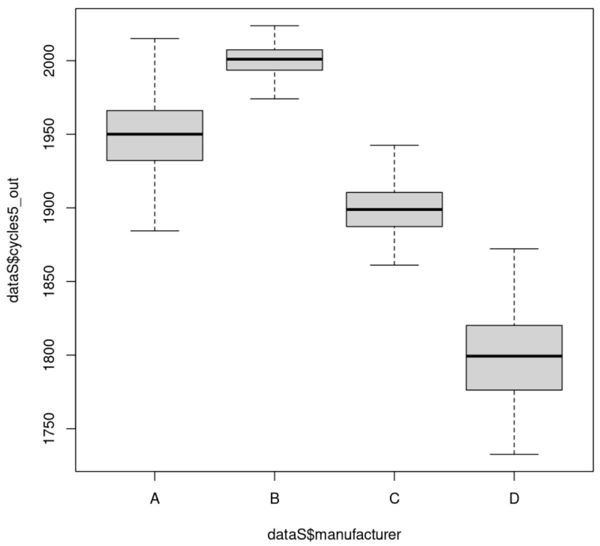
	manufacturer	cycles5	cycles100	id	drop	id2
	<chr>	<dbl>	<dbl>	<int>	<dbl>	<int>
1.A	A	1946.5	1780.4	1	166.1	1
2.A	A	1963.5	1751.4	2	212.1	2
3.A	A	1934.3	1743.5	3	190.8	3
4.A	A	1934.8	1727.4	4	207.4	4
5.A	A	1939.9	1728.8	5	211.1	5
6.A	A	1925.9	1767.5	6	158.4	6

Now we use the id column for creating new data column free of outliers

In [86]:
```
dataS$cycles5_out = ifelse(dataS$id %in% outliers_cycles5$id,NA,dataS$cycles5)
```

In [87]:
```
# compare
boxplot(dataS$cycles5~dataS$manufacturer)
boxplot(dataS$cycles5_out~dataS$manufacturer)
```





Histogram

We always plot for data without outliers !!

In [88]:

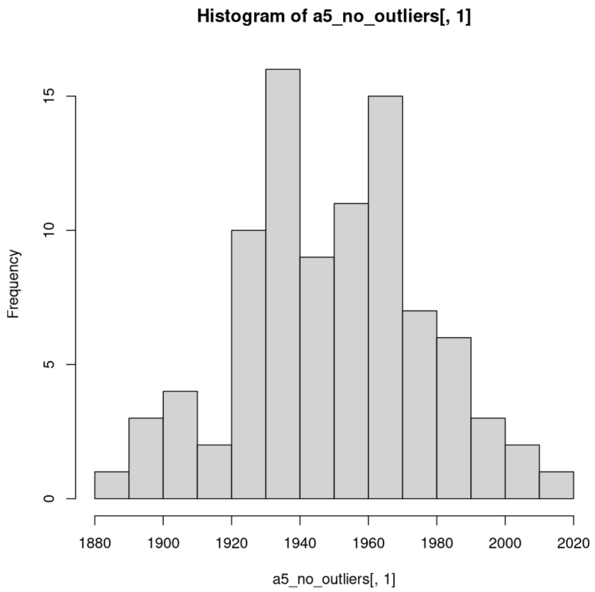
```
a5_no_outliers = dataS %>% filter(manufacturer=="A") %>% select(cycles5_out)
head(a5_no_outliers)
```

A data.frame: 6 × 1

	cycles5_out
	<dbl>
1.A	1946.5
2.A	1963.5
3.A	1934.3
4.A	1934.8
5.A	1939.9
6.A	1925.9

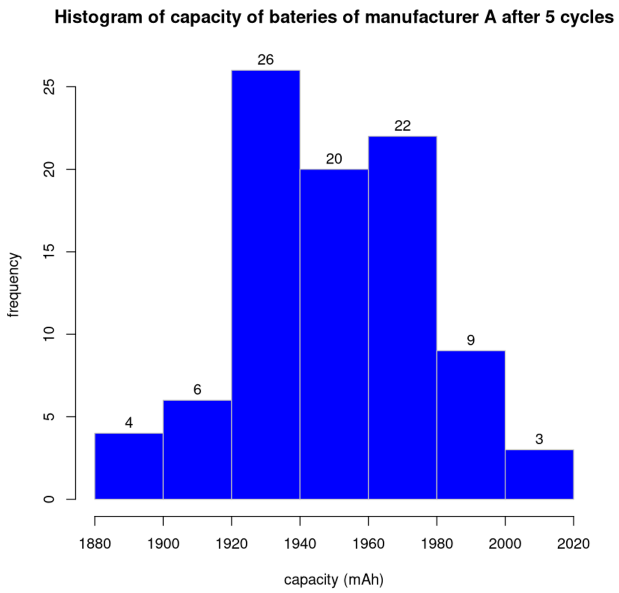
In [89]:

```
# hist does not like input as data frame, we can cheat it by selecting all of its values
hist(a5_no_outliers[,1], breaks=10)
```



In [90]:

```
# Labels, colors and other parameters can be set traditionally
hist(a5_no_outliers[,1],
     main="Histogram of capacity of bateries of manufacturer A after 5 cycles",
     xlab="capacity (mAh)",
     ylab="frequency",
     col="blue",       # fill color
     border="grey",    # column border color
     labels=TRUE)         # adds the absolute frequencies of the given categories in the form of labels
```

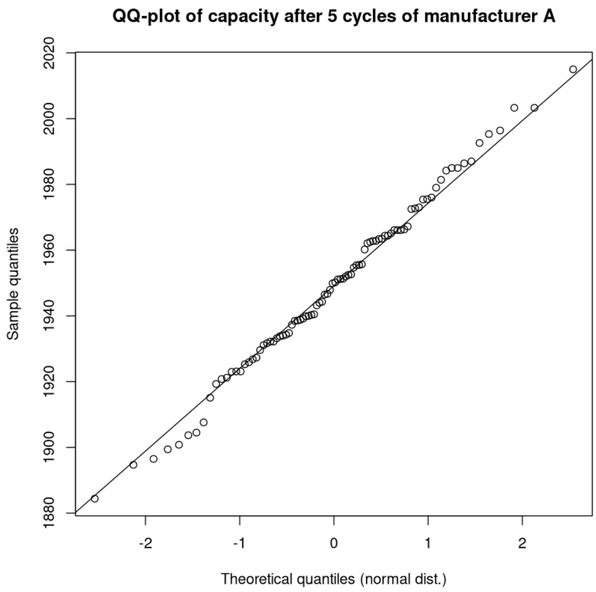


QQ-graph

We always plot for data without remote observations !!

In [91]:

```
# .. with adjustment of axis labels...
qqnorm(a5_no_outliers[,1],
       xlab="Theoretical quantiles (normal dist.)",
       ylab="Sample quantiles",
       main="QQ-plot of capacity after 5 cycles of manufacturer A")
qqline(a5_no_outliers[,1])
```



8. rule 3 σ and Chebyshev's inequality

Empirical verification of normality

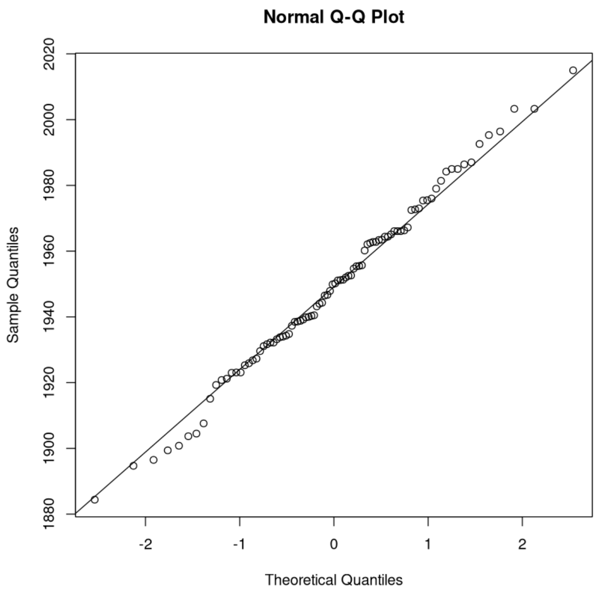
Based on data after deleting outliers:

In [92]:
```
# we will use the data from the removal example op
a5_no_outliers_cleared = na.omit(a5_no_outliers)[,1]
a5_no_outliers_cleared
```

1946.5 · 1963.5 · 1934.3 · 1934.8 · 1939.9 · 1925.9 · 1952.5 · 1894.7 · 1944 · 1946.7 · 1903.7 · 1967.2 · 1949.9 · 1938.5 · 1986.4 · 1962.5 · 1931.7 · 1979 · 1944.3 · 1919.3 · 1966.3 · 1884.4 · 1934 · 1992.6 · 1996.4 · 1920.8 · 1951.2 · 1896.5 · 1933.8 · 1947.9 · 1952.6 · 1940.5 · 1973 · 1952 · 2015 · 1975.4 · 1955.7 · 1927.3 · 1921.2 · 1923 · 1951.1 · 1907.6 · 1981.4 · 1943.2 · 1940 · 1904.5 · 1950.2 · 1929.6 · 1966.1 · 1940.3 · 1962.1 · 1954.7 · 1960.2 · 1976 · 1937.3 · 1965.1 · 1933.2 · 1900.8 · 1923.1 · 2003.3 · 1985 · 1964.4 · 1932.2 · 1966.1 · 1962.8 · 1915.1 · 1987 · 1951.3 · 1972.7 · 1955.4 · 1926.8 · 1963.4 · 1938.6 · 1975.5 · 1938.8 · 1972.5 · 1955.5 · 1925.3 · 1939.2 · 1995.3 · 1931.1 · 1923.1 · 2003.3 · 1984.2 · 1985 · 1964.4 · 1899.4 · 1932.2 · 1966.1 · 1962.8

We plot the QQ graph and calculate the skewness and sharpness:

In [93]:
```
qqnorm(a5_no_outliers_cleared)
qqline(a5_no_outliers_cleared)
```



In [94]:
```
skewness(a5_no_outliers_cleared)
kurtosis(a5_no_outliers_cleared) - 3 # another definition shifted by 3
```

-0.00522182449013756

-0.255921697074379

- the dots in the QQ graph must lie approximately on the line - ie. the quantiles correspond approximately to the quantiles of the normal distribution
- skewness must lie in the interval <-2, 2>
- kurtosis must lie in the interval <-2.2>
  - be careful we have to reduce the result of the R function by 3

If data normality is met -> rule 3σ

σ: $P(\mu - \sigma<X<\mu + \sigma)=0.6827$
2σ: $P(\mu - 2\sigma<X<\mu + 2\sigma)=0.9545$
3σ: $P(\mu - 3\sigma<X<\mu + 3\sigma)=0.9973$

If data normality is not met -> Chebyshev inequality

σ: $P(\mu - \sigma<X<\mu + \sigma)=0$
2σ: $P(\mu - 2\sigma<X<\mu + 2\sigma)=0.75$
3σ: $P(\mu - 3\sigma<X<\mu + 3\sigma)=0.8889$

In [95]:
```
mu = mean(a5_no_outliers_cleared)
sigma = sd(a5_no_outliers_cleared)
paste0("<", mu - sigma, ", ", mu + sigma, ">")
paste0("<", mu - 2*sigma, ", ", mu + 2*sigma, ">")
paste0("<", mu - 3*sigma, ", ", mu + 3*sigma, ">")
```

'<1922.3050697014, 1976.12159696526>'
'<1895.39680606948, 2003.02986059719>'
'<1868.48854243755, 2029.93812422912>'

9. Rounding

Most important:

- the standard deviation is rounded up to the prescribed number of digits(ceiling)
- data file size = <2,10> -> 1 valid digit
- data file size = (10,30> -> 2 valid digits
- data file size = (30,2000> -> 3 valid digits
- position measures(averages, quantiles,...) are then rounded (classically) to the same valid digit as the standard deviation

In [96]:
```
length(a5_no_outliers_cleared)
stdev = sd(a5_no_outliers_cleared)
stdev
```

90
26.908263631929

In [97]:
```
average = mean(a5_no_outliers_cleared)
average
```

1949.21333333333

In [98]:
```
max(a5_no_outliers_cleared)
```

2015