Introduction to R for Basic Statistics

Alessandra Meddis

Section of Biostatistics, University of Copenhagen

4-6 September , 2023

Learning Objectives

- Use R through the interface R studio
- Import/load data into R
- Implement basic calculation in R
- Manipulate data in R
- Create descriptive analysis in R
- Create simple plots in R

Structure of the course

- Today:
  - Download/Install R
  - Rstudio interface
  - What can R do?
  - Data structures in R
  - Data manipulation with R (Part I)
  - Descriptive analysis in R
- Wednesday:
  - Data manipulation in R (Part II)
  - Basic graphics in R

This is the first time that the course is running.
There might be last minute variation to the material.
Any feedback is much appreciated.

About me

What I do with R

- Write R code to test/develop methods
- Data analysis, visualisation and reporting for applied projects

What I like about R

- open source
- flexible and dynamic
- lot of support by statisticians

What I expect from this course

- lot of questions
- there is no right way to code something, be creative!
- error and warning are our friends

Why R?

AM: *Why is* R *a good statistical software?*
ChatGPT:

1. Open-source: a large and active community of users who continuously improve and expand its functionality.

2. Extensive package ecosystem: These packages provide specialized functions and tools for various statistical analyses, data manipulation, visualization, and machine learning.... allows users to customize their analyses and address specific research questions efficiently.

3. Reproducibility and collaboration: R promotes reproducible research by allowing users to document their analyses using literate programming techniques.

4. Community support: R has a *vibrant* and *active* community of users, comprising statisticians, data scientists, researchers, and developers. This community provides extensive support through online forums, mailing lists, and dedicated websites.

Why not R?

AM: *Why not R ?*
ChatGPT:

1. <u>Steep learning curve:</u>: The syntax and concepts used in R may be unfamiliar and require time and effort to master.

2. <u>Performance limitations:</u> R is an interpreted language, which can lead to slower execution times compared to compiled languages like C++ or Java.

3. <u>Team compatibility and collaboration:</u> if you are working in a team where the majority of members are already proficient in another statistical software or programming language, it may be more beneficial to stick with that choice.

Install R

1. Go to the link https://cran.r-project.org/
2. Click on *Download R for* (Windows/Mac/Linux) depending on your operation system
   - For Windows users: Install R for the first time → Download R-4.x.x for Windows
   - for Mac users: choose the latest release that supports the version of your operation system
   - For Linux users: choose the link corresponding to your distribution of Linux and follow instructions

Install Rstudio

1. Go to the link https://posit.co/download/rstudio-desktop/download
2. Click the appropriate link under Installers that correspond to your operation system and version

Rstudio Interface

- **Enviroment:** list of all loaded/initialized objects in R during the current session
- **Console:** where the code is compiled and output is printed
- **Files:** you can navigate through your files and open them
- **Plots:** visualize produced plots
- **Packages:** installed/loaded packages in the current session
- **Help:** Vignettes for support on functions

Rstudio Interface

File → New File → R script
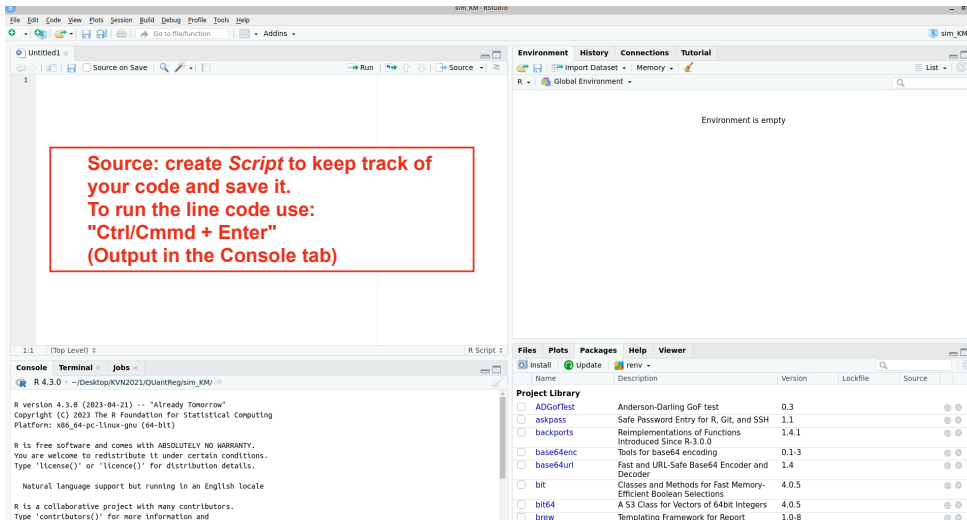
Set the Working Directory

When working on a project it is important to create a new folder where to save all relevant files: codes scripts, Data, reports, etc...
We can set the R Working Directory to that folder so to save everything directly there.

How?

- check the current working directory by the Command getwd() on the Console tab
- *Session → Set Working Directory → Choose Directory*
- Use the Command setwd (set working directory) on the Console Tab:

        setwd("~/Desktop/KVN2023/Course/IntrotoR")

What can ℝ do?

R as a calculator:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
...	...
sqrt()	Square root
exp()	Exponential
abs()	Aboslute value
log(, base=b)	Logarithm (base b)

R follows the standard ordering of operations: exponents and roots, then multiplication and divison, then addition and substraction.
We can use parentheses to change the order.

Functions in R

We can use pre-coded functions in R for calculations, create plots, implement some statistical tools.

Any function in R is used by writing the name of the function and *passing* the needed *arguments*. We can check them with the help command ?NameofFunction or help(NameofFunction) on the Console

```
> help(round)
```

Functions in R

Rounding of Numbers

Description

`ceiling` takes a single numeric argument x and returns a numeric vector containing the smallest integers not less than the corresponding elements of x.

`floor` takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x.

`trunc` takes a single numeric argument x and returns a numeric vector containing the integers formed by truncating the values in x toward 0.

`round` rounds the values in its first argument to the specified number of decimal places (default 0). See 'Details' about "round to even" when rounding off a 5.

`signif` rounds the values in its first argument to the specified number of *significant* digits. Hence, for numeric x, `signif(x, dig)` is the same as `round(x, dig - ceiling(log10(abs(x))))`. For complex x, this is not the case, see the 'Details'.

Usage

```
ceiling(x)
floor(x)
trunc(x, ...)

round(x, digits = 0)
signif(x, digits = 6)
```

Arguments

x   a numeric vector. Or, for round and signif, a complex vector.

digits   integer indicating the number of decimal places (round) or significant digits (signif) to be used. For round, negative values are allowed (see 'Details').

...   arguments to be passed to methods.

R packages

In R functions are stored in units, referred to as *packages*.

- In this course we use only the *basic packages* (base,graphics,...)
- Other packages need to be installed (only one time)

  ```
  >install.packages("Publish")
  ```

- To use the functions we have to *load* the package (every time you open Rstudio)

  ```
  >library("Publish")
  ```

- You can check the installed version of R and loaded packages

  ```
  >sessionInfo()
  ```

Data Structures in R

Types of Variable in R

Every object in R has a specific type.

Type	Value
Numeric	integer
	double (decimal numbers)
Character	text: words, letters
Logical	TRUE/FALSE
Date	calendar date
Factor	categorical variable

We can check the type of the object using the command `typeof()`

Initialize a variable

```
a<-1
b=0.6
c<-"0,3"

> typeof(a)
[1] "double"
> typeof(b)
[1] "double"
> typeof(c)
[1] "character"

> is.numeric(b)
[1] TRUE
```

Data structures in R

When programming it is important to organizing the data in the correct way.
In R we can use several data structures depending on the type of information we want to store.
The data structure is designed so that data can be accessed and worked with in specific ways.
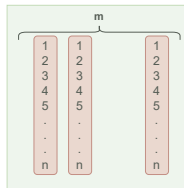Base data structures in R :

- vectors : collection of elements of the same type
- matrix: collection of vectors of the same type, access by row and column
- data.frame: collection of vectors of different type, access by row and column
- list: collection of different data structures of different type

Data structures in R

Vector:
one dimension

Matrix:
two-dimensions

Data frame:
two dimensions

List:
more stractures

Matrix

data.frame
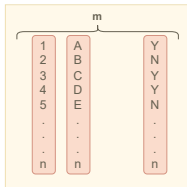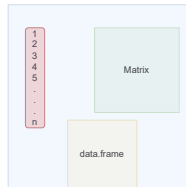
Vector

One dimensional collection of elements of the same type.

Create a vector using c()

```
v1<-c(0,0.3,7,20)
v2<-c("Male","Female")

> typeof(v1)
[1] "double"
> typeof(v2)
[1] "character"
```

Length of a vector:

```
> length(v1)
[1] 4
```

Vector

Create a vector with pre-coded function in R:

- Ordered sequence of integers:

```
> ID<-1:10
> ID
 [1]  1  2  3  4  5  6  7  8  9 10

> weight=0.5:0.7
> weight
[1] 0.5
```

Vector

Create a vector with pre-coded functions in R:

- Sequence of numbers from min to max with a specific lag or of a specific length:

```
> ID<-seq(1,10)
> ID
 [1]  1  2  3  4  5  6  7  8  9 10

> weight<-seq(0.5,0.7, by=0.1)
> weight
[1] 0.5 0.6 0.7

> weight2<-seq(0.5,0.7, length.out=4)
> weight2
[1] 0.5000000 0.5666667 0.6333333 0.7000000
```

Vector

Create a vector with pre-coded functions in R:

- Vector with same repeated value(s):

```
> v1<-rep(0,3)
> v1
[1] 0 0 0
> v2<-rep(c(0,1),2)
> v2
[1] 0 1 0 1
> v3<-rep(c(0,1), c(2,2))
> v3
[1] 0 0 1 1
```

Access a Vector

We can access to subsequences of vectors by using square brackets [ ]

- Specify the element(s) with the *index*

```
> v1
[1]  0.0  0.3  7.0 20.0

> v1[2]
[1] 0.3

> v1[-1] # negative index
[1]  0.3  7.0 20.0

 > v1[c(1,3)]
[1] 0.0 7.0
```
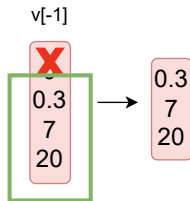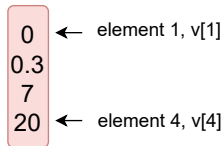
Access a Vector

- Specify the element(s) by conditions:

```
> v1>2
FALSE FALSE TRUE TRUE
> v1[v1>2]
[1]  7 20

> v3
 [1] "B" "A" "C" "C" "C" "C" "B" "C" "B" "B"

> v3[v3 %in% c("A","C")]
[1] "A" "C" "C" "C" "C" "C"
```

v[v>2]   v>2?

```
  0         F
 0.3        F
  7         T
 20         T
```

→

```
  7
 20
```
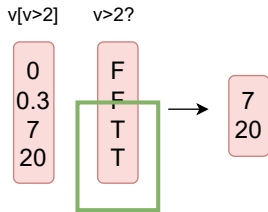
Matrix

A matrix is a two dimensional collection of elements of the same type.
Create it with `matix()`

```
m1<-matrix(c(1,2,3,4,5,6), nrow=2)
m2<-matrix(c(1,2,3,4,5,6), nrow=2, byrow = TRUE)
> m1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> m2
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Create a Matrix from vectors

```
> v1
[1]  0.0  0.3  7.0 20.0
> v2
[1] 1 4 3 5
> rbind(v1,v2)
   [,1] [,2] [,3] [,4]
v1    0  0.3    7   20
v2    1  4.0    3    5
> cbind(v1,v2)
       v1 v2
[1,]  0.0  1
[2,]  0.3  4
[3,]  7.0  3
[4,] 20.0  5
```

Access a Matrix

To access to one specific row/column/element we use $[r, c]$ where r, c are the index of the row and column of interest

```
> m1
        [,1] [,2] [,3]
  [1,]    1    3    5
  [2,]    2    4    6

> m1[,2]
  [1] 3 4

> m1[2,]
  [1] 2 4 6

> m1[2,2]
  [1] 4
```

Access a Matrix

As for vectors, we can use conditions :

```
> m1
        [,1] [,2] [,3]
  [1,]    1    3    5
  [2,]    2    4    6

> m1[m1>2]
 [1] 3 4 5 6
```

Data frame

Collection of vectors with same dimension that can be of different type. Create a Data set with data.frame()

```
> db1<-data.frame(id=c(1,2,3), sex=c("Male","Female","Male"),
                  age=c(46,53,38))

> db1
  id    sex age
1  1   Male  46
2  2 Female  53
3  3   Male  38
```

Access a Data frame

To access to one specific row/column/element we can use [ ] as for matrices or
$NameofColumn

```
> db1$sex
[1] "Male"   "Female" "Male"

> db1[,2]
[1] "Male"   "Female" "Male"

> db1[2,]
  id    sex age
2  2 Female  53
```

List

Collection of different data structures of different type. Create it with `list()`

```
family<-list(n=3,
    kids=data.frame(mum.id=c(1,1,2,3),
                    age=c(6,10,8,4),
                    IQ=c(97,101,103,102)),
    mother=data.frame(id=c(1,2,3),
                      smoker=c("Yes","No","Yes"))
    )
```

Access a list

To access to one object of the list we can use [*i*] where *i* is the index of the object in the list OR ["*NameofObjects*"] OR $*NameofObject*

```
> family[1]
$n
[1] 3

> family["n"]
[1] 3

> family$kids
  id age  IQ
1  1   6  97
2  1  10 101
3  2   8 103
4  3   4 102
```

Useful functions

Function	
str()	shows internal structure of an object
length()	dimension of vectors and lists
dim()	dimensions for matrix and data.frame
nrow(),ncols()	number of rows/columns in matrix and data.frame
rownmaes(),colnames()	check/assign names to row and columns

Operation with vectors

$$a=c(1,2,3), \ b=c(3,4,5)$$

```
> sum(a)
[1] 6

> a+b
[1] 4 6 8

> a*a
[1] 1 4 9

> prod(a)
[1] 6
```

Comparison and logical operators

A *logical value* is a value indicating whether something is TRUE or FALSE. This is the usual output of comparative operators:

Comparative operators	test if
Equality operator $==$	operands are equal
Inequality operator $!=$	operands are not equal
Disequality operator $<, <=, >, >=$	less (more) than or equal to
"in" operator $\%in\%$	elements are equal to one of a list of values

Logical operators are used to combine more comparison operators:

- : AND operator
- |: OR operator
- !: not operator

Comparison operators in R

```
> income<-c("low","low","medium","high","medium","high")

> income=="low"
[1]  TRUE TRUE FALSE  FALSE FALSE FALSE

> !(income %in% c("low","high"))
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE

> (income=="low" | income=="high")
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE

> (income=="low" & income=="high") ??
```

Factors

Categorical variables can be represented as a vector of characters. However, using *factors* is more convenient with easier representation.

```
> income<- c("low","low","high","medium","low","high","medium")
> income[1:3]
[1] "low"  "low"  "high"

> income2<-factor(income, levels=c("low","medium","high"))
> income2[1:3]
[1] low  low  high
Levels: low medium high
```

In the factor version the levels are explicitly listed, so it is clear that the two included levels are not all the possible levels.

Missing data

Real-world data sets have missing observations. In R

- NA: missing data (it has a type: logical)
- NaN: not a number
- NULL: empty object

```
> sex<-c("female",NA,"male","male","female","NA")
> is.na(sex)
[1] FALSE  TRUE FALSE FALSE FALSE FALSE

> 0/0
[1] NaN

> height<-NULL #initialize objects
> height
NULL
```

Conversion

We made a difference between types of variables.
We can *convert* one variable to be from one to another type

```
> weight<-c(10L,15L,27L,18L,22L) #with L we specify it is an integer
> typeof(weight)
[1] "integer"


 > as.character(weight)
[1] "10"   "25.4" "27"   "18"   "22"


> weight[2]<-25.4
> typeof(weight) #R converts it automatically
[1] "double"
```

Coercion

Vectors can contain only elements of the same type. Therefore, if more options are included in a vector R *coerces* the vector to be only of one type. This also happens when 2 different types of variables are used for one operation:

```
> c(0,1,FALSE,TRUE) # FALSE:0 TRUE:1
[1] 0 1 0 1

> c("low","high",2)
[1] "low"  "high" "2"

> as.numeric(c("0.4","0.2","0,1"))
[1] 0.4 0.2  NA
Warning message:
NAs introduced by coercion
```

Exercise I

Exercise: We want to conduct a research study about effect of hormonal contraception on **weight change after 3 months of treatment**. We are interested in comparing **two** types of contraception (oral,spiral) in women between **15**-**30** years old.

1. Define 5/6 variables that you would collect for this study (such as ID, treatment type, weight..)
2. Create a data.frame for 10 patients (5 in each treatment group) with the chosen 6 variables

+: Use https://padlet.com/alessandrameddis/intro-to-r-jl6hoio79ssxfhqm Write down the error/warning you encountered during the Exercise

Exercise I: example

```
data.frame(ID=c(1,2,...), weight0=c(56,52.3,...),
weight3=c(58,51,...), age=c(21,16,...), height=c(157,160...))
```

ID	treatment	weight0	weight3	age	height
1	A	56	58	21	157
2	A	52.3	51	16	160
...					

Exercise I:

ID	treatment	weight0	weight3	age	height
1	A	56	58	21	157
2	A	52.3	51	16	160
...					

4. Create a new vector for BMI from *weight0* and *height* (careful with units)
5. Check the type of variable *treatment*? (use the function `typeof()`)
6. Create a new variable to convert treatment into a factor (use the function `factor()`)
7. ADD these new variables to the `data.frame`
   (to add it, you can use the command
   data.frame(db,BMI=bmi,trt2=treatment.factor))
8. Check the dimension of your `data.frame` (number of cols, number of rows)

Exercise I: example

ID	treatment	weight0	weight3	age	height	BMI	trt.factor
1	A	56	58	21	157	22.7	A
2	A	52.3	51	16	160	20.5	A
...							...

Exercise I: example

ID	treatment	weight0	weight3	age	height	BMI	trt.factor
1	A	56	58	21	157	22.7	A
2	A	52.3	51	16	160	20.5	A
...							...

9. How many women have $BMI > 25$?

10. Calculate the difference in weights after three months

11. Calculate the mean weight difference, formula for the mean:
    $$\frac{diff_1 + diff_2 + ... + diff_1 0}{10}$$
    (you can use the function `sum()`)

Data manipulation in R

Data Import and Export

What we need to know about the data to import/export:

1. format of data (.csv, .xlsx, .txt, .rda ...)
2. where data are located

Text Files:

- csv : Comma Separated Values. Data re stored in plain text and each line can be separated by commas (,) or semicolons(;)
- txt: Data are stored in plain text and values are separated by spaces or tabs.

Loading CSV files

To import a .csv file, we have to specify:

- How values are separated
- Is first raw the variable names? (header=TRUE/FALSE)
- How missing values are formatted (na.string= " ")
- Decimal numbers are with periods (.) or commas (,) (dec=".")

Use the function `read.csv`:

```
data<- read.csv( "~/Desktop/KVN2023/Course/IntrotoR/data_ex.csv",
header=TRUE, na.string=" ", sep=",",dec=".")
```

Export CSV files

A data set can be exported as .csv or .txt files so that the data can be used in other programs.

We use the function `write.csv` where we need to precise

- x: object to save
- file: where to save it
- row.names=TRUE/FALSE: Is first raw the variable names?

Exploring the data

We can use several functions to check the structure of the data:

- `View()`: view data in spreadsheet style in the source tab
- `head()`: print first 6 rows of the data set
- `str()`: overview on the structure of the data (with types of variables)
- `summary()`: summary of all variables in the data set with `min,max,mean` for numeric variables and number of observation by levels for categorical variables.

head() function in R

```
 > dim(db1_ex)
[1] 50   4

> head(db1_ex)
  ID year   weight   height
1  1 2000 57.48904 1.677647
2  2 2002 60.65766 1.613070
3  3 2000 59.60541 1.708943
4  4 2002 64.43392 1.794873
5  5 2000 60.58486 1.586404
6  6 2002 61.59315 1.749023
```

str() function in R

```
> str(db1_ex)
'data.frame': 50 obs. of  4 variables:
 $ ID    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ year  : num  2000 2002 2000 2002 2000 ...
 $ weight: num  57.5 60.7 59.6 64.4 60.6 ...
 $ height: num  1.68 1.61 1.71 1.79 1.59 ...
```

summary() function in R

```
> summary(db1_ex)
      ID           year          weight          height
 Min.   : 1   Min.   :2000   Min.   :50.61   Min.   :1.586
 1st Qu.: 7   1st Qu.:2000   1st Qu.:57.87   1st Qu.:1.649
 Median :13   Median :2001   Median :60.33   Median :1.696
 Mean   :13   Mean   :2001   Mean   :60.41   Mean   :1.696
 3rd Qu.:19   3rd Qu.:2002   3rd Qu.:62.97   3rd Qu.:1.735
 Max.   :25   Max.   :2002   Max.   :71.55   Max.   :1.829
```

Create a new variable

```
> db1_ex$BMI=db1_ex$weight/(db1_ex$height)^2
> head(db1_ex)
  ID year   weight   height      BMI
1  1 2000 57.48904 1.677647 20.42603
2  2 2002 60.65766 1.613070 23.31198
3  3 2000 59.60541 1.708943 20.40941
4  4 2002 64.43392 1.794873 20.00078
5  5 2000 60.58486 1.586404 24.07336
6  6 2002 61.59315 1.749023 20.13452
```

Remove a variable

```
> db2<-db1_ex[,-1]

>db2<-db1_ex[, 2:ncol(db1_ex)]

>db2<-db1_ex
>db2$ID<-NULL

> head(db2)
   year   weight   height      BMI
1 2000 57.48904 1.677647 20.42603
2 2002 60.65766 1.613070 23.31198
3 2000 59.60541 1.708943 20.40941
4 2002 64.43392 1.794873 20.00078
5 2000 60.58486 1.586404 24.07336
6 2002 61.59315 1.749023 20.13452
```

Subsetting by condition

Several packages can be used for data management but we will only consider basic R .

```
> db3<-subset(db1_ex, BMI<20)
> head(db3)
   ID year   weight   height      BMI
8   8 2002 63.57266 1.791244 19.81350
9   9 2000 55.87370 1.769065 17.85336
14 14 2002 63.69920 1.829098 19.03970
17 17 2000 58.05573 1.731900 19.35529
19 19 2000 55.43093 1.696504 19.25938
21 21 2000 57.80955 1.722445 19.48538
```

From numerical to binary variable: `ifelse()`

```
 > db1_ex$BMI.cat=ifelse(db1_ex$BMI<20,"underweight","normal")
> head(db1_ex)
  ID year  weight   height      BMI BMI.cat
1  1 2000 57.48904 1.677647 20.42603  normal
2  2 2002 60.65766 1.613070 23.31198  normal
3  3 2000 59.60541 1.708943 20.40941  normal
4  4 2002 64.43392 1.794873 20.00078  normal
5  5 2000 60.58486 1.586404 24.07336  normal
6  6 2002 61.59315 1.749023 20.13452  normal
```

From numerical to categorical variable: cut()

```
> db1_ex$BMI.cat2<-cut(db1_ex$BMI, breaks=c(15,24,30))
> table(db1_ex$BMI.cat2)
(15,24] (24,30]
     47       3

> db1_ex$BMI.cat2<-cut(db1_ex$BMI, breaks=c(15,20,24,30),
                  labels = c("underweight","normal","overweight"))
> table(db1_ex$BMI.cat2)
underweight      normal  overweight
         12          35           3
```

Combining groups

```
> db1_ex$BMI.cat2[db1_ex$BMI.cat2=="overweight"]<-"normal"
 underweight      normal  overweight
         12          38           0

> db1_ex$BMI.cat2<-droplevels(db1_ex$BMI.cat2)
> summary(db1_ex$BMI.cat2)
 underweight      normal
         12          38
```

Descriptive analysis with R

Descriptive analysis

When working with data, it is important to summarize the available data to provide an idea on the population under study.
Example: Danish users of hormonal contraception, it might be of importance to know the age distribution of women, how many women are taking a specific treatment and so on....

Categorical Variable

- Create Tables with the numbers of individuals in each group
- Create Tables with proportion of individuals in each group

Numerical Variable

- Calculate min/max and different quantiles of the distribution
- Calculate mean/median/sd

Tables

```
    > head(db1_ex)
   ID year   weight   height      BMI    BMI.cat
 1  1 2000 58.33538 1.751734 19.01059 underweight
 2  2 2002 66.81557 1.782675 21.02487      normal
 3  3 2000 57.65426 1.699103 19.97065 underweight
 4  4 2002 64.21438 1.698790 22.25118      normal
 5  5 2000 52.71003 1.712512 17.97322 underweight
 6  6 2002 57.99847 1.683144 20.47265      normal
```

One-way Table:

```
 > table(db1_ex$BMI.cat)

  normal underweight
      30          20
```

Two-way Table:

```
 > table(db1_ex$BMI.cat,db1_ex$year)

                2000 2002
    normal        11   19
    underweight   14    6
```

table(): more example

```
> table(db1_ex$weight>60, useNA="ifany")
FALSE  TRUE  <NA>
   28    21     1

> table(db1_ex$weight>60,db1_ex$year,
                        useNA="ifany")
        2000 2002
  FALSE   17   11
  TRUE     7   14
  <NA>     1    0

> prop.table(table(db1_ex$BMI.cat))
 normal underweight
    0.6         0.4
```

Quantitative variables

```
> mean(db1_ex$height)
[1] NA
> mean(db1_ex$height, na.rm=TRUE)
[1] 1.705973

> sd(db1_ex$weight, na.rm=TRUE)
[1] 3.884619

> min(db1_ex$height, na.rm=TRUE)
[1] 1.597908
> max(db1_ex$height, na.rm=TRUE)
[1] 1.80843
> median(db1_ex$height, na.rm=TRUE)
[1] 1.699103
```

summary()

```
> summary(db1_ex$height)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
1.598   1.687   1.699   1.706   1.729   1.808       1

> summary(db1_ex$BMI.cat)
 Length     Class      Mode
     50 character character

> summary(factor(db1_ex$BMI.cat))
    normal underweight
        30          20
```

quantile()

```
> quantile(db1_ex$BMI)
    0%      25%      50%      75%     100%
17.10270 19.58117 20.44042 21.79128 23.00572

> quantile(db1_ex$BMI, c(0.25,0.75))
    25%      75%
19.58117 21.79128
```

Calculation groupwise

Sometimes we want to calculate summary statistics for different groups
Example we want to calculate mean of weight by BMI categories:

- tapply()

    ```
    > tapply(db1_ex$weight, db1_ex$BMI.cat, mean, na.rm=TRUE)
     normal underweight
    61.60447    56.53733
    ```

- aggregate()

    ```
    > aggregate(weight ~ BMI.cat, data=db1_ex, FUN=mean, na.rm=TRUE)
          BMI.cat   weight
    1      normal 61.60447
    2 underweight 56.53733
    ```