Chapter 1: Introduction and preliminaries

Contents

1 What is R?

R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.

R is easily extensible using a package library system. A wide-ranging and extensive set of contributed packages is also available from the R archive network (http://cran.r-project.org/).

2 Download R and RStudio

2.1 Installation of R software

Please be aware to have the most recent version installed!

Install R from http://cran.r-project.org.

2.2 Install RStudio

RStudio is a free, open source interface for working with **R**. You can download it from http://rstudio.com/, install, and run it on your computer.

It has 4 fields:

1. script space + data viewer
2. workspace browser + code history window
3. console: executed codes + output
4. files and folders from the work directory window + graphical window + window with the list of installed packages + help window (integrated with R).

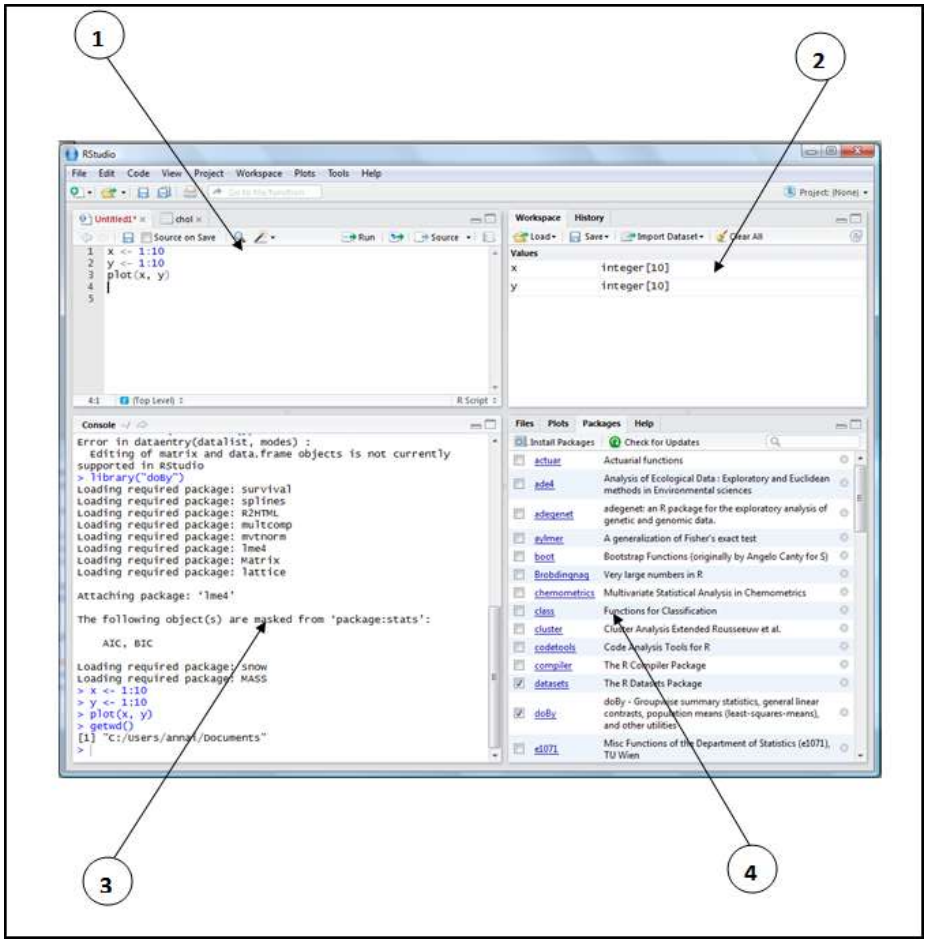
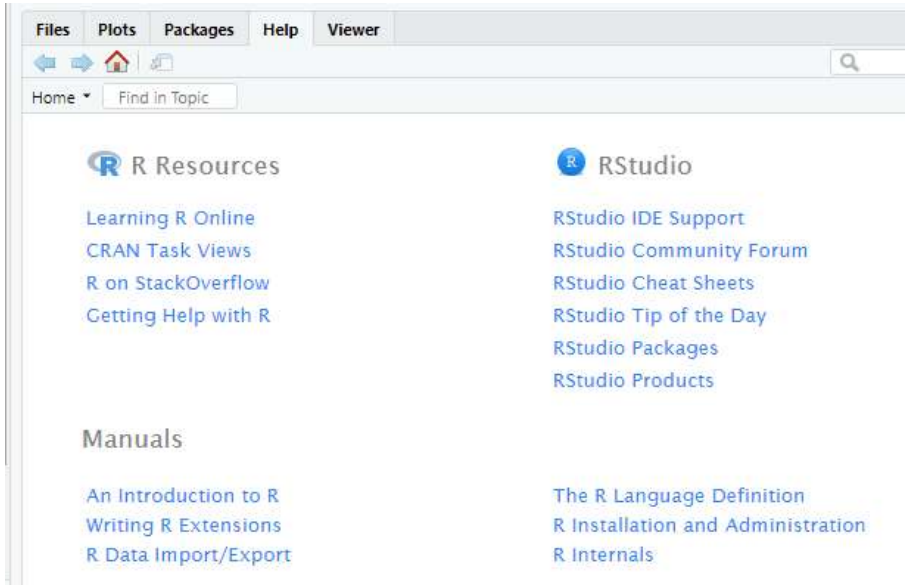


Figure 1: *Figure 1*

3 Manuals

The R distribution also comes with a lot of manuals. In RStudio, you can find the manuals in *Help* window, see window 4 of figure 1.

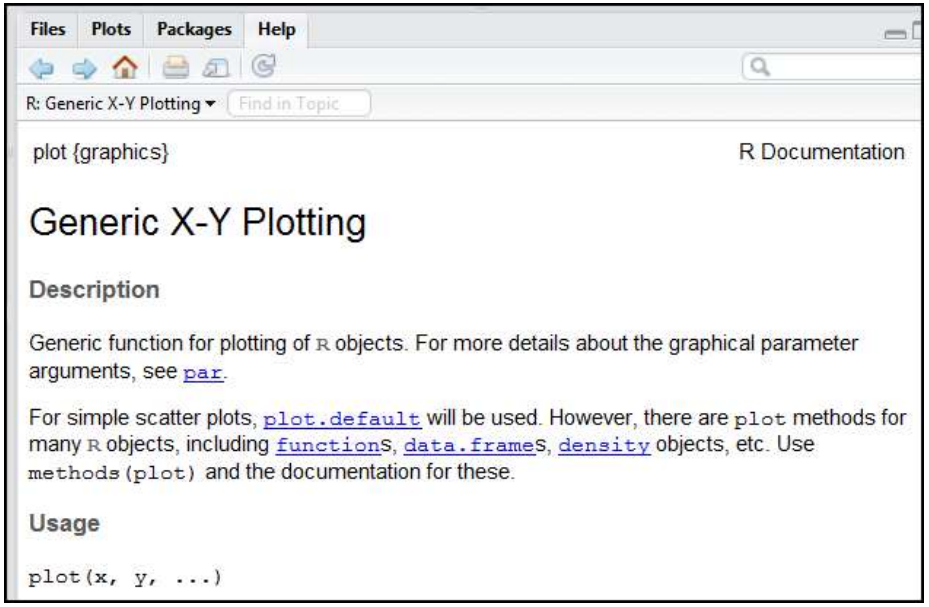


3.1 R help

Several hundreds of help pages, for all functions of R, are available online. They can be accessed by typing `help(NAME)` or `?NAME` at the console (see field 3 of figure 1) where `NAME` is the name of the function help is sought for.

Figure 2: *Example for accessing the help pages of the function* `plot`

As a result, the corresponding web-page from the help-sever is displayed field 4 of figure 1:



You can get the same result by using the search option in the *Help* window in field 4 of figure 1:

3.2 Statistical methods with R

- Classical statistical tests
- Generalized linear models
- Multivariate statistics
- Linear and non-linear mixed effects models
- Robust statistics
- Survival analysis
- Classification and regression trees

- Neural networks
  And many others...

4 Remarks

1. **R is case sensitive**
2. Commands are separated by a semi-colon (`;`) or by a new line.
3. **Comments** can be put anywhere, starting with a **hash mark** (`#`). Everything to the end of the line is a comment.
4. **Assign** a value to an object by `<-` or `=`.

Example:

```
x <- 5
x
```

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

```
x = 5
x
```

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

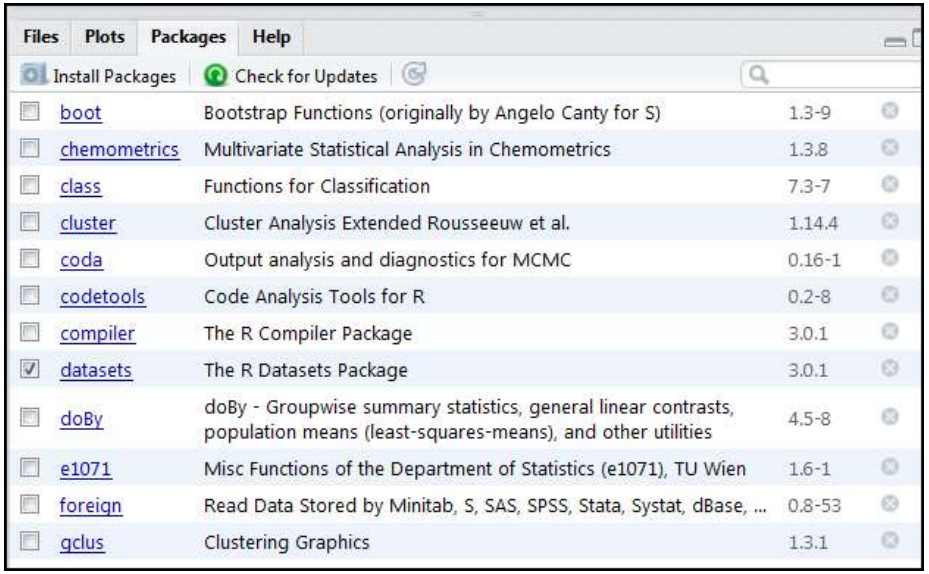
5 Add-on packages in R

It is important to distinguish the following about add-on packages in R:

- To **install** a package: To download a package
- To **load** a package: To activate all its functions

Every package has to be installed only once (if the same computer is used).

Remark: R comes with a limited list of packages.



5.1 Install package

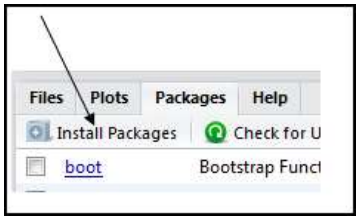
If you need to install (download) an extra package which is not yet available on your system, use the function `install.packages()`.

For instance, to install package `BayesTree`:

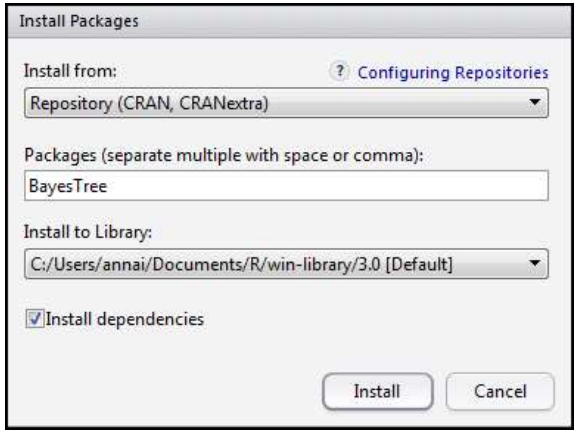
- Use the R code to install the package `BayesTree`:

```
install.packages("BayesTree")
```

- Use "Packages" window to install package `BayesTree`
  In the "Packages" window, click on "Install Packages" button:



Then, complete the dialog window as follows and click "Install"



Remark: Do not forget to load an installed package before you can use it (see next topic)

5.2 Load package

How can add-on packages be loaded?

5.2.1 Load packages by using R code

You can load the installed package `BayesTree` by typing in the console:

```
library(BayesTree)
```

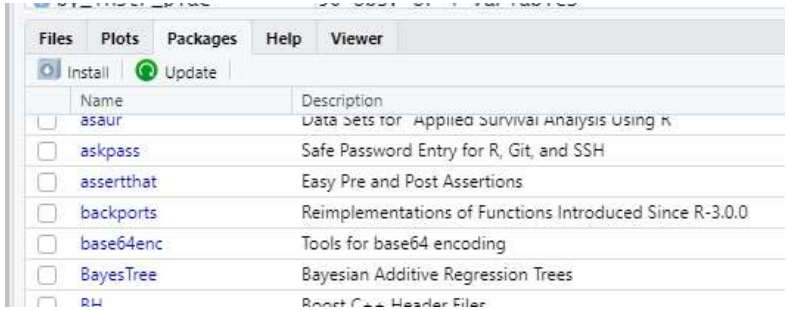
Remark: To find out which functions the package provides use

```
library(help = BayesTree)
help(package = BayesTree)
```

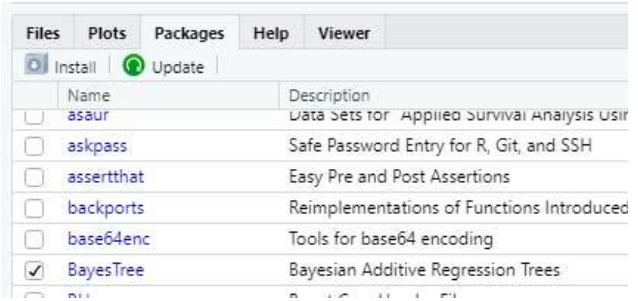
You will obtain in the help window the corresponding web-page from the R help-server.

5.2.2 Load packages by using "Packages" window

To find out which additional packages are installed on your system, check the list in the window "Packages" (see field 4 of figure 1). Here, the loaded packages are defined with a ticked box.



You can load the installed package `BayesTree` by ticking the box with this package:



Remark:

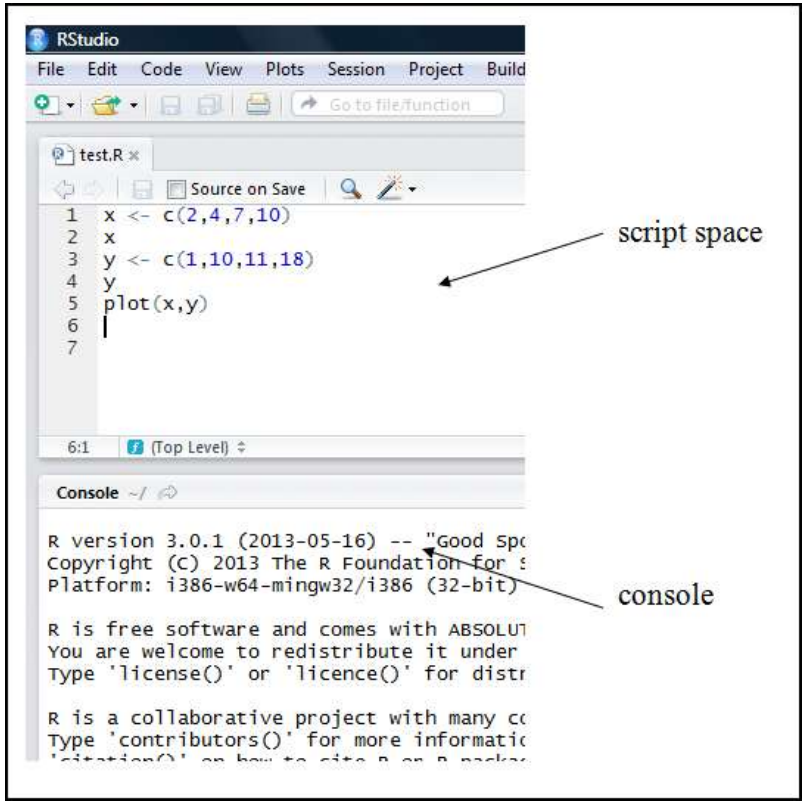
1. To ask for the contents of the package, simply click the underlined name of the package in the "Packages" window.
2. If you want to make it clear what package a function comes from, we will use the package name followed by the name of the function like: `dplyr::filter()`
   `dplyr` is the name of the package, `filter` is the name of a function in that package.

6 Exercise:

1. Check which packages are installed on your system.
2. Install and load the package `robustreg`
3. Check the contents of this package
4. Check which packages are loaded in your system.
   Hint: Use function `search()`

7 Writing scripts

Up to now we used for the syntax only the "Console" field. From now on, we will use 2 fields:



- **Script space** – contains the commands and comments, it can be stored as a file (see later more);
- **Console** – contains the executed commands and output. When you close RStudio, all syntax that was typed in the console will be lost!

A script is a list of commands in a file. Useful features of the script are:

- You can execute the code several times.
- You can include any comments (lines that begin with the `#` character) to remember or to inform others what the script is doing and why.

7.1 Common exercise

Here is an example step-by-step description of how to create and run a simple script that produces a plot.

1. Open in RStudio a text editor: `File > New File > R Script`
2. Type in following lines

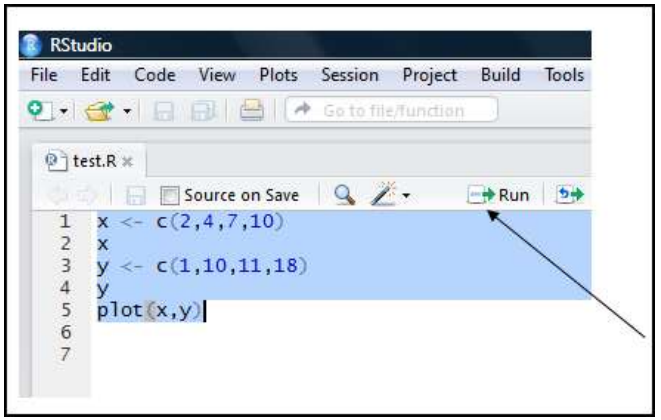
```
x <- c(2, 4, 7, 10)
x
y <- c(1, 10, 11, 18)
y
```



```
plot(x, y)
```

Note: `c()` is a function for creating a vector, `plot()` is the function for generating a scatter plot.

3. Save the file with the name `test.R` in a directory. Close this file.
4. In RStudio, select menu command
   `File > Open File...`
5. In the file selection dialog, locate the file `test.R` that you just saved and select it.
6. Execute the commands of the script:
   - highlight
   - use the "run" button from the toolbar (or use the combination of the keys `Ctrl` + `Enter`)



7. Examine the output.

Remark: When writing scripts, be sure to have the list of packages at the top of the script. Then it is easy to see which packages you need to run the script.

8 Working directory

- You can check the working directory by using the function `getwd()`.

You will certainly have another result than below.

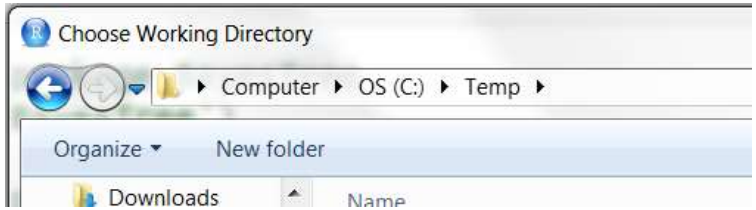
```
getwd()
```

```
## [1] "I:/AnCa/Data/Kursussen/Kursussen/R_software/2020-2021/notes_new/Chapter1"
```

Usually, R's default working directory is "C:/Users/User Name/Documents".

- You can change the working directory by using the function `setwd()`.
  For example: 'setwd("C:/Temp"). In this example, the working directory has been set to a folder *Temp*.

You can also use the menu Session in RStudio: `Session > Set Working Directory > Choose Directory...`



Remark:
`ls()`: list all objects in the current workspace
`rm(list = ls())`: removes all objects in the current workspace

Chapter 2: Data structures

Contents

R operates on data structures. There are different data structures: vector, factor, matrix, data frame and list.

1 Vectors

- Simplest type of data structure
- **Vector creation**: with function `c`:

```
vec <- c(5, 8, 4.6, 21)
```

- To **see** the content of a vector created and assigned to an object, call the name of this object:

```
vec
```

```
## [1]  5.0  8.0  4.6 21.0
```

OR

use an extra pair of brackets during the creation and assignment of the object:

```
(vec <- c(5, 8, 4.6, 21)) # Note the extra brackets!
```

```
## [1]  5.0  8.0  4.6 21.0
```

1.1 Vector arithmetic

1.1.1 Useful functions for vectors

1 vector: x	2 vectors: x, y
`length(x)`	
`sum(x)`	`sum(x, y)`
`prod(x)`	`prod(x, y)`
`max(x)`	`max(x, y)`
`min(x)`	`min(x, y)`
`sort(x)`	`sort(x, y)`
`rev(x)`	
`diff(x, lag = )`	
`unique(x)`	
`order(x)`	

Creating a vector `x` with 10 random elements drawn from a normal distribution. Some of the above listed functions will be afterwards applied on this vector

```
x <- rnorm(10)
x
```

```
##  [1]  1.16380713 -2.02159814  0.25914675 -0.40558169  0.27371012  1.30685921
##  [7] -0.07173989  1.02192342 -0.37770545 -1.86810075
```

The function `order()` applied on a vector `x` returns a vector with indices. The indices reveal which element of the original vector `x` needs to be put first, second ... in order to sort the vector `x` in ascending (default) or descending order.

```
z <- order(x)
z
```

```
##  [1]  2 10  4  9  7  3  5  8  1  6
```

The function `sort()` gives the elements itself instead of the index.

```
sort(x)
```

```
##  [1] -2.02159814 -1.86810075 -0.40558169 -0.37770545 -0.07173989  0.25914675
##  [7]  0.27371012  1.02192342  1.16380713  1.30685921
```

```
# OR, alternative way to obtain the same result
sort(x, decreasing = FALSE)
```

```
##  [1] -2.02159814 -1.86810075 -0.40558169 -0.37770545 -0.07173989  0.25914675
##  [7]  0.27371012  1.02192342  1.16380713  1.30685921
```

The function `rev()` returns a reversed version of its argument.

```
rev(sort(x))
```

```
##  [1]  1.30685921  1.16380713  1.02192342  0.27371012  0.25914675 -0.07173989
```



```
##  [7] -0.37770545 -0.40558169 -1.86810075 -2.02159814
```

OR, alternative way to obtain the same result

```
sort(x, decreasing = TRUE)
```

```
##  [1]  1.30685921  1.16380713  1.02192342  0.27371012  0.25914675 -0.07173989
##  [7] -0.37770545 -0.40558169 -1.86810075 -2.02159814
```

Creation of a vector `y` which will be used as argument for some other examples.

```
y <- c(1, 3, 8, 3, 7, 21)
```

The function `unique()` extracts unique elements.

```
unique(y)
```

```
## [1]  1  3  8  7 21
```

The function `diff()` returns suitably lagged and iterated differences.

```
diff(y, lag = 2)
```

```
## [1]  7  0 -1 18
```

1.1.2 Mathematical functions on numerical vectors

`abs`, `exp`, `log`, `floor`, `ceiling`, `trunc`, `gamma`, `log10`, `round`, `sin`, `cos`, `tan`, `sqrt`

1.1.3 Generating vectors

How to create regular sequences?

Colon operator `:`	→ To create a regular sequence
`seq`	→ To create a regular sequence
`rep`	→ To replicate value

Use of the colon operator (`:`) to create a regular sequence:

```
y <- 1:8
y
```

```
## [1] 1 2 3 4 5 6 7 8
```

To obtain a descending sequence

```
x <- 20:1
```

Use of the function `seq`

```
z <- seq(from = -2, to = 5, by = 0.5)
z
```

```
##  [1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0
```

Use of the function `rep`

```
u <- rep(2:5, 2)
u
```

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

```r
v <- rep(2:5, 2:5)
v
```

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

1.2 Logical vectors

The elements of a logical vector can have the values `T` (`TRUE`) or `F` (`FALSE`).

```r
x <- c(9, 10, 8, 5, 9) # numerical vector
y <- x<9 # logical vector
y
```

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

1.2.1 Logical expressions

Expression	Meaning
<	Smaller than
<=	Smaller than or equal to
>	Larger than
>=	Larger than or equal to
==	Equal to
!=	Unequal to

1.2.2 Logical operators

Operator	Meaning
&	And
|	Or

1.2.3 Examples

```r
T + T
```

```
## [1] 2
```

```r
T - F
```

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

```r
x <- rnorm(100)
y <- x>0
y
```

```
##   [1] FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
##  [25]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
##  [49] FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
##  [61]  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE
##  [73]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
##  [85]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
##  [97] FALSE FALSE  TRUE  TRUE
```



```r
sum(y)
```

```
## [1] 59
```

1.3 Character vectors

```r
test <- c("Leuven Statistics Research Centre", "2020-2021")
test
```

```
## [1] "Leuven Statistics Research Centre" "2020-2021"
```

1.3.1 Character vector operations

some character functions:

1. **`nchar(text)`**
   This function returns a vector with number of characters in each element of the argument `text`

```r
x <- c("start", "student")
nchar(x)
```

```
## [1] 5 7
```

2. **`paste`**
   Concatenate vectors after converting to character.

```r
paste(c("X", "Y"), 1:4, sep=" ")
```

```
## [1] "X 1" "Y 2" "X 3" "Y 4"
```

3. **`substring(text, start, stop)`**
   Extract substrings in a character vector.

```r
x <- c("start", "student")
substring(x, 1, 3)
```

```
## [1] "sta" "stu"
```

1.4 Coercing

Logical, numerical and character values can be used in one and the same vector.

```r
vector <- c(TRUE, -6.05, "Leuven")
vector
```

```
## [1] "TRUE"   "-6.05"  "Leuven"
```

Data type	Example
Logical	`T` or `F`
Numeric	`-6.05`, `86.06`, ...
Character	`Alabama`, `Leuven`, ...

When values of different modes are combined into one object, then R converts all values to a single mode in a way that preserves as much information as possible.
Increasing order information: logical, numeric, character

1.5 Missing values

`NA` (Not Available) is the symbol used in R to represent missing data (for logical, numerical or character values).

```
y <- c(1,2,3, NA)
is.na(y)
```

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

```
z <- c("Ward", "Wouter", "Lucas", NA)
!is.na(z)
```

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

```
x <- c(5,3,8,NA, 6)
is.na(x>5)
```

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

1.6 Subset of a vector

A part of a vector `x` can be selected by `x[subscript]`

```
y <- c(33, 55, 4, 22, 89)
y[c(2,3)]
```

```
## [1] 55  4
```

```
y[-5]
```

```
## [1] 33 55  4 22
```

```
y[y>30]
```

```
## [1] 33 55 89
```

```
x <- c(5, 9, NA)
y <- x[!is.na(x)]
y
```

```
## [1] 5 9
```

1.7 Give names to elements of a vector

```
y <- c(33, 55, 4, 22, 89)
names(y) <- 1:5
y
```

```
##  1  2  3  4  5
## 33 55  4 22 89
```

2 Factors

- A factor is a vector that contains predefined values. This type of data structure is used to store categorical data. The set of allowed values in a factor are defined by the levels.
- How to create?
  - `factor()`: To create a factor
  - `as.factor()`: To encode a vector as a factor


```
directions <- c("North", "East", "South", "South") # This is a character vector
directions
```

```
## [1] "North" "East"  "South" "South"
```

```
directions.f <- as.factor(directions) # This is a factor
directions.f
```

```
## [1] North East  South South
## Levels: East North South
```

To create a factor with labels

```
status <- factor(c("Medium", "Low", "High", "Medium", "Medium", "High"),
                 levels = c("Low", "Medium", "High"),
                 labels = c("Lo", "Me", "Hi"))
status
```

```
## [1] Me Lo Hi Me Me Hi
## Levels: Lo Me Hi
```

3 Matrices

All the values need to be of the same type.

3.1 Creating a matrix in R

How to create?:

- `matrix()`: To create a matrix
- `rbind()`: To add rows
- `cbind()`: To add columns

Example of the use of function `matrix()`

```
x <- matrix(1:8, nrow = 2, ncol = 4, byrow = FALSE)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Some properties of objects in R are stored as attributes. Examples of attributes are `dim` (the dimension of the object), `dimnames` (the names associated with the dimension of the object.) The attributes of an object can be accessed with the function `attributes`.

```
attributes(x)
```

```
## $dim
## [1] 2 4
```

Example of the use of function `rbind()`
`rbind()` combines the arguments row-wise.

```
x <- rbind(1:4, 5:8)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
```

```
dim(x)
```

```
## [1] 2 4
```

```
dimnames(x)
```

```
## NULL
```

```
attributes(x)
```

```
## $dim
## [1] 2 4
```

Example of the use of function `cbind()`
`cbind()` combines the arguments column-wise.

```
x <- cbind(1:4, 5:8)
x
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    6
## [3,]    3    7
## [4,]    4    8
```

3.2 Adding labels

Adding labels to the rows and to the columns of a matrix:

```
dimnames(x) <- list(paste("row", 1:2), paste("col", LETTERS[1:4]))
```

```
paste("row", 1:2)
```

```
## [1] "row 1" "row 2"
```

```
paste("col", LETTERS[1:4])
```

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

The row variable names comes first, then the column variable names.

```
x <- rbind(1:4, 5:8) # Creating matrix
dimnames(x) <- list(paste("row", 1:2), paste("col", LETTERS[1:4])) # Adding labels
x
```

```
##       col A col B col C col D
## row 1     1     2     3     4
## row 2     5     6     7     8
```

3.3 Calculations on matrices

```
x <- matrix(1:4, nrow = 2) # Creating a matrix x
x
```

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

```
nrow(x) # Returns number of rows present in x
```

```
## [1] 2
```



```
ncol(x) # Returns number of columns present in x
```

```
## [1] 2
```

```
x+x
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

Element-wise product:

```
x*x # Elementwise product
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

Element-wise division:

```
x/x # Elementwise division
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1
```

```
x^2
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

Matrix multiplication: `%*%`

```
x%*%x # Matrix product
```

```
##      [,1] [,2]
## [1,]    7   15
## [2,]   10   22
```

```
y <- c(1,5)
y
```

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

yxy^T:

```
y%*%x%*%y
```

```
##      [,1]
## [1,]  126
```

```
z <- rbind(x, y)
z
```

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

```
z <- cbind(x, y)
z
```

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

```
a <- rbind(1:4, 5:8)
b <- cbind(1:4, 5:8)
a%*%b # Matrix multiplication
```

```
##      [,1] [,2]
## [1,]   30   70
## [2,]   70  174
```

Matrix specific operations

Function	Function returns
`chol(x)`	Choleski decomposition of `x`
`col(x)`	Matrix of which the elements corresponds to the column number of the elements
`row(x)`	Matrix of which the elements corresponds to the row numbers of the elements
`diag(x)`	Diagonal matrix from vector `x`
`ncol(x)`	Number of columns of matrix `x`
`nrow(x)`	Number of rows of matrix `x`
`qr(x)`	QR matrix decomposition
`solve(x)`	Inverse
`svd(x)`	Singular value decomposition
`var(x)`	Covariance matrix of the columns
`t(x)`	Transpose
`eigen(x)`	Eigenvalues and eigenvectors of `x`

Functions for mathematical computing

Function	Action
`solve(a, b)`	Solve the system ax = b for x
`integrate(f, low = a, high = b)`	Integration
`polyroot`	Optimization: Find zeros of a real or complex polynomial.
`uniroot`	Optimization: Search in a interval for a root (i.e., zero) of a function.
`opimize`	Optimization: Search in a interval for a minimum or maximum of a function.
`approx`	Interpolation
`rnorm`, `pnorm`, `dnorm`, `qnorm`	Random generation, distribution function, density and quantile function for the normal distribution

3.4 Subset of a matrix

```
x <- matrix(1:100, ncol = 5) # Creating a matrix x
x
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1   21   41   61   81
```



```
##  [2,]    2   22   42   62   82
##  [3,]    3   23   43   63   83
##  [4,]    4   24   44   64   84
##  [5,]    5   25   45   65   85
##  [6,]    6   26   46   66   86
##  [7,]    7   27   47   67   87
##  [8,]    8   28   48   68   88
##  [9,]    9   29   49   69   89
## [10,]   10   30   50   70   90
## [11,]   11   31   51   71   91
## [12,]   12   32   52   72   92
## [13,]   13   33   53   73   93
## [14,]   14   34   54   74   94
## [15,]   15   35   55   75   95
## [16,]   16   36   56   76   96
## [17,]   17   37   57   77   97
## [18,]   18   38   58   78   98
## [19,]   19   39   59   79   99
## [20,]   20   40   60   80  100
```

```
x[8:12, 3:4] # Returns subset with rows from 8- 12 and columns 3-4
```

```
##      [,1] [,2]
## [1,]   48   68
## [2,]   49   69
## [3,]   50   70
## [4,]   51   71
## [5,]   52   72
```

```
x[2,] # Returns 2nd row of the matrix
```

```
## [1]  2 22 42 62 82
```

```
x[,2] # Returns 2nd column of the matrix
```

```
##  [1] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

4 Data frame

- Data frames are an extension of matrices. **Data frames can have columns of different data types.** Most statistical routines require a data frame as input.

4.1 Creating a data frame

- To create or change a data frame:
  - `data.frame`: Create data frames
  - `cbind`: Combine data frames by column (add new variables to existing data frame, merge data frames with same individuals)
  - `rbind`: Combine data frames by row (add new individuals to existing data frame, merge data frames with same variables)
  - `merge`: Join data frames by their common data

Create a data frame from vectors by using `data.frame`

```
employee <- c("John", "Peter", "Sylvie") # Data type = character
salary <- c(21000, 23400, 26800) # Data type = numeric
```

```
employ.df <- data.frame(employee, salary) #Creation of a data frame with two columns.
#These columns have a different data type.
employ.df
```

```
##   employee salary
## 1     John  21000
## 2    Peter  23400
## 3   Sylvie  26800
```

Adding a new row by rbind function

```
new.df <- data.frame(employee = "Katrien", salary = 25000)
employ2 <- rbind(employ.df, new.df)
head(employ2)
```

```
##   employee salary
## 1     John  21000
## 2    Peter  23400
## 3   Sylvie  26800
## 4  Katrien  25000
```

Adding a new column by cbind function

```
age <- c(40, 23, 31)
employ3 <- cbind(employ2, age)
head(employ3) # This gives an error
```

```
age <- c(40, 23, 31, NA) #Note the extra element in this vector (i.e. NA) which is
#crucial, otherwise an error is generated.
employ3 <- cbind(employ2, age)
head(employ3)
```

```
##   employee salary age
## 1     John  21000  40
## 2    Peter  23400  23
## 3   Sylvie  26800  31
## 4  Katrien  25000  NA
```

Create a data frame by using merge function

```
authors <- data.frame(
    ## I(*) : use character columns of names to get sensible sort order
    surname = I(c("Tukey", "Venables", "Tierney", "Ripley", "McNeil")),
    nationality = c("US", "Australia", "US", "UK", "Australia"),
    deceased = c("yes", rep("no", 4)))

books <- data.frame(
    name = I(c("Tukey", "Venables", "Tierney",
               "Ripley", "Ripley", "McNeil", "R Core")),
    title = c("Exploratory Data Analysis",
              "Modern Applied Statistics ...",
              "LISP-STAT",
              "Spatial Statistics", "Stochastic Simulation",
              "Interactive Data Analysis",
              "An Introduction to R"),
    other.author = c(NA, "Ripley", NA, NA, NA, NA,
                     "Venables & Smith"))
```


Examples where the data frames **authors** and **books** are merged with the **merge** function.

```
authors
```

```
##    surname nationality deceased
## 1    Tukey          US      yes
## 2 Venables   Australia       no
## 3  Tierney          US       no
## 4   Ripley          UK       no
## 5   McNeil   Australia       no
```

```
books
```

```
##       name                        title     other.author
## 1    Tukey     Exploratory Data Analysis             <NA>
## 2 Venables Modern Applied Statistics ...           Ripley
## 3  Tierney                    LISP-STAT             <NA>
## 4   Ripley            Spatial Statistics             <NA>
## 5   Ripley         Stochastic Simulation             <NA>
## 6   McNeil     Interactive Data Analysis             <NA>
## 7   R Core           An Introduction to R Venables & Smith
```

```
m1 <- merge(authors, books, by.x = "surname", by.y = "name")
# by.x and by.y specify the columns used for merging
m1
```

```
##    surname nationality deceased                       title other.author
## 1   McNeil   Australia       no     Interactive Data Analysis         <NA>
## 2   Ripley          UK       no            Spatial Statistics         <NA>
## 3   Ripley          UK       no         Stochastic Simulation         <NA>
## 4  Tierney          US       no                     LISP-STAT         <NA>
## 5    Tukey          US      yes     Exploratory Data Analysis         <NA>
## 6 Venables   Australia       no Modern Applied Statistics ...       Ripley
```

```
m2 <- merge(books, authors, by.x = "name", by.y = "surname")
m2
```

```
##       name                        title other.author nationality deceased
## 1   McNeil     Interactive Data Analysis         <NA>   Australia       no
## 2   Ripley            Spatial Statistics         <NA>          UK       no
## 3   Ripley         Stochastic Simulation         <NA>          UK       no
## 4  Tierney                    LISP-STAT         <NA>          US       no
## 5    Tukey     Exploratory Data Analysis         <NA>          US      yes
## 6 Venables Modern Applied Statistics ...       Ripley   Australia       no
```

4.2 Taking a subset from a data frame

1. Take a subset from a data frame like a **matrix**

```
sub1 <- authors[1:3, 1:2]
sub1
```

```
##    surname nationality
## 1    Tukey          US
## 2 Venables   Australia
## 3  Tierney          US
```

2. Taking a subset from a data frame like a **list** (to be explained later)

```
sub2 <- authors$nationality
sub2
```

```
## [1] US        Australia US        UK        Australia
## Levels: Australia UK US
```

Remark:
To sort the data frame `airquality` by two variables, use the function `orderBy` from the package `doBy`.

More information about the `airquality` data:

```
?airquality
```

```
## starting httpd help server ... done
```

airquality {datasets} R Documentation

New York Air Quality Measurements

Description

Daily air quality measurements in New York, May to September 1973.

Usage

```
airquality
```

Format

A data frame with 153 observations on 6 variables.

[,1]	Ozone	numeric Ozone (ppb)
[,2]	Solar.R	numeric Solar R (lang)
[,3]	Wind	numeric Wind (mph)
[,4]	Temp	numeric Temperature (degrees F)
[,5]	Month	numeric Month (1--12)
[,6]	Day	numeric Day of month (1--31)

Details

Daily readings of the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.

- `Ozone`: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
- `Solar.R`: Solar radiation in Langleys in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park
- `Wind`: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
- `Temp`: Maximum daily temperature in degrees Fahrenheit at La Guardia Airport.

```
sort_air <- orderBy(~Month + Temp, data = airquality)
head(sort_air)
```

```
##    Ozone Solar.R Wind Temp Month Day
## 5     NA      NA 14.3   56     5   5
## 18     6      78 18.4   57     5  18
## 25    NA      66 16.6   57     5  25
## 27    NA      NA  8.0   57     5  27
## 15    18      65 13.2   58     5  15
## 26    NA     266 14.9   58     5  26
```

5 List

- A **list** can contain elements of different data types.

- It is an ordered collection of components.

5.1 Creating a list

Creation of a list with the function `list`.

```
employee <- c("John", "Peter", "Sylvie") # creation of a character vector
salary <- c(21000, 23400, 26800) # creation of a numeric vector
employ.df <- data.frame(employee, salary) # creation of a data frame

y <- letters[1:5] # creation of a character vector
z <- 1:3 # creation of a numeric vector

list(employ.df, y, z)
```

```
## [[1]]
##   employee salary
## 1     John  21000
## 2    Peter  23400
## 3   Sylvie  26800
##
## [[2]]
## [1] "a" "b" "c" "d" "e"
##
## [[3]]
## [1] 1 2 3
```

5.2 Named list

The elements of a list can have a name, i.e., named lists have the attributes `names`.

```
mixlist <- list(logica = c(T, T, T, F),
                plant = c("tree", "bush", "grass"),
                comment = "these components are unrelated")
mixlist
```

```
## $logica
## [1]  TRUE  TRUE  TRUE FALSE
##
## $plant
## [1] "tree"  "bush"  "grass"
##
## $comment
## [1] "these components are unrelated"
```

```
attributes(mixlist)
```

```
## $names
## [1] "logica"  "plant"   "comment"
```

5.3 Accessing elements of a list

To access a list component: use component number or component name.

```
mixlist$plant # Returns a character vector
```

```
## [1] "tree"  "bush"  "grass"
```

```
mixlist[[2]] # Returns a character vector
```

```
## [1] "tree"  "bush"  "grass"
```

```
mixlist[2] # Returns a list
```

```
## $plant
## [1] "tree"  "bush"  "grass"
```

6 Exercises

6.1 Vectors

a. Create the object `test1` with numbers 1.5, 0.7, 45.6.
b. Create a vector `y1` with the numbers from 1 to 10.
c. Create a logical vector `y2` from `y1`. An element of `y2` should be `TRUE` if the corresponding element of `y1` is larger than 5.
d. How many elements from `y1` has a value larger than 5?

6.2 Creating sequences

a. Create a vector `x` with elements (1, 2, 3, . . . , 100)
b. Create a vector `y` with elements (0, 5, 10, 15, . . . , 500)
c. Create a vector `z1` with elements (1, 1, 1, 2, 2, 2, . . . , 50, 50, 50)
d. Create a vector `z2` with elements (1, 2, 2, 3, 3, 3, . . . , 10)
e. Create a vector `z3` with elements (1, 2, 2, 3, 4, 4, 5, 6, 6, . . . , 50, 50)

6.3 Matrix creation

a. Create a vector with 100 random normal numbers and use that to generate a 10 by 10 matrix. Call this matrix `mat1`.
Hint: to generate a random normal vector, use the function `rnorm()`.
b. Add an extra row to `mat1` with the numbers 1 to 10 which will be the new first row. Also add the row with numbers 10 to 1 which will be the last row.
c. Add an extra column to the matrix obtained in step *b* with the number 1 to 12 (as first column).

6.4 Working with data frames

`tips` (reshape package) One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

- tip in dollars
- bill in dollars
- sex of the bill payer
- whether there were smokers in the party
- day of the week
- time of the day
- size of the party

In all he recorded 244 tips.

a. Install and load the package `reshape`.
b. Check the data description of this data frame.
c. Ask for the names of the variables in this data frame.
d. Take a subset of data tips which contains the observations from 1 until 20 and only the variables `tip`, `sex` and `day`.

Chapter 3: Importing and exporting data

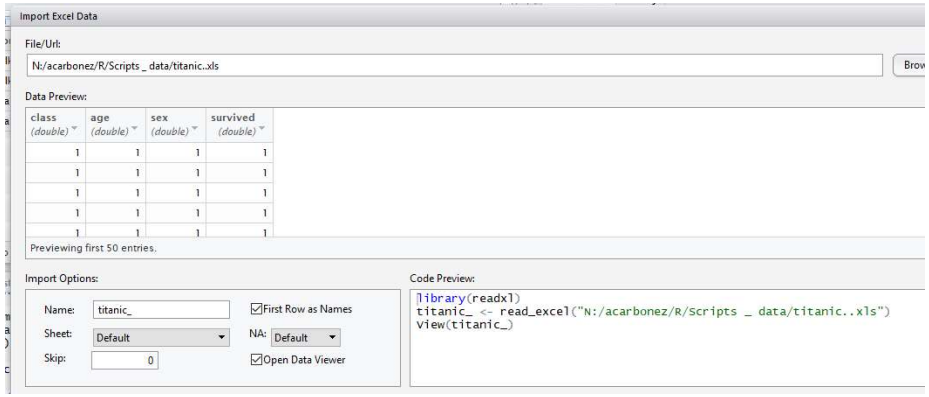
Contents

1 Importing an *Excel* file using *Workspace* window

Example: *Titanic*
Import data *titanic.xlsx*.
This *Excel* file contains information about the passengers of the Titanic. The class (1^{st}, 2^{nd},. . .), the age group, gender and whether the passenger survived. We want to import the worksheet *titanic*.

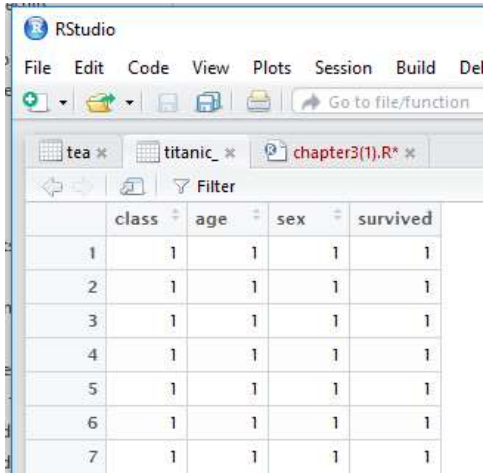
	A	B	C	D
1	class	age	sex	survived
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	1	1	1	1

→ **Environment** → **Import Dataset** → **From Excel . . .**



Do not forget to click the 'First row as Names' check box.

There is now an R data frame titanic_ created



2 Importing an *Excel* file using R function

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

Import the *Excel* file *titanic.xlsx* with the function `read_excel`.

```
titanic2 <- read_excel("C:/Users/.../titanic.xlsx") # Note that '/' needs to be used (not '\')
```

```
head(titanic2, n = 6)
```

```
## # A tibble: 6 x 4
##   class   age   sex survived
##   <dbl> <dbl> <dbl>    <dbl>
## 1     1     1     1        1
## 2     1     1     1        1
## 3     1     1     1        1
## 4     1     1     1        1
## 5     1     1     1        1
## 6     1     1     1        1
```

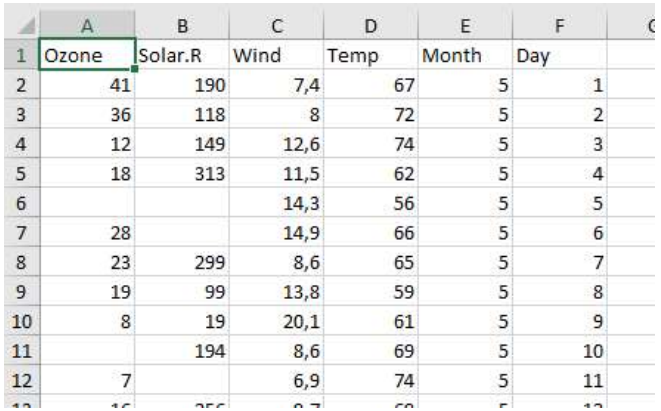

3 Export a data frame to a *xlsx* file

We want to export the R data frame `airquality` (package `datasets`) to a *xlsx* file.

```
head(airquality, n = 6)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
install.packages("openxlsx")
library(openxlsx)
write.xlsx(airquality, file = "AirData.xlsx")
```



	A	B	C	D	E	F
1	Ozone	Solar.R	Wind	Temp	Month	Day
2	41	190	7,4	67	5	1
3	36	118	8	72	5	2
4	12	149	12,6	74	5	3
5	18	313	11,5	62	5	4
6			14,3	56	5	5
7	28		14,9	66	5	6
8	23	299	8,6	65	5	7
9	19	99	13,8	59	5	8
10	8	19	20,1	61	5	9
11		194	8,6	69	5	10
12	7		6,9	74	5	11

4 Importing a *txt* file using the `read.table()` function

Example: To import the data *chol_R.txt* using the `read.table()` function:

```
chol <- read.table(file = file.choose(), header = TRUE)
```

Note:

- The function `file.choose()` allows us to choose the file interactively, rather than typing it.
- The argument `header = TRUE` says that the first line is a line of headings (column names).

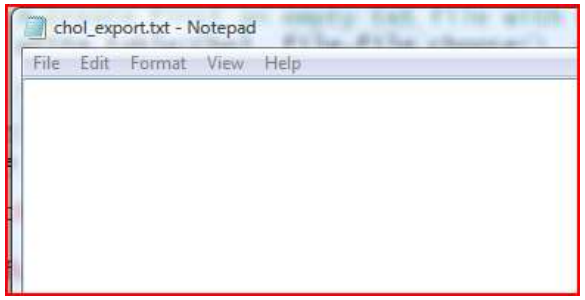
Other possibility:

```r
chol2 <- read.table(file = "C:/Users/.../chol_R.txt", header = TRUE)
```

5 Export a data frame to a *.txt* file

Option 1:

1. Create an empty *.txt* file (for example, in *Notepad*). Save it with the name *chol_export.txt* in your directory.

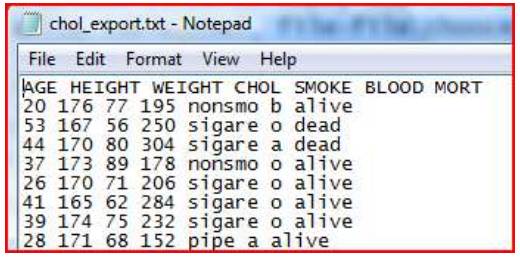


2. Use the function `write.table` to write the data to this file

```r
write.table(chol, file = file.choose(), quote = FALSE,
            sep = " ", row.names = FALSE, col.names = TRUE)
```

Click on *chol_export.txt* in the file selection dialog window when R asks for it.

3. When you open *chol_export.txt*, you obtain the following result:



Option 2:

Use the name and location of your *txt* file

```r
write.table(chol,
            file = "C:/Users/.../chol_out.txt",
            quote = FALSE,
            sep = " ",
            row.names = FALSE,
            col.names = TRUE)
```

On your chosen location (C:/Users/...), the text file *chol_out.txt* will appear which contains the data of the data frame `chol`.

Chapter 4: Writing your own functions

Contents

Syntax:

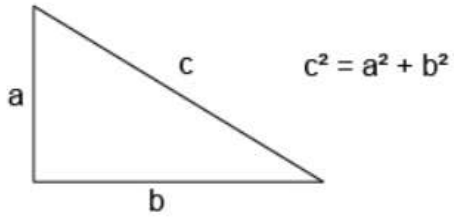
name_function <- function(arg1, arg2,...){Expression}

Expression is an R expression that uses the arguments (*arg1*, *arg2*,...) to calculate a value.

1 Rules to write a function:

1. Make and check the body of the function.
2. If the body of the function is OK, generalize it.
3. Apply the function.

Example
Write a function *Pythagoras* to calculate the length of hypotenuse from length of the legs of a right-angled triangle (Theorem of Pythagoras).



Apply your function when a = 1 and b = 1 and apply the function when a = 3 and b = 4.

1.1 Make and check body

Step 1: Make the body of the function for a specific case and check whether the body of the function is OK:

```
# Step 1: Make and check the body of the function
a <- 3
b <- 4
c <- sqrt(a^2 + b^2)
c
```

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

1.2 Generalization

Step 2: If the body of the function is OK, generalize it:

```
# Step 2: Write the function
Pythagoras <- function(a,b)
{
  c <- sqrt(a^2 + b^2)
  print(a)
  print(b)
  c
}
```

1.3 Apply

Step 3: Apply the function.
How to apply?

- Highlight and submit the function. Then R will recognize it as an R function.
- Apply the created function for other values of the argument(s).

```
# Step 3: Apply the function
Pythagoras(a=1,b=1)
```

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

```
## [1] 1.414214
```

```
Pythagoras(a=3,b=4)
```

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

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

Remark:

1. The last command executed is the **return value** of the function. This can be forced by:
   - using `return` function;
   - using `print` function to force the printout.
2. If you want to obtain several components as result of your function, you have to make use of a list statement.


```
# Use of the list function in your Pythagoras function
Pythagoras <- function(a,b)
{
  c <- sqrt(a^2 + b^2)
  list(a=a, b=b, hypothenusa=c)
}

# Apply your function
Pythagoras(a=1,b=1)
```

```
## $a
## [1] 1
##
## $b
## [1] 1
##
## $hypothenusa
## [1] 1.414214
```

```
Pythagoras(a=3,b=4)
```

```
## $a
## [1] 3
##
## $b
## [1] 4
##
## $hypothenusa
## [1] 5
```

2 Overview of some useful functions in R

Some standard functions

2.1 Functions to convert to integers

```
x <- -3.526
```

Function	Description	Result in R for x = -3.526
`round(x)`	rounds to nearest integer	-4
`trunc(x)`	leaves out the decimal part	-3
`floor(x)`	takes the nearest integer which is smaller than x	-4
`ceiling(x)`	takes the nearest integer which is larger than x	-3

2.2 Integer operators

```
x1 <- 21
x2 <- 5
```

Function	Description	Alternative way	Result in R for x1 = 21 and x2 = 5
%/%	Integer divide	`floor(x1/x2)`	4
%%	Modulu reduction	`x1-floor(x1/x2)*x2`	1

2.3 Some common functions

`abs` (computes the absolute value), `log`, `sqrt` (computes the square root), `exp`, `sin`, `cos`, `tan`, `acos`, `asin`, `atan`, `cosh`, `sinh`, `tanh`...
`log(x, base)` has a second (optional) argument, i.e. the base number (default e)

2.4 Functions on 1 vector

Function	Description	Result in R for `vec <- 1:5`
`length(vec)`	Returns the length of an object	5
`sum(vec)`	Returns the sum of all the values present in `vec`	15
`prod(vec)`	Returns the product of all the values present in `vec`	120
`cumsum(vec)`	Returns a vector whose elements are the cumulative sums of `vec`	1, 3, 6, 10, 15
`cumprod(vec)`	Returns a vector whose elements are the cumulative products of `vec`	1, 2, 6, 24, 120
`max(vec)`		5
`min(vec)`		1
`cummax(vec)`	Returns a vector whose elements are the cumulative maxima of `vec`	1, 2, 3, 4, 5
`cummin(vec)`	Returns a vector whose elements are the cumulative minima of `vec`	1, 1, 1, 1, 1
`range(vec)`	Returns a vector containing the minimum and maximum	1, 5
`sort(vec)`		1, 2, 3, 4, 5
`rev(vec)`		5, 4, 3, 2, 1

2.5 Functions on 2 vectors or more

`pmax(vec1, vec2...)`, `pmin(vec1, vec2...)`, `max(vec1, vec2...)`, `min(vec1, vec2...)`, etc.

Function	Description	Result in R
`pmax(c(1,7,3), c(3,4,5))`	Returns a vector with the parallel maxima of the argument vectors	3, 7, 5
`max(c(1,7,3), c(3,4,5))`	Returns the maximum of all the values present in their arguments	7

2.6 Statistical functions

`mean(vec)`, `var(vec)`, `sd(vec)`

3 Exercises

1. Write a function which gives the most elementary statistics for a sample x: min, median, max, mean, sd and length. Apply your function on a vector x with values from 25 to 80.
2. Write a function fun1 which produces the text 'Non-negative number' if you apply fun1 to a positive number and 'negative number' if you apply fun1 to a negative number. You can make use of the `ifelse` function in R. Apply this function to the values 9 and -13:

Input	Desired output
`x <- 9; fun1(x)`	"Non-negative number"
`x <- -13; fun1(x)`	"Negative number"

3. Write a function to solve an equation of second degree ($ax^2 + bx + c = 0$). To solve this equation, first calculate $D = b^2 - 4ac$. In the case $D > 0$, there are two roots: $x_1 = \frac{-b+\sqrt{D}}{2a}$ and $x_2 = \frac{-b-\sqrt{D}}{2a}$.
If possible, make also a plot of the function. Apply your function for the equation $-8x^2 + 6x + 4 = 0$.

Chapter 5: Graphics with R

Contents

From 2005 on, the library `ggplot2` was started. It is an attempt to take the good things about base and lattice graphics and improve on them (see later topic).

1 Scatterplot

Syntax:

```
plot(Y ~ X, ...)
```

Create a two-dimensional scatterplot of `X` versus `Y` in R as:
`plot(X, Y)` or `plot(Y~X)`
with possible arguments:

- `main`: plot title
- `xlab`: label for the x-axis
- `ylab`: label for the y-axis
- `col`: color of what is plotted
- `xlim`: limits for the x-axis
- `ylim`: limits for the y-axis
- `cex`: magnification factor
- `type`: `p` for points, `l` for lines, or `h` for histogram-like vertical lines, ...
- `pch`: style of the points
- `lty`: type of the line
- `lwd`: thickness of the line

High-level graphic functions will set up the plot figure.
Low-level graphic functions add to the existing figure.

Function	High- or low-level function	Description
`plot(x, y)` or `plot(y ~ x)`	High-level	Function to create a plot
`points()`	Low-level	Function for drawing points
`lines()`	Low-level	Function for joining points with line segments
`abline()`	Low-level	Function for adding lines to a figure
`curve()`	Low- or high-level. It depends on the argument `add = TRUE` or `add = FALSE`	Function for drawing a curve corresponding to a function.
`text()`	Low-level	Adds text to a figure or to special points
`title()`	Low-level	Adds a title to a figure
`legend()`	Low-level	Adds a legend to a figure

Consider the `airquality` data (package `datasets`)

airquality {datasets}

New York Air Quality Measurements

Description

Daily air quality measurements in New York, May to September 1973.

Usage

```
airquality
```

Format

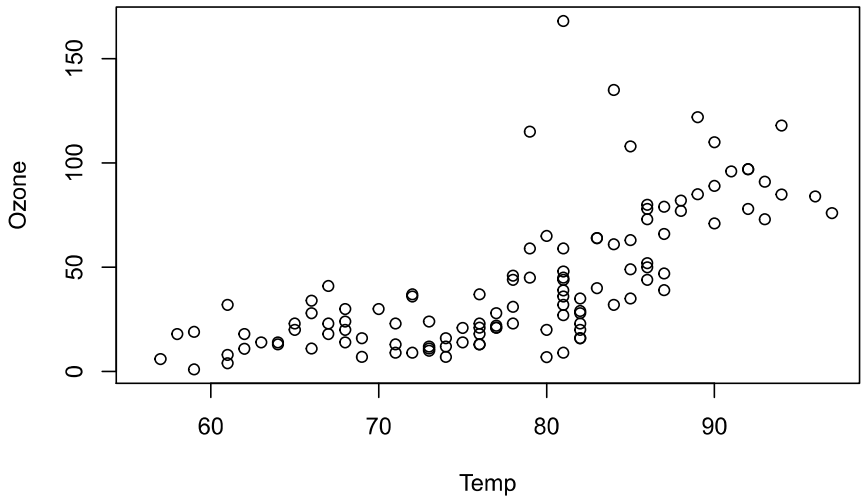
A data frame with 153 observations on 6 variables.

`[,1]`	`Ozone`	numeric Ozone (ppb)
`[,2]`	`Solar.R`	numeric Solar R (lang)
`[,3]`	`Wind`	numeric Wind (mph)
`[,4]`	`Temp`	numeric Temperature (degrees F)
`[,5]`	`Month`	numeric Month (1--12)
`[,6]`	`Day`	numeric Day of month (1--31)

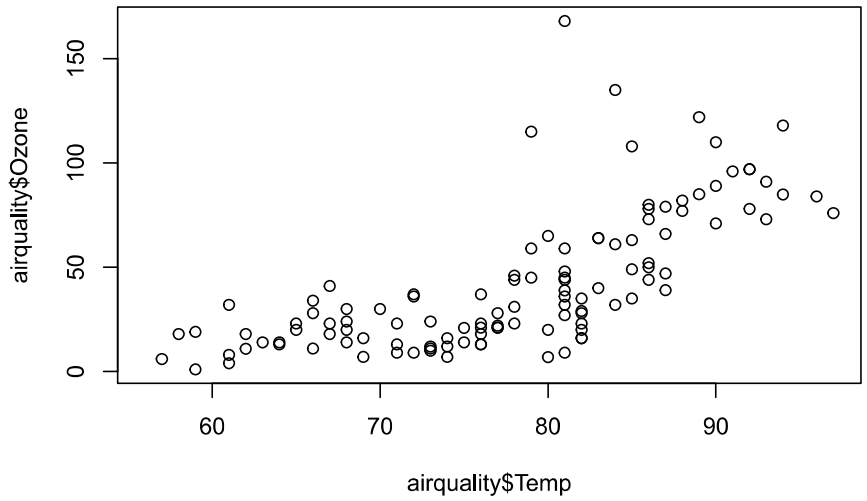
```
library(datasets)
names(airquality)
```

```
## [1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```

```
plot(Ozone~Temp, data = airquality)
```




```
plot(airquality$Temp, airquality$Ozone)
```

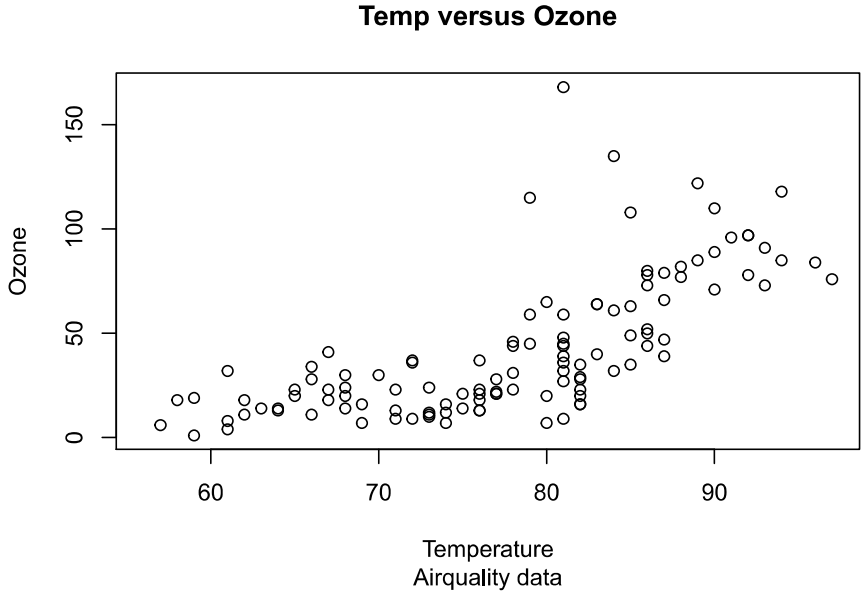



2 Adding to a figure

2.1 Adding titles and labels to a figure

Adding a overall title, a sub title and adjust the label of the x-axis:

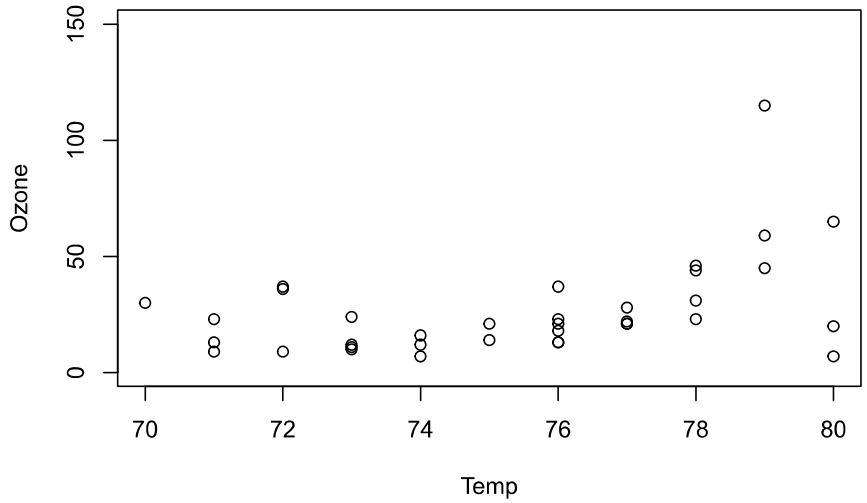
```
plot(Ozone~Temp, data = airquality, main = 'Temp versus Ozone',
     sub = 'Airquality data', xlab = 'Temperature')
```



2.2 Setting up the coordinate system and axes of a graph

```
xlim = c(xmin, xmax)
ylim = c(ymin, ymax)
```

```
plot(Ozone~Temp, data = airquality, xlim = c(70, 80), ylim = c(0, 150))
```

2.3 Plotting points, line types,...

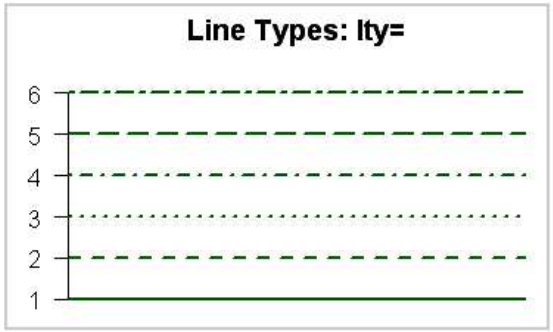
	Argument
What to visualize?	`type = ...`
Line types	`lty = ...`
Plotting characters	`pch = ...`
Plotting colors	`col = ...`
Rescale the size of the symbol	`cex = ...`
Rescale the thickness of the line	`lwd = ...`

2.3.1 Options for `type = ...`

What to visualize?	`type = ...`
Points	`type = 'p'`
Lines	`type = 'l'`
Points and lines	`type = 'b'`
"Overstruck" points and lines	`type = 'o'`
Vertical lines	`type = 'h'`
Stairstep plot	`type = 's'`
Empty plot	`type = 'n'`

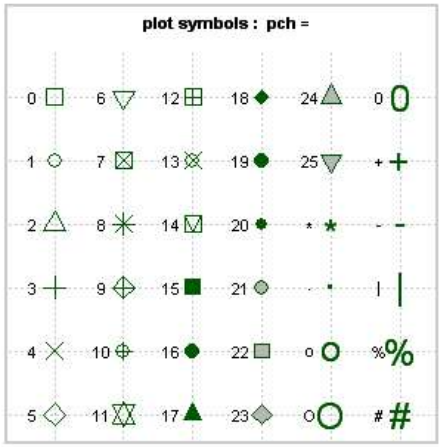
2.3.2 Options for `lty = ...` (line type)

You can select the line type by using the corresponding value for the option `lty = ...`.
The default version of R is `lty = 1` (solid line).
The following possibilities for the line type are available in R:



2.3.3 Options for `pch = ...` (plotting character)

To select the plot symbol, you can use option `pch = ...`.
The default version of R is `pch = 1` (circle). The following symbols are available in R:



2.3.4 Options for `col = ...` (color of the plot)

R stores a current palette that allows for reference of colors by a number.

Every color has its own number:

Color	`col = ...`
Black	`col = 1`
Red	`col = 2`
Green	`col = 3`
Blue	`col = 4`
Cyan	`col = 5`
Magenta	`col = 6`
Yellow	`col = 7`
Gray	`col = 8`

For more info about using colors in R, click here

2.3.5 Options for `cex = ...` (scale of the size of the symbol)

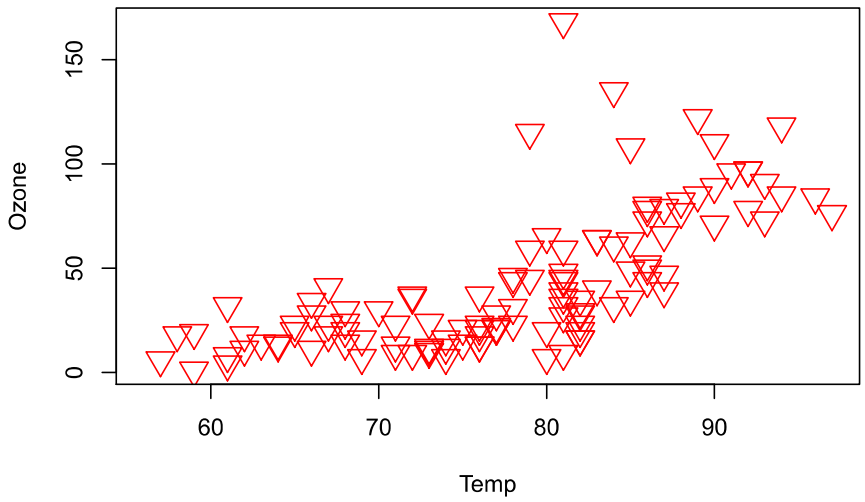
The default version of R is `cex = 1`. For `cex = x`, the size of the symbol is `x` times larger than the default. For instance, if `cex = 2`, the size of the symbol is 2 times larger than the default.

2.3.6 Options for `lwd = ...` (thickness of the line)

The line width can be adjusted with the argument `lwd = ...`. The default version of R is `lwd = 1`. Other values represent the line width relative to the default. For instance, the line is twice as wide than the default if `lwd = 2`.

2.3.7 Several arguments

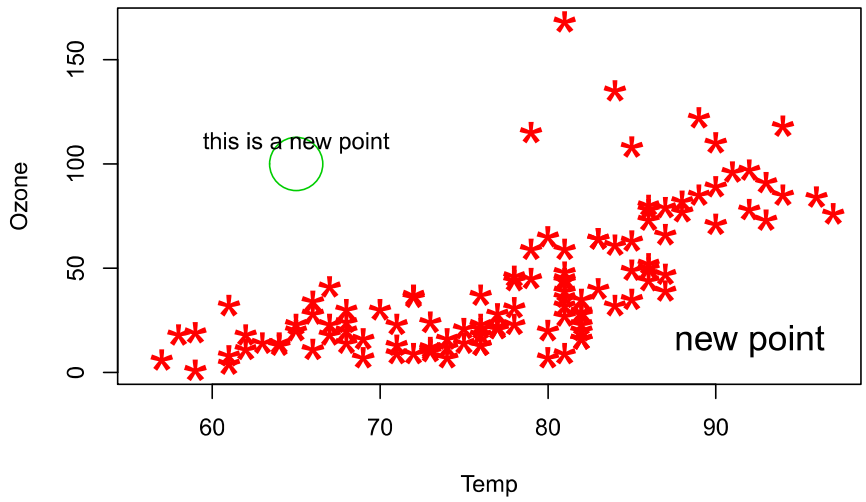
```r
plot(Ozone ~ Temp, data = airquality, type = 'p', col = 2, pch = 6, lty = 3, cex = 2)
```



2.4 Adding information to a plot

Adding points or text to a figure

```
plot(Ozone ~ Temp, data = airquality, pch = '*', cex = 3, col = 2)
points(65, 100, col = 3, cex = 5)
text(locator(1), 'new point', cex = 1.5)
text(65, 110, 'this is a new point')
```

Remark:
By using the `locator()` you can add text interactively, after clicking the chosen location on the graph.

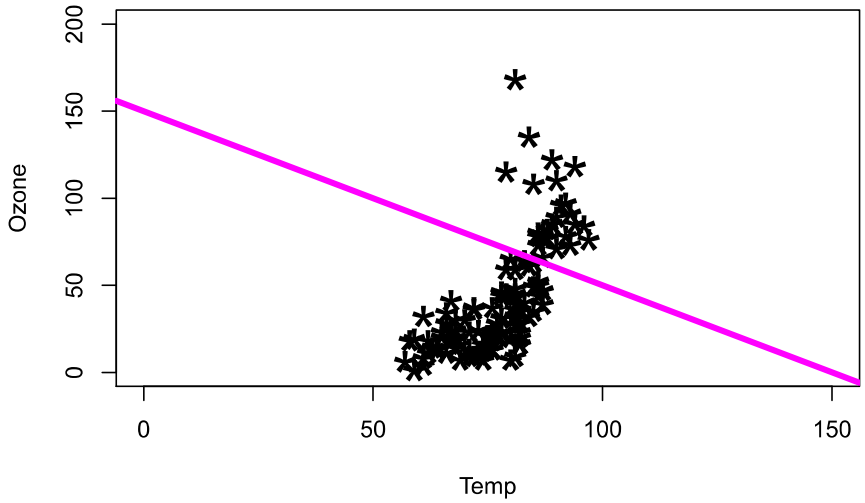
2.5 Adding lines to a graph

You can use the function `lines` or `abline` (or `curve` when you work with functions) to add a line to a graph.

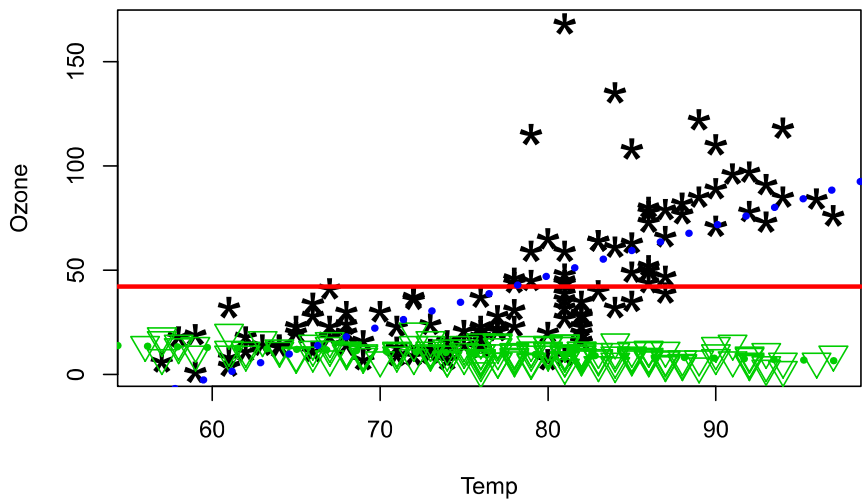
2.5.1 Function `abline`

- *Description:* This function adds one or more straight lines through the current plot.
- *Usage:*
  `abline(a = NULL, b = NULL, h = NULL, v = NULL, ...)`
- *Some arguments:*
  - `a, b` → the intercept and slope, single values.
  - `h` → the y-value(s) for horizontal line(s).

```
plot(Ozone~Temp, data = airquality, pch = "*", cex = 3, xlim = c(0,150), ylim = c(0,200))
abline(150, -1, lty = 1, col = 6, lwd = 4)
```



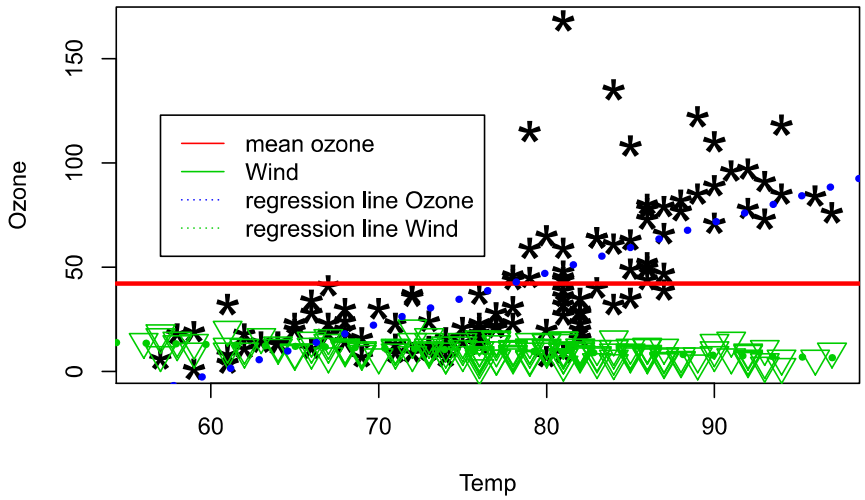

```r
plot(Ozone~Temp, data = airquality, pch = "*", cex = 3)
mean_ozone <- mean(airquality$Ozone, na.rm = T)
abline(h = mean_ozone, lty = 1, col = 2, lwd = 3)

# Adding extra wind measurements
lines(Wind~Temp, data = airquality, pch = 6, col = 3, cex = 2, type = 'p')

# Adding a regression line for Ozone versus Temp
result.lm <- lm(Ozone~Temp, data = airquality)
abline(result.lm, lty = 3, col = 4, lwd = 5)

# Adding a regression line for Wind versus Temp
result2.lm <- lm(Wind~Temp, data = airquality)
abline(result2.lm, lty = 4, col = 3, lwd = 5)
```

Remark:
The function `lm(formula, data)` is used to fit linear models. The argument `formula` is a symbolic description of the model to be fitted. A typical model has the form `response ~ terms`. The function `lm` returns a list that contain the coefficients, the residuals, the fitted mean values, etc.

2.6 Adding a legend to a plot

```r
legend(locator(1), c('mean ozone', 'Wind', 'regression line Ozone', 'regression line Wind'),
       lty = c(1, 1, 3, 4), col = c(2, 3, 4, 3))
```



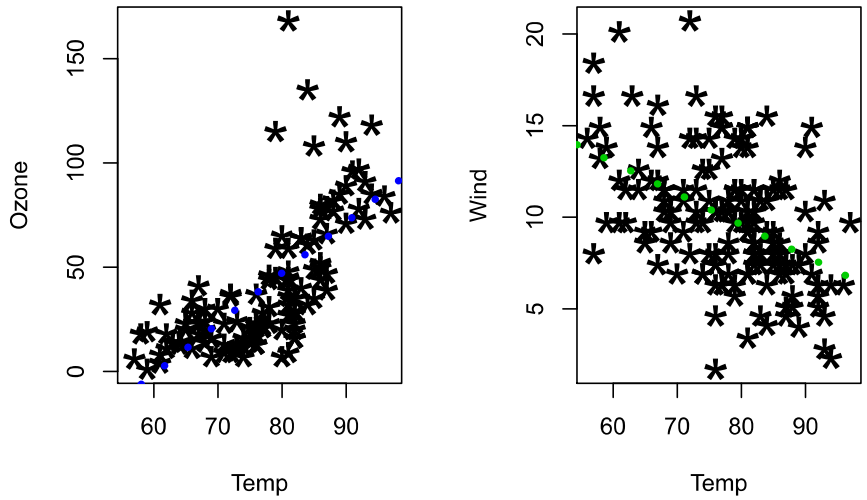
3 Multiple plots on one graphical window

Syntax:

```
par(mfrow = c(3,2))
```

This code tells R to split the graphical window into six equal pieces, with 3 rows and 2 columns.

```
# Multiple plots on same graphical window
par(mfrow = c(1,2))

# Scatterplot and regression line of Ozone versus Temp
plot(Ozone ~ Temp, data = airquality, pch = '*', cex = 3)
result.lm <- lm(Ozone ~ Temp, data = airquality)
abline(result.lm, lty = 3, col = 4, lwd = 5)

# Scatterplot and regression line of Wind versus Temp
plot(Wind ~ Temp, data = airquality, pch = '*', cex = 3)
result2.lm <- lm(Wind ~ Temp, data = airquality)
abline(result2.lm, lty = 4, col = 3, lwd = 5)
```

4 Histogram with usual R graphics

4.1 General

Syntax:

```
hist(x, ...)
```

Creating vectors (`x`, `y`, and `z`) of values for which a histogram will be created.

```
x <- rnorm(50)
y <- rexp(50, rate = 2)
range(y)
```

```
## [1] 0.01011667 1.72388573
```

```
z <- rchisq(50, 10)
range(z)
```

```
## [1]  2.977256 18.341695
```

Split the graphical window in three equal pieces:

```
par(mfrow = c(1, 3))
```

By default, R draws a histogram of absolute frequencies. By setting the argument `probability = TRUE`, we ask for a histogram of relative frequencies.

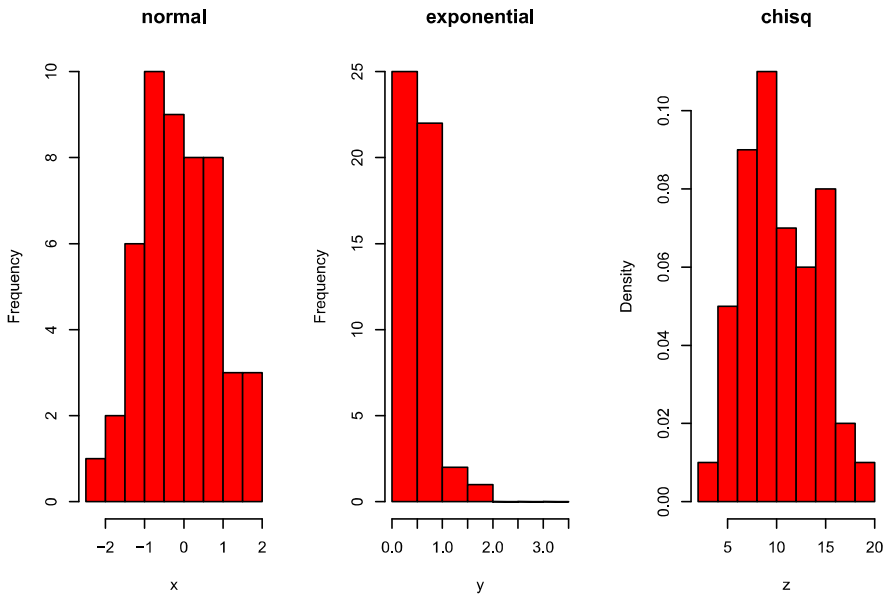
The first two histogram graphics are a representation of absolute frequencies.

```r
hist(x, nclass = 10, main = 'normal', col = 2)

hist(y, breaks = seq(0.0, 3.5, 0.5), main = 'exponential', col = 2)
```

In this third histogram, relative frequencies are plotted (since `probability = T` for this histogram).

```r
hist(z, nclass = 7, probability = T, main = 'chisq', col = 2)
```

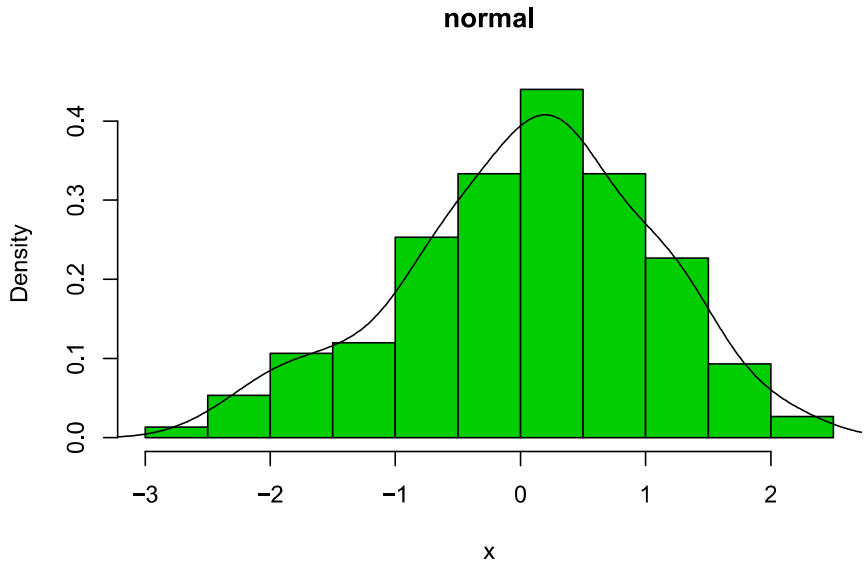
For the first and third histogram, `nclass = ...` determines the number of bars for the histogram (respectively 10 and 7).
In the second histogram, the vector `seq(0.0, 3.5, 0.5)` determines the breakpoints.



4.2 Creating a histogram with a density curve on it

The `density()` function will find a density estimate from the data.
We then use the `lines()` to add the density estimate to the existing graph.

```r
x <- rnorm(150)
hist(x, prob = TRUE, main = 'normal', col = 3)
lines(density(x))
```

5 Boxplots

Syntax:

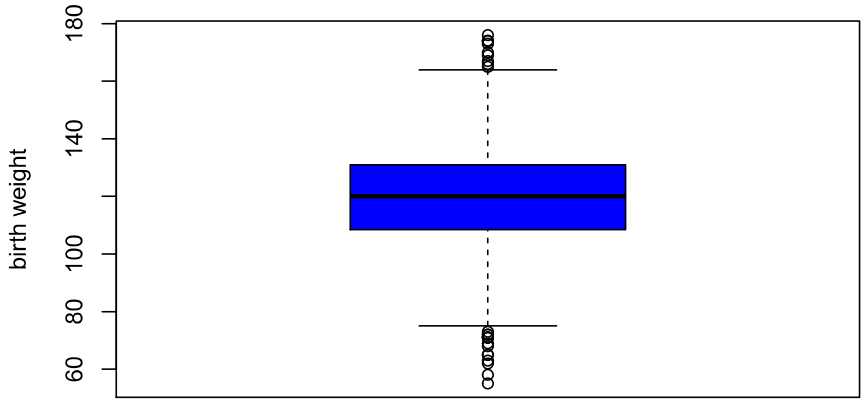
```r
boxplot(x, ...)
```

In this section, we use the data frame `babies` from the package `UsingR`. This data frame is about new born babies. Some important variables are

- `Weight`: Birth weight of the baby (in ounces)
- `Smoke`: Smoke behavior of the mother.
  Does the mother smoke?
  - 0 = never
  - 1 = smokes now
  - 2 = until current pregnancy
  - 3 = once did, not now
  - 9 = unknown

We are making a boxplot for the birth weight (`wt`) variable from the data set `babies`.

```r
boxplot(babies$wt, ylab = 'birth weight', col = 4)
```




```
f <- fivenum(babies$wt)
f
```

```
## [1]  55.0 108.5 120.0 131.0 176.0
```

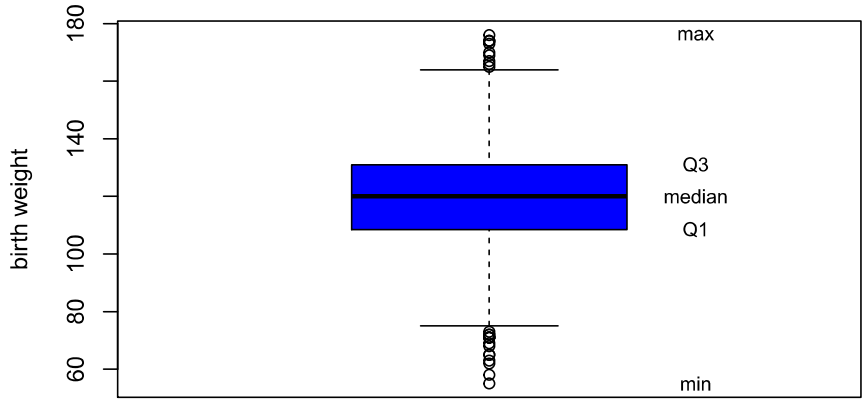
The function `fivenum(x, na.rm = TRUE)` returns Tukey's five number summary for the input data `x`, i.e.,

1. minimum
2. lower-hinge (= lower quantile, Q1)
3. median
4. upper-hinge (= upper quantile, Q3)
5. maximum.

The `text()` function places the values of labels on the graph as specified.
Syntax:

```
text(x, y, labels = ...)
```

```
boxplot(babies$wt, ylab = 'birth weight', col = 4)
text(rep(1.3, 5), f, labels = c('min', 'Q1', 'median', 'Q3', 'max'), cex = 0.8)
```

Remark:

1. Getting the outliers

Outliers are considered as data points which are not within 1.5 · IQR (interquartile range) from the lower quartile Q1 or the upper quantile Q3.

Looking for outliers in birth weight of the babies (`wt`):

```
f <- fivenum(babies$wt)
IQR <- f[4] - f[2]

# What is the identification number of the outlier babies?
outliers <- babies$id[babies$wt > f[4] + 1.5*IQR | babies$wt < f[2] - 1.5*IQR]
outliers
```

```
##  [1] 3906 5287 5845 6241 6343 6460 6534 6660 6760 6997 7080 7083 7109 7290 7334
## [16] 7524 7544 7722 7828 7884 7979 7984 8054 8187 8219 8369
```

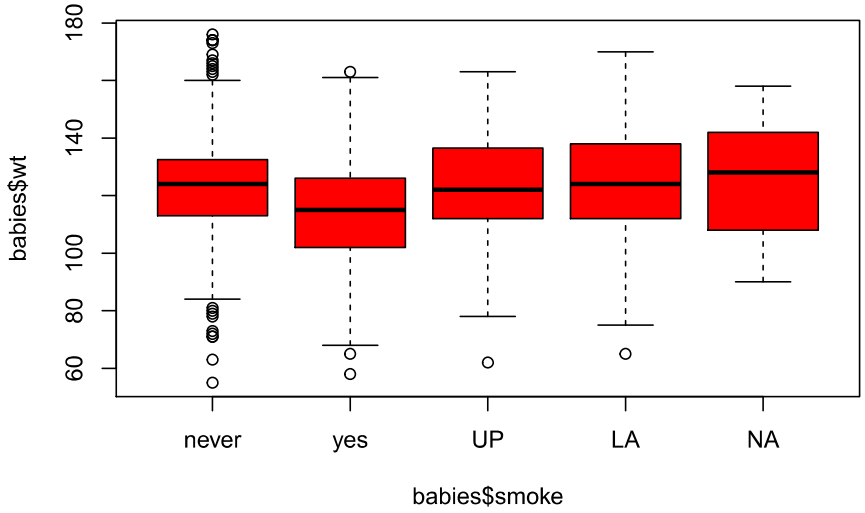
```
# What is the weight of the outlier babies?
outliers_wt <- babies$wt[babies$wt > f[4] + 1.5*IQR | babies$wt < f[2] - 1.5*IQR]
outliers_wt
```

```
##  [1] 173  71  71  68  69  71 174 170 176 166 167  71 174 165  62  72  58  55 169
## [20]  65  73  65 174  63  72  71
```

2. **Grouped** boxplots

We use the data set `babies` and compare the weight of the babies (`wt`) over the several smoking groups of the mothers (`smoke`).

```r
smoke.names <- c("never", "yes", "UP", "LA", "NA")

boxplot(babies$wt ~ babies$smoke, col = 2, data = babies, names = smoke.names)
```



6 Exercises

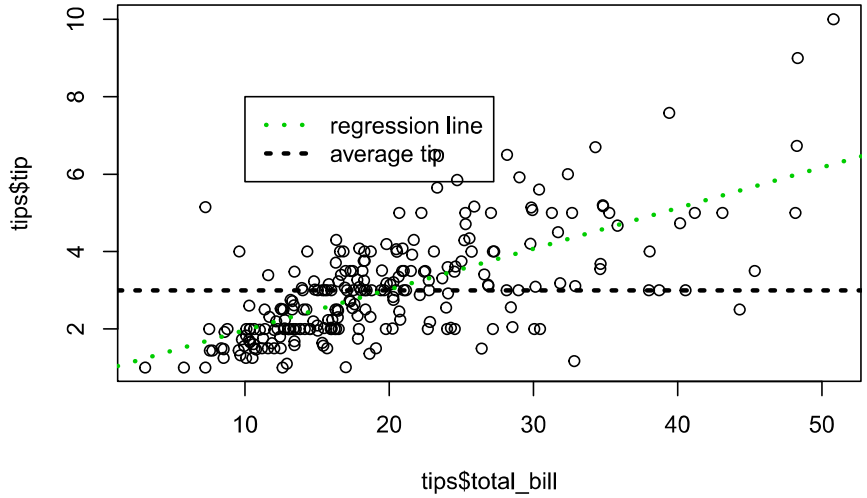
In the first two exercises, the data `tips` from the package `reshape` will be used.

One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

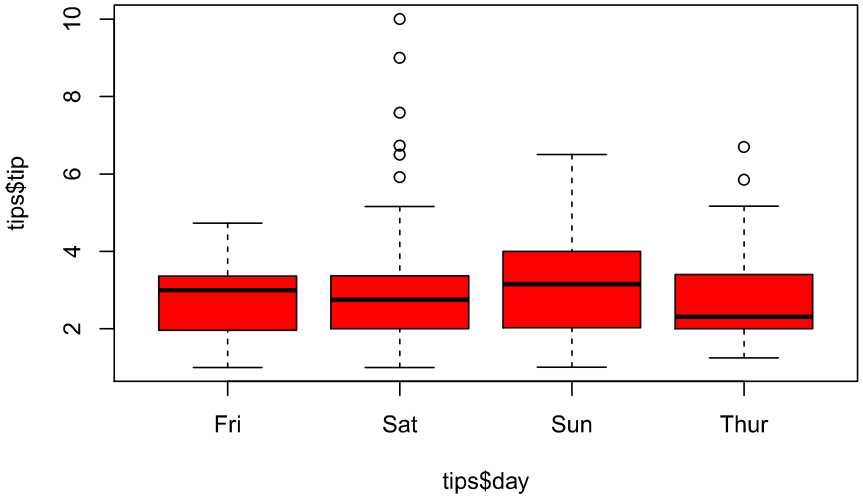
- `tip`: tip in dollars
- `total_bill`: bill in dollars
- `sex`: sex of the bill payer
- `smoker`: whether there were smokers in the party
- `day`: day of the week
- `time`: time of the day
- `size`: size of the party

In all he recorded 244 tips.

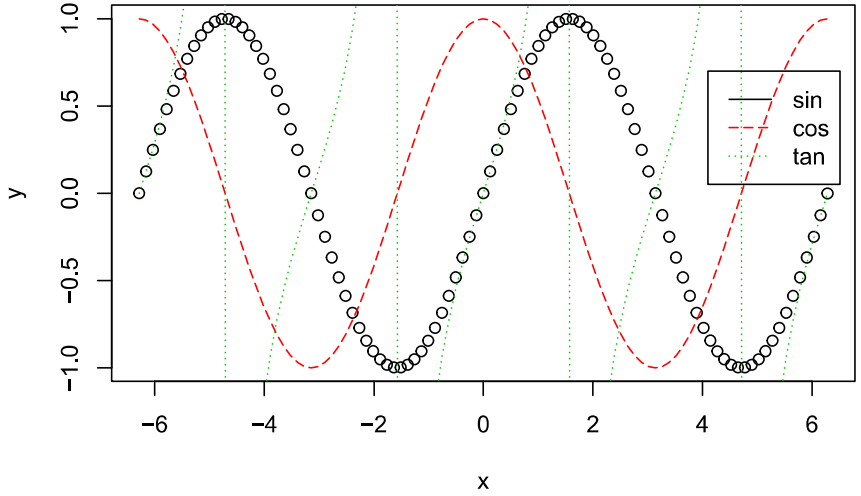
1. Make a simple scatterplot of the tip (Y) versus bill (X). Add a regression line and add a horizontal reference line with average tip. Add a legend.



2. Make a grouped boxplot of tip by day of the week



3. Plotting multiple lines
   This exercise is not using the `tips` *data frame*
   Create the graph of the functions `sin`, `cos`, and `tan` in one figure. Plot the functions on the interval $[-2\pi, 2\pi]$ and separate the graphs by line style (add a legend).



Chapter 6: Some concepts of `dplyr` package *

Contents

1 Data frame: `flights`

The data frame `flights` contains on-time data for all 336 776 flights that departed from New York City in 2013.

```
library(nycflights13)
library(tidyverse)
library(dplyr)
```

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

*Used reference: R for Data Science, Garret Grolemund and Hadley Wickham (see http://r4ds.had.co.nz/transform.html)

This data frame is a **tibble**. It is a data frame with a special format in order to work better with functions of the `dplyr` package.

Abbreviation at column head	Data type
`int`	Integer
`dbl`	Double (real)
`chr`	Character
`dttm`	Date-time
`lgl`	Logical (`T` or `F`)
`fctr`	Factor
`date`	Dates

2 Basic functions in `dplyr` package

Action	Function
Select observations	`filter()`
Select variables	`select()`
Sort data frames	`arrange()`
Create new variables	`mutate()`, `transmute`
Aggregate	`summarise()`
Grouping	`group_by()`
Merging	`inner_join()`, `left_join()`, `right_join()`, `full_join()`

Always same structure:

```
function(data frame, arguments)
```

2.1 Select observations: `filter()`

The `filter()` function is used to subset a data frame, retaining all rows that satisfy your condition. To be retained, the row must produce a value of `TRUE` for all conditions.

- Select all flights of October 1st

```
oct <- filter(flights, month == 10, day ==1)
head(oct)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013    10     1      447            500       -13      614            648
## 2  2013    10     1      522            517         5      735            757
## 3  2013    10     1      536            545        -9      809            855
## 4  2013    10     1      539            545        -6      801            827
## 5  2013    10     1      539            545        -6      917            933
## 6  2013    10     1      544            550        -6      912            932
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- Select all flights of October and all flights of 1st day of the month

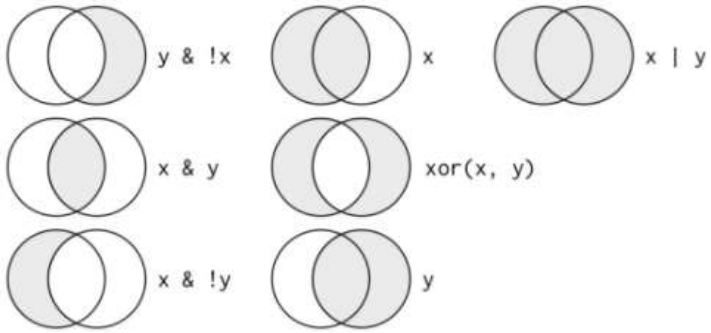
```
first <- filter(flights, month == 10 | day == 1)
head(first)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Comparison operators

>, >=, <, <=	Strict larger, larger or equal, ...
!=, ==	Not equal, equal
month == 7 | month == 8, or alternatively	month $\in \{7, 8\}$
month %in% c(7,8)	

Visualization of Boolean operators



- Select flights on the 1st day of the month or flights of October but not the flights on the 1st of October

```
filter2 <- filter(flights, xor(month==10, day==1)) # Note the use of 'xor'
head(filter2)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
```



```
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

2.2 Select variables: `select()`

The `select()` function selects variables in a data frame.

We first check the names of all the variables of the data frame `flights`:

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

There are several options to select some of these variables:

- Select the variables by *specifying the name*

```
select1 <- select(flights, year, month, day, dest)
names(select1)
```

```
## [1] "year"  "month" "day"   "dest"
```

- Select variables *between certain variables*.
  e.g. select the variables between `origin` and `distance` (inclusive):

```
select2 <- select(flights, origin:distance)
names(select2)
```

```
## [1] "origin"   "dest"     "air_time" "distance"
```

- Select all variables *except certain variables*.
  e.g. select the variables except the variables `origin`, `distance` and all the variables betweeen `origin` and `distance`:

```
select3 <- select(flights, -(origin:distance))
names(select3)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "hour"           "minute"         "time_hour"
```

- Select variables by their column number

```
select4 <- select(flights, c(3:6, 9))
names(select4)
```

```
## [1] "day"            "dep_time"       "sched_dep_time" "dep_delay"
## [5] "arr_delay"
```

- ***Helper functions***
  Helper functions which can be used in `select()`:

`Starts_with("abc")`	Matches names beginning with *abc*
`Ends_with("xyz")`	Matches names ending with *xyz*
`Contains("ijk")`	Matches names containing *ijk*

2.3 Sorting data frame by one (or more) of its variables: `arrange()`

The function `arrange()` orders the rows of a data frame by the values of selected columns.

Example:
Sort the data frame `select1` by `year`, `month`, `day` and descending `destination`

```
sort1 <- arrange(select1, year, month, day, desc(dest))
head(sort1)
```

```
## # A tibble: 6 x 4
##    year month   day dest
##   <int> <int> <int> <chr>
## 1  2013     1     1 XNA
## 2  2013     1     1 XNA
## 3  2013     1     1 XNA
## 4  2013     1     1 TYS
## 5  2013     1     1 TUL
## 6  2013     1     1 TPA
```

Remark: Missing values:
Missing values are always sorted at the end for logical, numerical, and character variables.

Example:
We create a tibble with missing data:

```
df2 <- tibble(x = c(5, 2, NA), y = c('start', NA, 'end'), z = c(NA, TRUE, FALSE))
df2
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1     5 start NA
## 2     2 <NA>  TRUE
## 3    NA end   FALSE
```

```
arrange(df2, x)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1     2 <NA>  TRUE
## 2     5 start NA
## 3    NA end   FALSE
```

```
arrange(df2, desc(x))
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1     5 start NA
## 2     2 <NA>  TRUE
## 3    NA end   FALSE
```



```
arrange(df2, y)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1    NA end   FALSE
## 2     5 start NA
## 3     2 <NA>  TRUE
```

```
arrange(df2, z)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <lgl>
## 1    NA end   FALSE
## 2     2 <NA>  TRUE
## 3     5 start NA
```

2.4 Create new variables: `mutate()` and `transmute()`

The function `mutate()` adds new variables and preserves existing ones. The function `transmute()` adds new variables and drops existing ones.

Example of `mutate()`:

1. Create a new tibble `flights_sml` by selecting the variables `dep_delay`, `arr_delay`, `distance`, and `air_time`
2. Add 2 new variables:
   - `extra` = `arr_delay` - `dep_delay`
   - `hours` = `air_time` / 60

Step 1:

```
flights_sml <- select(flights, ends_with("delay"), distance, air_time)
names(flights_sml)
```

```
## [1] "dep_delay" "arr_delay" "distance"  "air_time"
```

Step 2:

```
new1 <- mutate(flights_sml,
               extra = arr_delay - dep_delay,
               hours = air_time/60)
head(new1)
```

```
## # A tibble: 6 x 6
##   dep_delay arr_delay distance air_time extra hours
##       <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
## 1         2        11     1400      227     9  3.78
## 2         4        20     1416      227    16  3.78
## 3         2        33     1089      160    31  2.67
## 4        -1       -18     1576      183   -17  3.05
## 5        -6       -25      762      116   -19  1.93
## 6        -4        12      719      150    16  2.5
```

Example of `transmute()`:
If you only want to keep the new variables `extra` and `hours`

```r
new2 <- transmute(flights_sml,
                  extra = arr_delay - dep_delay,
                  hours = air_time/60)
head(new2)
```

```
## # A tibble: 6 x 2
##   extra hours
##   <dbl> <dbl>
## 1     9  3.78
## 2    16  3.78
## 3    31  2.67
## 4   -17  3.05
## 5   -19  1.93
## 6    16  2.5
```

2.5 Grouped summaries: `summarise()`

The function `summarise()` aggregates or summarises the input data. This function is usually used on grouped data which is created by the function `group_by()`. If there are grouping variables, the returned data frame will have one (or more) rows for each combination of grouping variables. The output will have one column fr each grouping variable and one column for each of the summary statistics that you have specified.
If there are no grouping variables, the output will have a single row summarising all observations.

2.5.1 Without grouping variable

Using `summarize()` in the absence of grouping variables:
All observations are summarized in single value per summary statistic.

```r
summarise(flights, Avgdelay = mean(dep_delay, na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   Avgdelay
##      <dbl>
## 1     12.6
```

Remark: How is R handling missing data:
Arithmetic functions on missing values yield missing values. For some functions, missing data can be neglected by using the option **`na.rm = TRUE`**. `na.rm` is a logical value indicating whether `NA` values should be deleted before the computation proceeds.

Example: *Titanic*
Creation of vector with missing data

```r
test <- c(1:10, NA, NA)
test
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA
```

Calculating the mean value of this vector

```r
mean1 <- mean(test, na.rm = TRUE)
mean1
```

```
## [1] 5.5
```

```r
mean2 <- mean(test, na.rm = FALSE)
mean2
```

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


The option `na.rm` is also available in other descriptive statistics (e.g., `median()`, `quantile()`, `sd()`, `var()`, `min()`, `max()`, ...).

There exist a function **`is.na()`** to check whether a vector contains missing values.

```r
df2 <- tibble(x = c(5, 2, NA))
x1 <- is.na(df2$x)
x1
```

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

```r
x2 <- !is.na(df2$x)
x2
```

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

2.5.2 Grouping variable is present

Using `summarize()` in the presence of grouping variables:
For each summary statistic, there is an output value per group of observations. To compute summary statistics for every level of a categorical grouping variable or combination of levels of grouping variables, two functions are required:

1. First, the function `group_by()` is used to create a grouped tibble.
2. Afterwards, the function `summarise()` is applied on this grouped tibble to compute summary statistics.

```r
# Create a grouped tibble from an existing tibble.
# Grouping tibble 'flights' by date
by_date <- group_by(flights, year, month, day)

# Compute the mean value per group
summarise(by_date, Avgdelay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day Avgdelay
##    <int> <int> <int>    <dbl>
##  1  2013     1     1    11.5
##  2  2013     1     2    13.9
##  3  2013     1     3    11.0
##  4  2013     1     4     8.95
##  5  2013     1     5     5.73
##  6  2013     1     6     7.15
##  7  2013     1     7     5.42
##  8  2013     1     8     2.55
##  9  2013     1     9     2.28
## 10  2013     1    10     2.84
## # ... with 355 more rows
```

```r
# Compute the number of flights per day
summarise(by_date, n_flights = n())
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day n_flights
##    <int> <int> <int>     <int>
##  1  2013     1     1       842
##  2  2013     1     2       943
##  3  2013     1     3       914
```

```
##  4  2013     1     4       915
##  5  2013     1     5       720
##  6  2013     1     6       832
##  7  2013     1     7       933
##  8  2013     1     8       899
##  9  2013     1     9       902
## 10  2013     1    10       932
## # ... with 355 more rows
```

Remark:
You can use the output to compute other aggregate information.

2.5.3 Useful summary functions

Measure of ...	Summary functions
Center	`mean()`, `median()`
Spread	`sd()`, `IQR()`
Range	`min()`, `max()`, `quantile()`
Position	`first()`, `last()`, `nth()`
Count	`n()`, `n_distinct()`, `sum(!is.na)` (to count number of non-missing)

Example 1

1. Compute by destination:
   - average arrival delay
   - standard deviation of arrival delay
   - max arrival delay
   - count how many non-cancelled flights you have. A cancelled flight is a flight with a `NA` for 'arr_delay.
2. Order these destinations so that the destination with the highest number of flights is on top and then according to increasing average delay.

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
              avg_delay = mean(arr_delay, na.rm = TRUE),
              sd_delay = sd(arr_delay, na.rm = TRUE),
              max_delay = max(arr_delay, na.rm = TRUE),
              count = sum(!is.na(arr_delay)))
arrange(fl_descr, desc(count), avg_delay)
```

```
## # A tibble: 105 x 5
##    dest  avg_delay sd_delay max_delay count
##    <chr>     <dbl>    <dbl>     <dbl> <int>
##  1 ATL      11.3       47.0       895 16837
##  2 ORD       5.88      48.0      1109 16566
##  3 LAX       0.547     39.8       784 16026
##  4 BOS       2.91      38.3       422 15022
##  5 MCO       5.45      42.0       744 13967
##  6 CLT       7.36      41.0       744 13674
##  7 SFO       2.67      47.7      1007 13173
##  8 FLL       8.08      42.9       405 11897
##  9 MIA       0.299     41.3       878 11593
## 10 DCA       9.07      39.9       384  9111
## # ... with 95 more rows
```


Example 2

1. Compute the proportion of flights (by destination) which has an arrival delay of more than 1 hour.
2. Select only those observations where total number of non-missing flights > 100.
3. Order these by proportion long delay.

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
                count = sum(!is.na(arr_delay)),
                prop_long_delay = sum(arr_delay > 60, na.rm = TRUE) / count )
sub1 <- filter(fl_descr, count > 100)
arrange(sub1, prop_long_delay)
```

```
## # A tibble: 92 x 3
##    dest  count prop_long_delay
##    <chr> <int>           <dbl>
##  1 SNA     812          0.0283
##  2 STT     518          0.0347
##  3 HNL     701          0.0442
##  4 ACK     264          0.0455
##  5 MVY     210          0.0524
##  6 SLC    2451          0.0539
##  7 LAS    5952          0.0553
##  8 RSW    3502          0.0571
##  9 MIA   11593          0.0590
## 10 LAX   16026          0.0600
## # ... with 82 more rows
```

3 Combining multiple operations with the pipe

You can avoid creating data frames at every step by using **pipe %>%**.

`x %>% f(y)` turns into $f(x, y)$ `x %>% f(y) %>% g(z)` turns into $g(f(x, y), z)$

Example 1

1. Compute by destination:
   - average arrival delay
   - standard deviation of arrival delay
   - max arrival delay
   - count how many non-cancelled flights you have. A cancelled flight is a flight with a `NA` for 'arr_delay.
2. Order these destinations so that the destination with the highest number of flights is on top and then according to increasing average delay.

Example 1 **without** the use of pipe

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
              avg_delay = mean(arr_delay, na.rm = TRUE),
              sd_delay = sd(arr_delay, na.rm = TRUE),
              max_delay = max(arr_delay, na.rm = TRUE),
              count = sum(!is.na(arr_delay)))
sort_example1 <- arrange(fl_descr, desc(count), avg_delay)
```

Example 1 **with** the use of pipe

```
sort_example1 <- flights %>%
          group_by(dest) %>%
          summarise(
              avg_delay = mean(arr_delay, na.rm = T),
              sd_delay = sd(arr_delay, na.rm = T),
              count = sum(!is.na(arr_delay))
                        ) %>%
          arrange(desc(count), avg_delay)
```

Example 2

1. Compute the proportion of flights (by destination) which has an arrival delay of more than 1 hour.
2. Select only those observations where total number of non-missing flights > 100.
3. Order these by proportion long delay.

Example 2 **without** the use of pipe

```
fl_group <- group_by(flights, dest)
fl_descr <- summarise(fl_group,
                count = sum(!is.na(arr_delay)),
                prop_long_delay = sum(arr_delay > 60, na.rm = TRUE) / count )
sub1 <- filter(fl_descr, count > 100)
arrange(sub1, prop_long_delay)
```

Example 2 **with** the use of pipe

```
sort_example2 <- flights %>%
          group_by(dest) %>%
          summarise(
              count = sum(!is.na(arr_delay)),
              prop_long_delay = sum(arr_delay > 60, na.rm = T)/ count
                        ) %>%
          filter(count > 100) %>%
          arrange(prop_long_delay)
```

Example 3

Imagine that we want to explore the relationship between the *average distance* and *average arrival delay* for every destination.

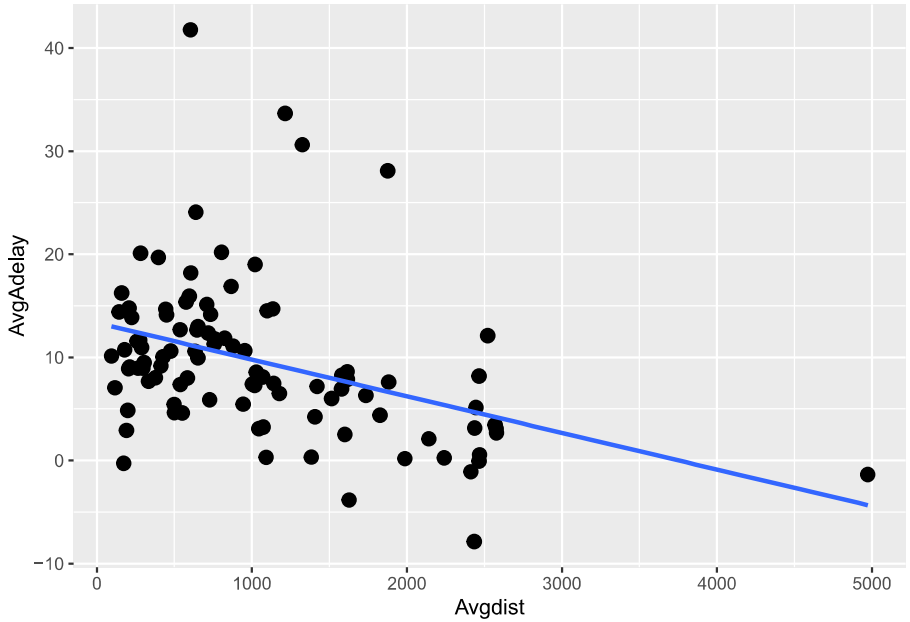
1. Group data frame `flights` by destination
2. Compute average distance (`Avgdist`), average arrival delay (`AvgAdelay`) and number of flights (`count`)
3. Filter the obtained dataset because we are only interested in destination with at least 21 flights.
4. Make a scatterplot of average distance (X) versus average delay (Y). Fit a regression line.
5. We see that there is a destination which completely determines the regression line. Remover that data point and make the plot again.

Example 3 **without** the use of pipe

```
# Step 1
by_dest <- group_by(flights, dest)

# Step 2
sum_dest <- summarise(by_dest,
                      count = n(),
                      Avgdist = mean(distance, na.rm = TRUE),
                      AvgAdelay = mean(arr_delay, na.rm = TRUE))

# Step 3
```

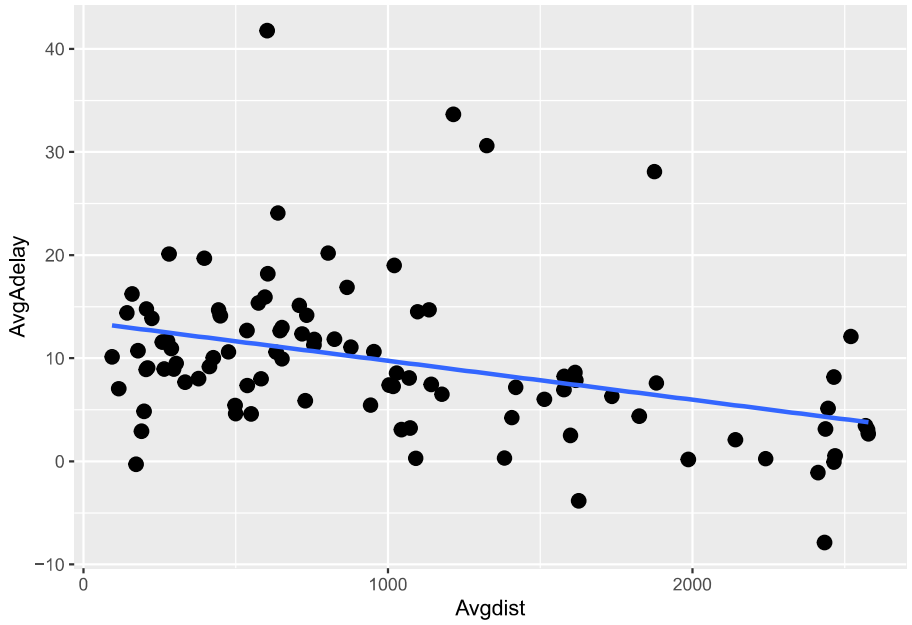


```
sum_dest_sub <- filter(sum_dest, count > 20)

# Step 4
ggplot(data = sum_dest_sub, aes(x = Avgdist, y = AvgAdelay)) +
  geom_point(size = 3) + geom_smooth(se = FALSE, method = 'lm')
```



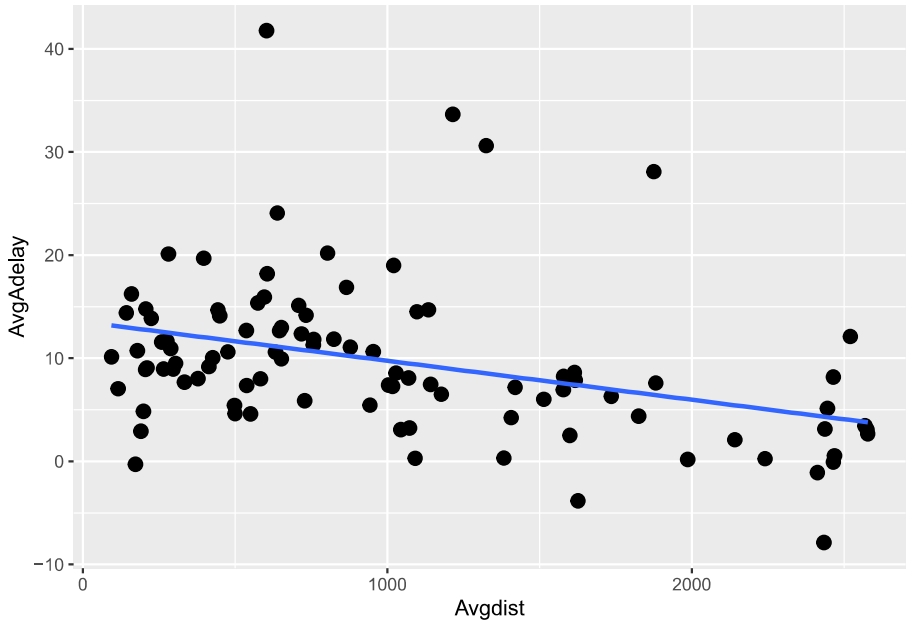
We see that there is a destination which completely determines the regression line. Remove that data point.

```
# Step 5
sum_dest_sub2 <- filter(sum_dest_sub, Avgdist < 3000)
ggplot(data = sum_dest_sub2, aes(x = Avgdist, y = AvgAdelay)) +
  geom_point(size = 3) + geom_smooth(se = FALSE, method = 'lm')
```



Example 3 **with** the use of pipe

```
sum_dest_sub2 <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    Avgdist = mean(distance, na.rm = TRUE),
    AvgAdelay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, Avgdist < 3000)

ggplot(data = sum_dest_sub2, aes(x = Avgdist, y = AvgAdelay)) +
  geom_point(size = 3) + geom_smooth(se = FALSE, method = 'lm')
```

Remark:
When not to use pipe:

1. When you want to debug your code, you will need intermediate steps.
2. When you have multiple inputs (e.g. combining many data frames)

4 Integration of multiple sources: JOIN

1. Matching when key variables have a different name in the two data sets

Example *Offenses 1993 & 1994*
Import *stat93_nmk.xlsx* (reporting of offenses happened in 1993) and import *stat94_nmk.xlsx* (reporting of offenses happened in 1994).

```
stat93_nmk
```

```
## # A tibble: 7 x 3
##   off_code93 DESCR                          yr93
##   <chr>      <chr>                         <dbl>
## 1 1102       Making threats to kill           34
## 2 1112       Assault on an male person       300
## 3 1154       Attempted rape                    7
## 4 1214       Theft form shops                292
## 5 1220       Robbery                           6
## 6 1305       Delivering drugs to a person     64
## 7 1307       Trafficking in controlled drugs   1
```

```
stat94_nmk
```

```
## # A tibble: 6 x 3
##   off_code94 DESCR                          yr94
##   <chr>      <chr>                         <dbl>
## 1 1102       Making threats to kill           56
## 2 1112       Assault on an male person       297
## 3 1154       Attempted rape                    2
## 4 1220       Robbery                          15
## 5 1307       Trafficking in controlled drugs   1
## 6 1311       stalking                         30
```

These data sets can be merged in different ways. The key variable in each data table is `off_code93` or `off_code94`.

4.1 Inner join

An inner join matches pairs of observations whenever their keys are equal. It keeps observations that appear in both tables.
Function: `inner_join()`

```
injoin <- inner_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
injoin[, c(1:3, 5)]
```

```
## # A tibble: 5 x 4
##   off_code93 DESCR.x                       yr93  yr94
##   <chr>      <chr>                        <dbl> <dbl>
## 1 1102       Making threats to kill          34    56
## 2 1112       Assault on an male person      300   297
## 3 1154       Attempted rape                   7     2
## 4 1220       Robbery                          6    15
## 5 1307       Trafficking in controlled drugs  1     1
```

Remark:
Unmatched rows are deleted (e.g. observations with off_code 1214)

4.2 Outer join

An outer join keeps observations that appear in at least one of the tables.

Function:

- `left_join(x, y)`: Left join keeps all observations in `x`
- `right_join(x, y)`: Right join keeps all observations in `y`
- `full_join(x, y)`: Full join keeps all observations that appear in `x` or `y`

```
Ljoin <- left_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
Ljoin[, c(1:3, 5)]
```

```
## # A tibble: 7 x 4
##   off_code93 DESCR.x                       yr93  yr94
##   <chr>      <chr>                        <dbl> <dbl>
## 1 1102       Making threats to kill          34    56
## 2 1112       Assault on an male person      300   297
## 3 1154       Attempted rape                   7     2
## 4 1214       Theft form shops               292    NA
## 5 1220       Robbery                          6    15
## 6 1305       Delivering drugs to a person    64    NA
```



```
## 7 1307       Trafficking in controlled drugs     1     1
```

```
Rjoin <- right_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
Rjoin[, c(1:3, 5)]
```

```
## # A tibble: 6 x 4
##   off_code93 DESCR.x                       yr93  yr94
##   <chr>      <chr>                        <dbl> <dbl>
## 1 1102       Making threats to kill          34    56
## 2 1112       Assault on an male person      300   297
## 3 1154       Attempted rape                   7     2
## 4 1220       Robbery                          6    15
## 5 1307       Trafficking in controlled drugs  1     1
## 6 1311       <NA>                            NA    30
```

```
Fjoin <- full_join(stat93_nmk, stat94_nmk, by = c("off_code93" = "off_code94"))
Fjoin
```

```
## # A tibble: 8 x 5
##   off_code93 DESCR.x                    yr93 DESCR.y                    yr94
##   <chr>      <chr>                     <dbl> <chr>                     <dbl>
## 1 1102       Making threats to kill       34 Making threats to kill       56
## 2 1112       Assault on an male person   300 Assault on an male person   297
## 3 1154       Attempted rape                7 Attempted rape               2
## 4 1214       Theft form shops            292 <NA>                         NA
## 5 1220       Robbery                       6 Robbery                      15
## 6 1305       Delivering drugs to a pers~  64 <NA>                         NA
## 7 1307       Trafficking in controlled ~   1 Trafficking in controlled ~   1
## 8 1311       <NA>                         NA stalking                     30
```

Remark:
the `base::merge()` can perform all four types of join:

dplyr	merge
`inner_join(x, y)`	`merge(x, y)`
`left_join(x, y)`	`merge(x, y, all.x = TRUE)`
`right_join(x, y)`	`merge(x, y, all.y = TRUE)`
`full_join(x, y)`	`merge(x, y, all.x = TRUE, all.y = TRUE)`

5 Differences between a *tibble* and a *data frame*

Most R packages work with data frames, sometimes you need to be able to switch between a tibble and a data frame.

- Make from a data frame a tibble by using the function `as_tibble()`

```
tips.tib <- as_tibble(tips)
head(tips.tib)
```

```
## # A tibble: 6 x 7
##   total_bill   tip sex    smoker day   time    size
##        <dbl> <dbl> <fct>  <fct>  <fct> <fct>  <int>
## 1       17.0  1.01 Female No     Sun   Dinner     2
## 2       10.3  1.66 Male   No     Sun   Dinner     3
## 3       21.0  3.5  Male   No     Sun   Dinner     3
```

```
## 4       23.7  3.31 Male   No     Sun   Dinner     2
## 5       24.6  3.61 Female No     Sun   Dinner     4
## 6       25.3  4.71 Male   No     Sun   Dinner     4
```

- Make from a tibble a data frame by using the functionas_data_frame

```
flights.df <- as_data_frame(flights)
```

```
## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
head(flights.df)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

6 Exercises

6.1 Exercise `tips`

In this exercise, the data set `tips` from the package `reshape` will be used.

One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

- `tip`: tip in dollars
- `total_bill`: bill in dollars
- `sex`: sex of the bill payer
- `smoker`: whether there were smokers in the party
- `day`: day of the week
- `time`: time of the day
- `size`: size of the party

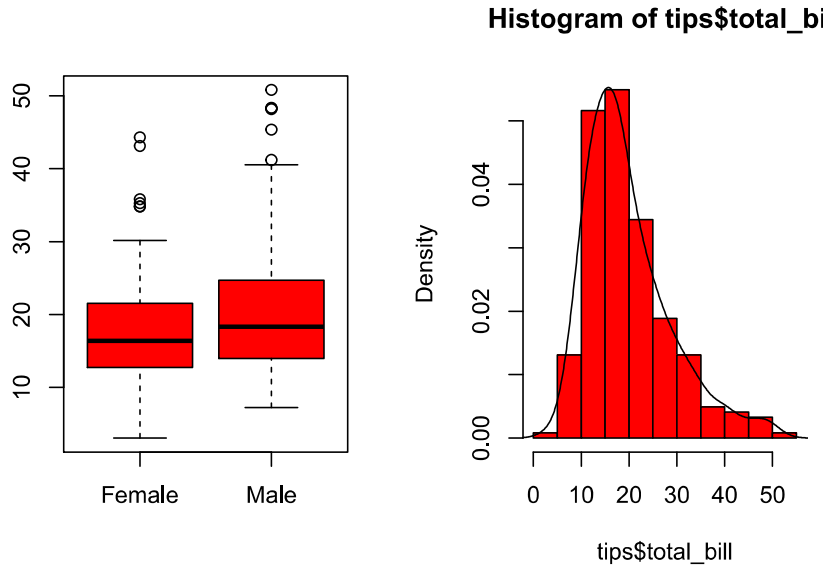
In all he recorded 244 tips.

1. Create a tibble `tips.tbl` from the data frame `tips`
2. Create a subset `sub1` from `tips.tbl` with only those observations with gender `Male` **and** with size of the table larger than or equal to three.
3. a) Create a subset `sub2` from `tips.tbl` with only those observations with gender `Male` **or** with size of the table larger than or equal to three or with both gender `Male` and size ≥ 3.
   b) Create a subset `sub3` from `sub2` with all variables except smoker.
   c) Create a tibble `sort1` by sorting `sub3` by time and decreasing size of table.
   d) Compute average tip by gender (use `sort1`)
   e) Use now the pipe operator to do steps a-d of exercise 3. (start from `tips.tbl`)
4. Compute average tip by day of the week (use the tibble `tips.tbl`).
5. Compute average tip by gender and day of the week (use the tibble `tips.tbl`).

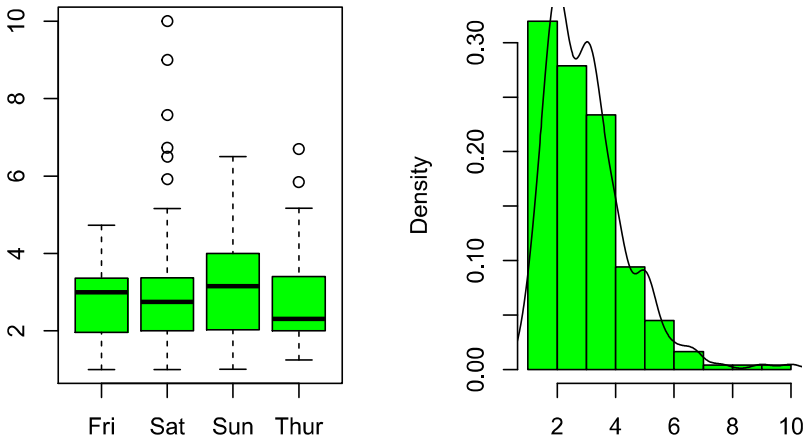
6.2 Overall exercise

In this exercise, the data `tips` from the package `reshape` will be used.

1. Compute descriptive statistics (mean, median, n, stdev) of the variable `total_bill` by gender of the bill payer.
2. Make a grouped boxplot of total bill by gender of the bill payer.
3. Create a histogram of total bill and overlay it with a density curve.
4. Put both graphs next to each other in one graphical window (see below)



5. Create now a function `VISUAL` which is doing subquestion 2, 3, and 4 for you. The input parameters for this function are
   a) `DFR` which is the name of your data frame
   b) `CONT` which is the continuous variable
   c) `CAT` which is the categorical variable
   d) `COL1` which gives you the color number of your graph
6. Apply this function now to obtain visuals for tip by day of the week as below:



6.3 Exercise, missing data

1. Use the `airquality` data frame form the package `datasets`. Use the `summary` function and interpret the result.
2. Count the number of rows in this `airquality` dataframe (use `nrow` function)
3. Count the number of missing values for variable `Ozone`
4. Create a subset `air_complete` of `airquality` with only the complete cases (hint: you can use the function `complete.cases`).
5. Count the number of rows in `air_complete`.

Chapter 7: More programming in R

Contents

1 The family of `apply` functions in R

The family of `apply` functions in R allows to repetitively perform an action on multiple slices of data. The use of the `apply` function or one of its variants avoids the explicit use of loop constructs.

Each function belonging to the `apply` family requires another function as one of the arguments. This specified function will be applied on the input data.

There are so many different `apply` functions because they are meant to operate on different types of data. Not all the variants of the `apply` function will be discussed in this section.

1.1 The apply function in R

Syntax:

```
apply(X, MARGIN, FUN)
```

- X is an array, including a matrix
- MARGIN specifies how the function will be applied:
  - MARGIN = 1: rows
  - MARGIN = 2: columns
  - MARGIN = c(1,2): rows and columns
- FUN is the function to be applied

The function apply returns a vector, an array or a list of values.

Example *boiler*
Use the boiler data (qcc package). It reports temperature readings from eight burners on a boiler. There are 25 observations and 8 variables

```
data(boiler)
head(boiler)
```

```
##    t1  t2  t3  t4  t5  t6  t7  t8
## 1 507 516 527 516 499 512 472 477
## 2 512 513 533 518 502 510 476 475
## 3 520 512 537 518 503 512 480 477
## 4 520 514 538 516 504 517 480 479
## 5 530 515 542 525 504 512 481 477
## 6 528 516 541 524 505 514 482 480
```

- Assume that we want to work with the log values instead of the values itself

```
ln_boiler <- apply(boiler, c(1,2), log)
head(ln_boiler)
```

```
##           t1       t2       t3       t4       t5       t6       t7       t8
## [1,] 6.228511 6.246107 6.267201 6.246107 6.212606 6.238325 6.156979 6.167516
## [2,] 6.238325 6.240276 6.278521 6.249975 6.218600 6.234411 6.165418 6.163315
## [3,] 6.253829 6.238325 6.285998 6.249975 6.220590 6.238325 6.173786 6.167516
## [4,] 6.253829 6.242223 6.287859 6.246107 6.222576 6.248043 6.173786 6.171701
## [5,] 6.272877 6.244167 6.295266 6.263398 6.222576 6.238325 6.175867 6.167516
## [6,] 6.269096 6.246107 6.293419 6.261492 6.224558 6.242223 6.177944 6.173786
```

- Compute the average temperature per row (for every observation)

```
mean_per_row <- apply(boiler, 1, mean)
mean_per_row
```

```
##  [1] 503.250 504.875 507.375 508.500 510.750 511.250 507.625 507.000 510.375
## [10] 510.250 510.500 509.875 510.750 506.250 512.125 511.250 512.625 507.125
## [19] 503.000 511.750 507.250 507.500 510.750 509.000 512.000
```

- Compute the average per column

```
mean_per_col <- apply(boiler, 2, mean)
mean_per_col
```

```
##     t1     t2     t3     t4     t5     t6     t7     t8
## 525.00 513.56 538.92 521.68 503.80 512.44 478.72 477.24
```


1.2 The tapply function in R

The function tapply is helpful while dealing with categorical variables. It allows to apply a function to numeric data distributed across various categories.

Syntax:

```
tapply(X, INDEX, FUN)
```

- X is an R object, usually a vector
- INDEX is a list of one or more factors
- FUN is the function to be applied

Example *tips*
We use the tips data (package reshape)
A waiter collected information about the amount of tip he received over a period of a few months.

```
head(tips)
```

```
##   total_bill  tip    sex smoker day   time size
## 1      16.99 1.01 Female     No Sun Dinner    2
## 2      10.34 1.66   Male     No Sun Dinner    3
## 3      21.01 3.50   Male     No Sun Dinner    3
## 4      23.68 3.31   Male     No Sun Dinner    2
## 5      24.59 3.61 Female     No Sun Dinner    4
## 6      25.29 4.71   Male     No Sun Dinner    4
```

1. Compute the average amount of tip, based on the day of the week

```
?tapply
```

```
tapply(tips$tip, tips$day, mean)
```

```
##      Fri      Sat      Sun     Thur
## 2.734737 2.993103 3.255132 2.771452
```

Remark:
In case you work with missing data, you can use tapply(tips$tip, tips$day, mean, na.rm = TRUE).

2. Compute the average amount of tip, based on the day of the week and the time of the day.

```
tapply(tips$tip, list(tips$day, tips$time), mean)
```

```
##        Dinner    Lunch
## Fri  2.940000 2.382857
## Sat  2.993103       NA
## Sun  3.255132       NA
## Thur 3.000000 2.767705
```

Remark:
The tapply function does the same as the by and the aggregate function in R.

1.3 The lapply function in R

Syntax:

```
lapply(X, FUN)
```

- X is the input data which can be a list, a vector or a data frame.
- FUN is the function to be applied.

The functionlapply returns a list of the same length as X. The specified function FUN is only applicable through columns.

Example *tips*
Because `lapply` works on a list, we first create a list of the `tips` data frame by using the `split` function. By using the `split` function, we divide a vector into groups defined by a factor.
Here we divide the total amount of the tip by day of the week

```
list_tip <- split(tips$tip, tips$day)
list_tip
```

```
## $Fri
##  [1] 3.00 3.50 1.00 4.30 3.25 4.73 4.00 1.50 3.00 1.50 2.50 3.00 2.20 3.48 1.92
## [16] 3.00 1.58 2.50 2.00
##
## $Sat
##  [1]  3.35  4.08  2.75  2.23  7.58  3.18  2.34  2.00  2.00  4.30  3.00  1.45
## [13]  2.50  3.00  2.45  3.27  3.60  2.00  3.07  2.31  5.00  2.24  3.00  1.50
## [25]  1.76  6.73  3.21  2.00  1.98  3.76  2.64  3.15  2.47  1.00  2.01  2.09
## [37]  1.97  3.00  3.14  5.00  2.20  1.25  3.08  2.50  3.48  4.08  1.64  4.06
## [49]  4.29  3.76  4.00  3.00  1.00  1.61  2.00 10.00  3.16  3.41  3.00  2.03
## [61]  2.23  2.00  5.16  9.00  2.50  6.50  1.10  3.00  1.50  1.44  3.09  3.00
## [73]  2.72  2.88  2.00  3.00  3.39  1.47  3.00  1.25  1.00  1.17  4.67  5.92
## [85]  2.00  2.00  1.75
##
## $Sun
##  [1] 1.01 1.66 3.50 3.31 3.61 4.71 2.00 3.12 1.96 3.23 1.71 5.00 1.57 3.00 3.02
## [16] 3.92 1.67 3.71 3.50 2.54 3.06 1.32 5.60 3.00 5.00 6.00 2.05 3.00 2.50 2.60
## [31] 5.20 1.56 4.34 3.51 4.00 2.55 4.00 3.50 5.07 2.50 2.00 2.74 2.00 2.00 5.14
## [46] 5.00 3.75 2.61 2.00 3.50 2.50 2.00 2.00 3.00 3.48 2.24 4.50 5.15 3.18 4.00
## [61] 3.11 2.00 2.00 4.00 3.55 3.68 5.65 3.50 6.50 3.00 5.00 3.50 2.00 3.50 4.00
## [76] 1.50
##
## $Thur
##  [1] 4.00 3.00 2.71 3.00 3.40 1.83 5.00 2.03 5.17 2.00 4.00 5.85 3.00 1.50 1.80
## [16] 2.92 2.31 1.68 2.50 2.00 2.52 4.20 1.48 2.00 2.00 2.18 1.50 2.83 1.50 2.00
## [31] 3.25 1.25 2.00 2.00 2.00 2.75 3.50 6.70 5.00 5.00 2.30 1.50 1.36 1.63 1.73
## [46] 2.00 4.19 2.56 2.02 4.00 1.44 2.00 5.00 2.00 2.00 4.00 2.01 2.00 2.50 4.00
## [61] 3.23 3.00
```

Now we want to compute the average tip per day of the week on this list.

```
lapply(list_tip, mean)
```

```
## $Fri
## [1] 2.734737
##
## $Sat
## [1] 2.993103
##
## $Sun
## [1] 3.255132
##
## $Thur
## [1] 2.771452
```

Remark:
We have the same result as with the `tapply` function earlier.

2 Loops in R

Remark:
If you can omit an R loop, do not use loops!

2.1 For loop in R

Syntax:

```
for(val in sequence){statement}
```

Example *tips* 1
In this example, `tips` data is used.

```
head(tips, n=3)
```

```
##   total_bill  tip    sex smoker day   time size
## 1      16.99 1.01 Female     No Sun Dinner    2
## 2      10.34 1.66   Male     No Sun Dinner    3
## 3      21.01 3.50   Male     No Sun Dinner    3
```

Count the number of reservations with even number of people at the table. Use the variable `size`.

```
sum_even <- 0
for (val in tips$size) {
  if (val %% 2 == 0) sum_even = sum_even + 1
}
# %% is modulo division in R
sum_even
```

```
## [1] 197
```

Example *tips* 2
In this example, `tips` data is used.
Assume that we want to create a new variable which is the total of the bill and the tip.

The best way to do this in R is

```
tips$total <- tips$total_bill + tips$tip
head(tips$total)
```

```
## [1] 18.00 12.00 24.51 26.99 28.20 30.00
```

Another way to do it with a for loop is

```
tips$total <- 0
for (i in 1:length(tips$total_bill)) {
  tips$total[i] <- tips$total_bill[i] + tips$tip[i]
}
head(tips)
```

```
##   total_bill  tip    sex smoker day   time size total
## 1      16.99 1.01 Female     No Sun Dinner    2 18.00
## 2      10.34 1.66   Male     No Sun Dinner    3 12.00
## 3      21.01 3.50   Male     No Sun Dinner    3 24.51
## 4      23.68 3.31   Male     No Sun Dinner    2 26.99
## 5      24.59 3.61 Female     No Sun Dinner    4 28.20
## 6      25.29 4.71   Male     No Sun Dinner    4 30.00
```

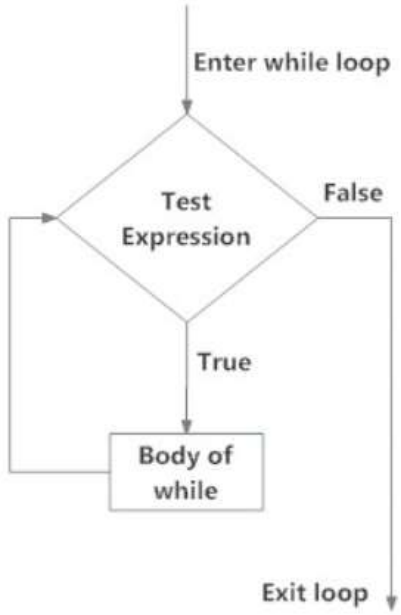
Remark:

1. The number of iterations in a for loop is fixed and known in advance.
2. If you can avoid loops, then do not use loops.

2.2 While loop in R

Syntax:

```
while(test_expression){statement}
```



Example 1
We want to print the values from 2 to 6

```
i <- 1
while(i<6){
  i = i+1
  print(i)
}
```

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

Example 2
`break` will end the loop abruptly.


```
i <- 1
while(i<6){
  i = i+1
  if (i==4) break
  print(i)
}
```

```
## [1] 2
## [1] 3
```

Example 3
`next` can skip one step of the loop

```
i <- 1
while(i<6){
  i = i+1
  if (i==4) next
  print(i)
}
```

```
## [1] 2
## [1] 3
## [1] 5
## [1] 6
```

2.3 Repeat loop in R

Syntax:

```
repeat{statement}
```

Example

```
i <- 1
repeat{
  print(i)
  i = i+1
  if (i==6){break}
}
```

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

Remark:
Be careful with the `repeat` function, ensure that there is a termination of the loop otherwise you might have an infinite loop.

3 *If then else* in R

Syntax:

```
if(test_expression){statement}
```

```
if(test_expression){statement1} else {statement2}
```

Example 1

```r
x <- -5
if(x>0){
  print("Non-negative number")
} else {
  print("Negative number")
}
```

```
## [1] "Negative number"
```

4 Examples of writing functions in R

4.1 General example

Write a function `Fn(vec)` which computes a vector of moving averages of width 3. $\frac{x_1+x_2+x_3}{3}$, $\frac{x_2+x_3+x_4}{3}$, ..., $\frac{x_{n-2}+x_{n-1}+x_n}{3}$.

When your original vector has length n, then the vector of moving averages will have length $n-2$.
Apply your function on the vector `1:6`.

```r
Fn <- function(vec)
{
  for(i in 3:length(vec))
  {x[i] <- (vec[i-2] + vec[i-1] + vec[i])/3
  print(x[i])}
}
z <- 1:6
Fn(z)
```

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

4.2 Illustration of the Central Limit Theorem (use of `for` loop and `apply` functions)

4.2.1 Description of the CLT illustration

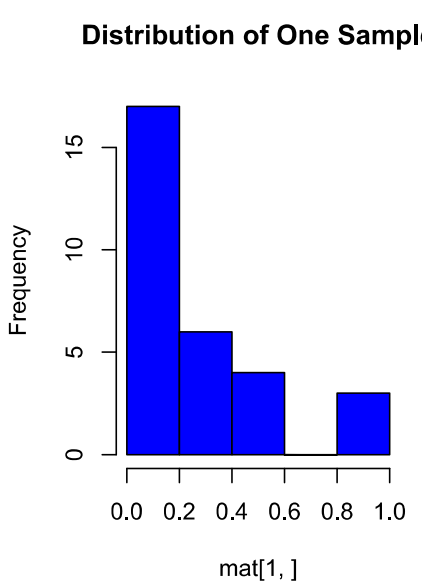
Make a visualization of the central limit theorem (CLT).
The **CLT** says that if you take samples from a distribution and compute the average. Then these sample averages have a normal distribution, no matter the distribution of the original population as long as the sample size is large enough.

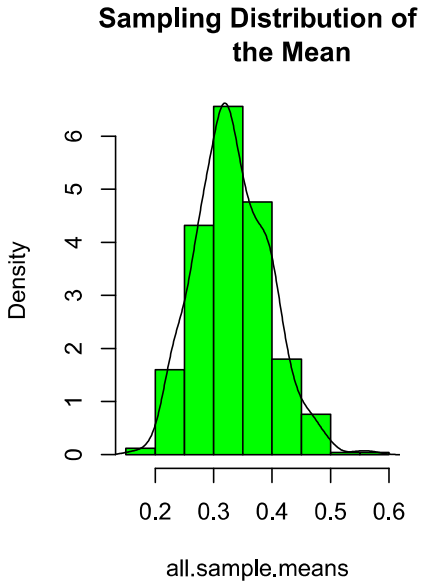
- Step 1: Generate 30 (=`n`) data points from an *exponential distribution* with rate 3. (hint: use the function `rexp`).
- Step 2: Do this now 5 times. Consider every sample as one new line in a data matrix `mat`. Hence `mat` will be a 5×30 matrix.
- Step 3: Compute a vector `all.samples.means` which has the averages for every sample. This vector has a length of 5.
- Step 4: Make two histograms next to each other (in one and the same graphical window).
  a) The first histogram is a frequency histogram of the first sample (the data from step 1).
  b) The second histogram is a relative frequency histogram of the sample averages, overlayed with the corresponding density curve.
- Step 5: Create now a function which is producing the previous steps (2-4) and has as parameters: `n` (number of data points, default 30), `rpt` (number of samples to take, default 5).
- Step 6: Apply this function when `n = 30` and `rpt = 500`.

4.2.2 Solution of the CLT illustration

Generate a matrix with rows and columns each row is one sample. One sample consists of 30 data points generated from an exponential distribution with rate 3.

```r
# Step 1
rexp(30, rate=3)

# Step 2
mat <- matrix(rep(0,150), nrow=5)
for (i in (1:5))
{
mat[i,] <- rexp(30, rate=3)
}

# Step 3
# compute the average for every sample
all.sample.means <- apply(mat,1,mean)

# Step 4
# create a histogram with the original data of 1st row
#and another histogram with the averages
par(mfrow=c(1,2))
hist(mat[1,],col="blue", main="Distribution of One Sample")
hist(all.sample.means, col="green", main="Sampling Distribution of
     the Mean", prob=T)
lines(density(all.sample.means))


# Step 5
# create a function out of this
clt_fun <- function(rpt = 5, n=30)
{
  mat <- matrix(rep(0,n*rpt), nrow=rpt)
    for (i in (1:rpt))
        {
      mat[i,] <- rexp(n, rate=3)
        }

  # compute the average for every sample
  all.sample.means <- apply(mat,1,mean)

  # create a histogram with the original data for 1st row
  #and another histogram with the averages
  par(mfrow=c(1,2))
  hist(mat[1,],col="blue", main="Distribution of One Sample")
  hist(all.sample.means, col="green", main="Sampling Distribution of
         the Mean", prob=T)
  lines(density(all.sample.means))
}

# Step 6
clt_fun(rpt=500,n=30)
```





4.3 Reuse tidyverse code

Example 1:

```
# example 1: compute grouped average delay for flights
 flights %>%
  group_by( month) %>%
  summarise(n=n(), Avg = mean(dep_delay, na.rm = T)) %>%
  arrange(Avg)


# example 2: compute average tip by day
tips %>%
  group_by(day) %>%
  summarise(n=n(), Avg = mean(tip)) %>%
  arrange(Avg)
```

A possible function can be written as :

```
group_mean <- function(data, var, by) {
  data %>%
    group_by(by) %>%
    summarise(average = mean(var, na.rm = TRUE)) %>%
    arrange(average)
}
group_mean(data=tips, var=tip, by=day)
```


The statement above is not working. We need to make some adaptations in the function. One possibility is to make use of the embracing operator (package rlang).

```
# we need to make some adaptations
library(rlang)

# use embracing operator {{ }}
group_mean2 <- function(data, var, by) {
  data %>%
    group_by({{ by }}) %>%
    summarise(average = mean({{ var }}, na.rm = TRUE))%>%
    arrange(average)
}
# now we can reuse dplyr code
group_mean2(data=tips, var=tip, by=day)
```

```
## # A tibble: 4 x 2
##   day   average
##   <fct>   <dbl>
## 1 Fri      2.73
## 2 Thur     2.77
## 3 Sat      2.99
## 4 Sun      3.26
```

5 Dates

Here, we use the functions from the `lubridate` package

```
library(lubridate)
library(nycflights13)
library(ggplot2)
```

5.1 Create date from strings

Current date:

```
today()
```

R starts counting from 1 January 1970
Internally, `Date` objects are stored as the number of days since January 1, 1970, using negative numbers for earlier dates.

This is a vector:

```
vec <- c("1970-01-01", "2020-01-31", "2020-02-01") # This is a vector
```

This is a date:

```
dates <- ymd(c("1970-01-01", "2020-01-31", "2020-02-01")) # This is a date
as.numeric(dates)
```

```
## [1]     0 18292 18293
```

5.2 Create date from individual components

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

To create a date from year, month and day:

```
flights_date <- flights %>%
  select(year, month, day) %>%
  mutate(dep_date = make_date(year, month, day))
head(flights_date)
```

```
## # A tibble: 6 x 4
##    year month   day dep_date
##   <int> <int> <int> <date>
## 1  2013     1     1 2013-01-01
## 2  2013     1     1 2013-01-01
## 3  2013     1     1 2013-01-01
## 4  2013     1     1 2013-01-01
## 5  2013     1     1 2013-01-01
## 6  2013     1     1 2013-01-01
```

`make_date` produces objects of class `Date`.

5.3 How to compare to a fixed date?

We now want e.g. to select the flights with departure date on January 22, 2013.

```
sub <- flights_date %>%
  filter(dep_date == ymd(20130122))
head(sub)
```

```
## # A tibble: 6 x 4
##    year month   day dep_date
##   <int> <int> <int> <date>
## 1  2013     1    22 2013-01-22
## 2  2013     1    22 2013-01-22
## 3  2013     1    22 2013-01-22
## 4  2013     1    22 2013-01-22
## 5  2013     1    22 2013-01-22
## 6  2013     1    22 2013-01-22
```

5.4 Once you have a date, you can get components

```
head(economics, n=3)
```

```
## # A tibble: 3 x 6
##   date         pce    pop psavert uempmed unemploy
##   <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
```



```
## 1 1967-07-01  507. 198712    12.6     4.5     2944
## 2 1967-08-01  510. 198911    12.6     4.7     2945
## 3 1967-09-01  516. 199113    11.9     4.6     2958
```

- `year()` and `month()` returns respectively the years and month component of a date-time as a decimal number.
- `mday()` returns the day of the month.
- `wday()` returns the day of the week as a decimal number or an ordered factor if label is `TRUE`.

```
econ <- economics
econ2 <- econ %>%
  select(date) %>%
  mutate(date_Y = year(date), date_M = month(date), date_D = mday(date),
         date_wkd = wday(date, label = TRUE))
head(econ2)
```

```
## # A tibble: 6 x 5
##   date       date_Y date_M date_D date_wkd
##   <date>      <dbl>  <dbl>  <int> <ord>
## 1 1967-07-01   1967      7      1 za
## 2 1967-08-01   1967      8      1 di
## 3 1967-09-01   1967      9      1 vr
## 4 1967-10-01   1967     10      1 zo
## 5 1967-11-01   1967     11      1 wo
## 6 1967-12-01   1967     12      1 vr
```

6 Spreading and gathering

6.1 Gathering: make a long table

The function `pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns.

```
# Usage of pivot_longer (see help page):
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
    ...
)
```

```
library(tidyr)
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```
table_long <- pivot_longer(table4a, 2:3, names_to = "year", values_to = "cases")
table_long
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

6.2 Spreading: make a wide table

`pivot_wider()` "widens" data, increasing the number of columns and decreasing the number of rows.

```
# Usage of pivot_wider (see help page):
pivot_wider(
  data,
  id_cols = NULL,
  names_from = name,
  values_from = value,
    ...
)
```

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```
table_wide <- pivot_wider(table2, id_cols=1,names_from=c(year,type),  values_from=count)
table_wide
```

```
## # A tibble: 3 x 5
##   country     `1999_cases` `1999_population` `2000_cases` `2000_population`
##   <chr>              <int>             <int>        <int>             <int>
## 1 Afghanistan          745          19987071         2666          20595360
## 2 Brazil             37737         172006362        80488         174504898
## 3 China             212258        1272915272       213766        1280428583
```

7 Exercises

7.1 Exercise 1

Write a method with a **while loop** to print `nr 1` through `nr n-1`. For example if `n = 6`, we have:


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

7.2 Exercise 2

Write a method with a **while loop** that computes the sum of first n positive integers:
sum = 1 + 2 + 3 + ... + n
Example: for n = 5, sum = 15

7.3 Exercise 3

Use a **nested while loop** to produce the following output

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

7.4 Exercise 4

(1) Use the **data `billboard`** from package `tidyr` for this exercise. This data set contains the song rankings for billboard top 100 in the year 2000.

- `date.enter` is the date the song entered the top 100
- `wk1`, `wk2`, ..., `wk76` is the rank of the song in each week after it entered the top 100
- `artist` and `track` are respectively the artist name and song name.

a. Create the variables `Year`, `Month` and `Day_nr` that correspond to the year, month and day of the month of the entering date. Select from `billboard_date` only the created variables and the variables `artist`, `wk1`, `wk2`, `wk3` and `wk4`. The name of the new data set is `billboard_date`.

```
head(billboard_date, n=8)
```

```
## # A tibble: 8 x 8
##   artist         wk1   wk2   wk3   wk4  Year Month Day_nr
##   <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <int>
## 1 2 Pac           87    82    72    77  2000     2     26
## 2 2Ge+her         91    87    92    NA  2000     9      2
## 3 3 Doors Down    81    70    68    67  2000     4      8
## 4 3 Doors Down    76    76    72    69  2000    10     21
## 5 504 Boyz        57    34    25    17  2000     4     15
## 6 98^0            51    39    34    26  2000     8     19
## 7 A*Teens         97    97    96    95  2000     7      8
## 8 Aaliyah         84    62    51    41  2000     1     29
```

b. Create from `billboard_date` a data set `b_billboard`that looks as follows. Compare the dimensions of `billboard_date` and `b_billboard`.

```
head(b_billboard, n=10)
```

```
## # A tibble: 10 x 6
##    artist        Year Month Day_nr Week   Rank
```

```
##    <chr>        <dbl> <dbl>  <int> <chr> <dbl>
##  1 2 Pac         2000     2     26 wk1      87
##  2 2 Pac         2000     2     26 wk2      82
##  3 2 Pac         2000     2     26 wk3      72
##  4 2 Pac         2000     2     26 wk4      77
##  5 2Ge+her       2000     9      2 wk1      91
##  6 2Ge+her       2000     9      2 wk2      87
##  7 2Ge+her       2000     9      2 wk3      92
##  8 2Ge+her       2000     9      2 wk4      NA
##  9 3 Doors Down  2000     4      8 wk1      81
## 10 3 Doors Down  2000     4      8 wk2      70
```

(2) Use the **data `us_rent_income`** from package `tidyr`. Make this data set more wide by increasing the number of columns. Both the column `estimate` and the column `moe` should have a separate column for each possible level of the column `variable`. The new data set should look like this:

```
## # A tibble: 10 x 6
##    GEOID NAME                 estimate_income estimate_rent moe_income moe_rent
##    <chr> <chr>                          <dbl>         <dbl>      <dbl>    <dbl>
##  1 01    Alabama                        24476           747        136        3
##  2 02    Alaska                         32940          1200        508       13
##  3 04    Arizona                        27517           972        148        4
##  4 05    Arkansas                       23789           709        165        5
##  5 06    California                     29454          1358        109        3
##  6 08    Colorado                       32401          1125        109        5
##  7 09    Connecticut                    35326          1123        195        5
##  8 10    Delaware                       31560          1076        247       10
##  9 11    District of Columbia           43198          1424        681       17
## 10 12    Florida                        25952          1077         70        3
```

7.5 Exercise 5

Use the `boiler` data frame of the `qcc` package. We have 25 time points ($i = 1, 2, ..., 25$) and at every time point we observe one measurement `t1` (this is `xi`). (We do not use the other variables `t2`, `t3`, ... `t8`). We are going to construct a moving range and individual chart for this data in the following way.
Here some background information:

Moving range chart

- What is plotted? $mr_i = |x_i - x_{i-1}|, for\ i = 2, 3, ...$
- What is center line? $\overline{mr}$ = average of the mr_i for the first 20 time points
- Control limits: $LCL = D_3 \cdot \overline{mr}$ and $UCL = D_4 \cdot \overline{mr}$, with $D_3 = 0$ and $D_4 = 3.267$

Individual chart

- What is plotted? x_i = measurement at time point i
- What is center line? $\overline{x}$ = sample mean of the measurements for the 20 first time points.
- Control limits: $LCL = \overline{x} - E_2 \cdot \overline{mr}$ and $UCL = \overline{x} + E_2 \cdot \overline{mr}$ with $E_2 = 2.66$

Questions

a) Create `total` data frame with three columns as is given below: time point, measurement x and moving range mr.

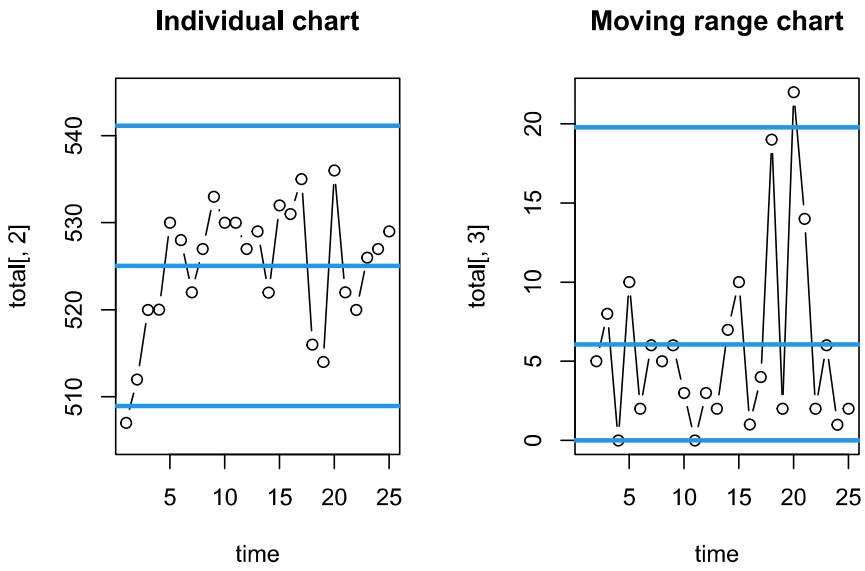
```
head(total, n=6)
```

```
##      time   x mr
## [1,]    1 507 NA
## [2,]    2 512  5
## [3,]    3 520  8
## [4,]    4 520  0
## [5,]    5 530 10
## [6,]    6 528  2
```

b) Compute $\overline{mr}$, the average of the mr for the first 20 time points, and $\overline{x}$, the average measurement for the first 20 time points.

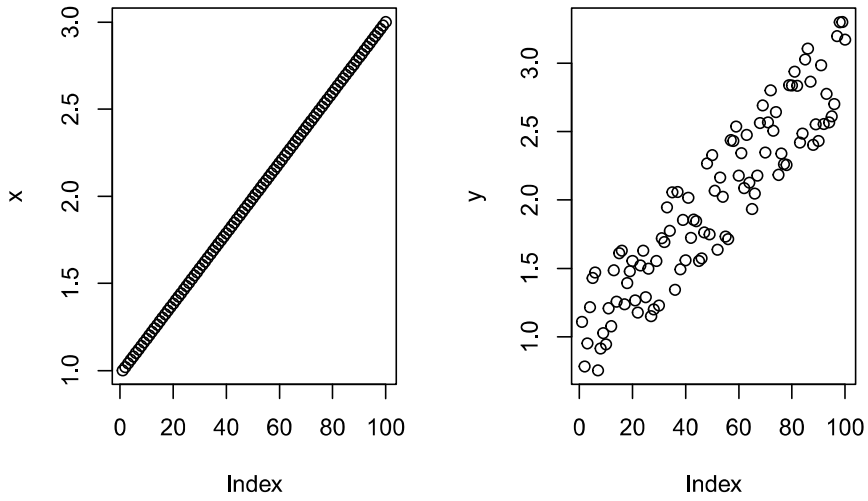
c) Compute the corresponding control limits by using the above formulas.

d) Make the moving range plot and the individual measurement plot as given below:

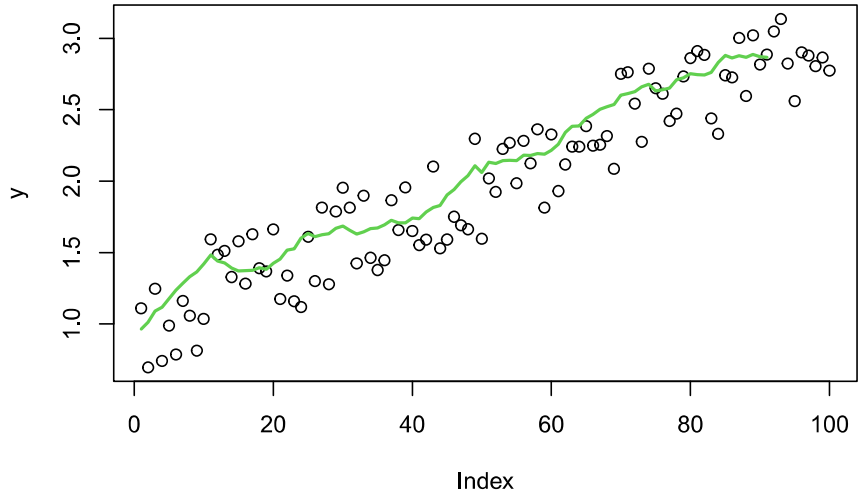


7.6 Exercise on rolling average to detect trends in your data

- Step 1: Generate sequence of data between 1 and 3 of total length 100. Use the `jitter` function (with a large factor) to add noise to your data. This is the vector y.



- Step 2: Compute the vector of rolling averages `roll.mean` with the average of 5 consecutive points. This vector has only 96 averages.
- Step 3: Add the vector of these averages to your plot.
- Step 4: Generalize step 2 and step 3 by making a function with parameters `consec` (default = 5) and `y`.
- Step 5: Apply your function to rolling averages of 10 consecutive points.



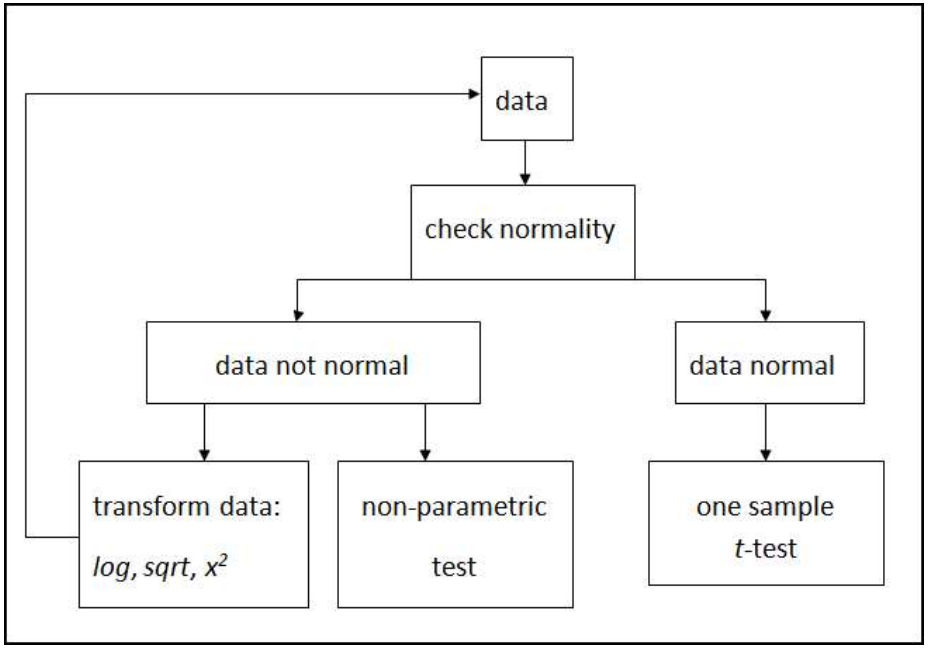
Chapter 8: Statistical inference for continuous data

Contents

1 One sample

The scheme of the analysis is as follows:

1. When sample size is large enough, we can use the CLT which assures that the average is normally distributed. In this situation, the one sample t-test can be used.
2. When sample size is small, then we have to use the scheme below. Check for normality (e.g. Shapiro-Wilk test). If normality is not rejected, we can use the one sample t-test. When normality is rejected, a non-parametric alternative or a transformation can be used.



We are using the data set `normtemp` from the package `UsingR`.

normtemp {UsingR} R Documentation

Body temperature and heart rate of 130 health individuals

Description

A data set used to investigate the claim that "normal" temperature is 98.6 degrees.

Usage

```
data(normtemp)
```

Format

A data frame with 130 observations on the following 3 variables.

temperature

normal body temperature

gender

Gender 1 = male, 2 = female

hr

Resting heart rate

```
head(normtemp)
```

```
##   temperature gender hr
## 1        96.3      1 70
## 2        96.7      1 71
## 3        96.9      1 74
## 4        97.0      1 80
## 5        97.1      1 73
## 6        97.1      1 75
```

We want to test the following hypothesis:

$H_0 : \mu_{temp} = 100$
$H_1 : \mu_{temp} \neq 100$

Descriptive statistics

```
library(rstatix)
summary_result <- normtemp %>%
  get_summary_stats(temperature, hr, show = c("n", "mean", "sd", "median"))
summary_result
```

```
## # A tibble: 2 x 5
##   variable        n  mean    sd median
##   <chr>       <dbl> <dbl> <dbl>  <dbl>
## 1 hr            130  73.8 7.06    74
## 2 temperature   130  98.2 0.733   98.3
```


1.1 One sample t-test

If the sample size is large enough, the CLT can be applied and one sample t-test can be used.

```
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

There are 3 options:

`conf.level`	Confidence level for CI (confidence interval)
`mu`	population mean under the null hypothesis
`alternative`	alternative hypothesis: can be `two-sided`, `greater`, `less`

The sample size of the data `normtemp` is large ($n = 130$), hence CLT can be applied and the one sample t-test can be used to test the earlier mentioned hypothesis.

```
temp <- normtemp$temperature
t.test(temp, mu = 100)
```

```
##
##  One Sample t-test
##
## data:  temp
## t = -27.226, df = 129, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 100
## 95 percent confidence interval:
##  98.12200 98.37646
## sample estimates:
## mean of x
##  98.24923
```

The output returns `p-value < 2.2e-16` hence the null hypothesis H_0 is rejected and the average temperature is significant different from 100.

1.2 Non-parametric test

In case CLT cannot be used or the normality is not fulfilled, a non-parametric test can be applied

```
wilcox.test(x, y = NULL,
            alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
            conf.int = FALSE, conf.level = 0.95,
            tol.root = 1e-4, digits.rank = Inf, ...)
```

By using the Wilcoxon signed rank test (for one sample) you can check if the distribution of x is symmetric around `mu`.

```
wilcox.test(temp, mu = 100)
```

```
##
```

```
##  Wilcoxon signed rank test with continuity correction
##
## data:  temp
## V = 8.5, p-value < 2.2e-16
## alternative hypothesis: true location is not equal to 100
```

Remark:

1. You can also use the `t_test` in `rstatix`

```
# One sample t-test
normtemp %>% t_test(temperature ~ 1, mu = 100)
```

```
## # A tibble: 1 x 7
##   .y.         group1 group2         n statistic    df        p
## * <chr>       <chr>  <chr>      <int>     <dbl> <dbl>    <dbl>
## 1 temperature 1      null model   130     -27.2   129 2.54e-55
```

2. You can use the `wilcox_test` in `rstatix`

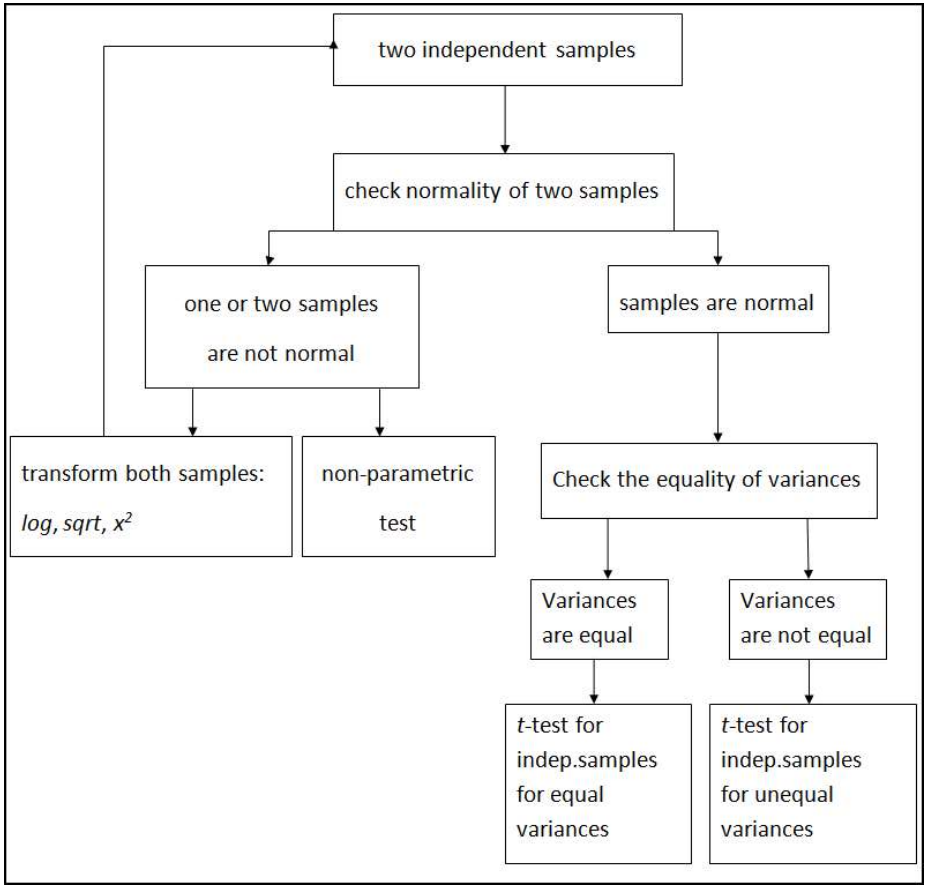
```
normtemp %>% wilcox_test(temperature ~ 1, mu = 100)
```

```
## # A tibble: 1 x 6
##   .y.         group1 group2         n statistic        p
## * <chr>       <chr>  <chr>      <int>     <dbl>    <dbl>
## 1 temperature 1      null model   130       8.5 7.74e-23
```

2 Two samples

2.1 Two sample t-test

1. When both sample sizes are large enough, we can use CLT and apply the two sample t-test.
2. When one (or both) have a small sample size, we have to test for normality and follow the scheme below.



We want to test whether the average body temperature for men is equal to the average body temperature for women. We first create two new vectors with the body temperature for men and women separately.

```
mentemp <- normtemp[normtemp$gender==1, 'temperature']
length(mentemp)
```

```
## [1] 65
```

```
womentemp <- normtemp[normtemp$gender==2, 'temperature']
length(womentemp)
```

```
## [1] 65
```

Here, both sample sizes are large enough to use the CLT. Below we show how to use the Shapiro Wilk test in case sample sizes are small.

Ask for some descriptive statistics:

```r
# Summary statistics by gender
gender_result <- normtemp %>%
  group_by(gender) %>%
  get_summary_stats(temperature, show = c("n", "mean", "sd", "median"))
gender_result
```

```
## # A tibble: 2 x 6
##   gender variable        n  mean    sd median
##    <int> <chr>       <dbl> <dbl> <dbl>  <dbl>
## 1      1 temperature    65  98.1 0.699   98.1
## 2      2 temperature    65  98.4 0.743   98.4
```

2.1.1 Test normality in both groups

Test normality in both groups is not necessary in this example, but it is given for illustrative purposes.

H_0 : **distribution of the data is normal**
H_1 : **distribution of the data is not normal**

2.1.1.1 Shapiro-Wilk test: `shapiro.test` test normality in both groups

```r
shapiro.test(mentemp)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mentemp
## W = 0.98941, p-value = 0.8545
```

```r
shapiro.test(womentemp)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  womentemp
## W = 0.96797, p-value = 0.09017
```

The p-value is large in both groups. There is no evidence that the data is not normally distributed.

2.1.2 Test equality of variances: `var.test`

$H_0 : \sigma^2_{men} = \sigma^2_{women}$
$H_1 : \sigma^2_{men} \neq \sigma^2_{women}$

Test if variances are equal

```r
var.test(mentemp, womentemp)
```

```
##
##  F test to compare two variances
##
## data:  mentemp and womentemp
## F = 0.88329, num df = 64, denom df = 64, p-value = 0.6211
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.5387604 1.4481404
## sample estimates:
## ratio of variances
```



```
##           0.8832897
```

Another way to test if variances are equal

```r
var.test(normtemp$temperature ~ normtemp$gender)
```

```
##
##  F test to compare two variances
##
## data:  normtemp$temperature by normtemp$gender
## F = 0.88329, num df = 64, denom df = 64, p-value = 0.6211
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.5387604 1.4481404
## sample estimates:
## ratio of variances
##           0.8832897
```

2.1.3 Test equality of means

$H_0 : \mu_{men} = \mu_{women}$
$H_1 : \mu_{men} \neq \mu_{women}$

```r
t.test(mentemp, womentemp, var.equal = TRUE)
```

2.1.3.1 T-test when equal variances can be assumed

```
##
##  Two Sample t-test
##
## data:  mentemp and womentemp
## t = -2.2854, df = 128, p-value = 0.02393
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.53963938 -0.03882216
## sample estimates:
## mean of x mean of y
##  98.10462  98.39385
```

Another possibility is:

```r
t.test(normtemp$temperature ~ normtemp$gender, var.equal = T)
```

```r
t.test(mentemp, womentemp, var.equal = FALSE)
```

2.1.3.2 In case the variances are not equal, apply the t-test for unequal variances

```
##
##  Welch Two Sample t-test
##
## data:  mentemp and womentemp
## t = -2.2854, df = 127.51, p-value = 0.02394
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.53964856 -0.03881298
## sample estimates:
```

```
## mean of x mean of y
##  98.10462  98.39385
```

```
wilcox.test(mentemp, womentemp)
```

2.1.3.3 A non-parametric alternative

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  mentemp and womentemp
## W = 1637, p-value = 0.02676
## alternative hypothesis: true location shift is not equal to 0
```

Remark:
We can also use functions of the `rstatix` package

2.1.3.4 Functions of `rstatix` package[1]

```
normtemp %>%
  group_by(gender) %>%
  shapiro_test(temperature)
```

2.1.3.4.1 To test normality

```
## # A tibble: 2 x 4
##   gender variable    statistic      p
##    <int> <chr>           <dbl>  <dbl>
## 1      1 temperature     0.989 0.855
## 2      2 temperature     0.968 0.0902
```

```
normtemp %>%
  levene_test(temperature ~ as.factor(gender))
```

2.1.3.4.2 Test homogeneity of variances

```
## # A tibble: 1 x 4
##     df1   df2 statistic     p
##   <int> <int>     <dbl> <dbl>
## 1     1   128    0.0635 0.801
```

2.1.3.4.3 Two sample t-test Two sample t-test, assuming equal variances

```
normtemp %>%
  t_test(temperature ~ gender, var.equal = T)
```

```
## # A tibble: 1 x 8
##   .y.         group1 group2    n1    n2 statistic    df      p
## * <chr>       <chr>  <chr>  <int> <int>     <dbl> <dbl>  <dbl>
## 1 temperature 1      2         65    65     -2.29   128 0.0239
```

[1] https://cran.r-project.org/web/packages/rstatix/readme/README.html#:~:text=rstatix,Kruskal%2DWallis%20and%20correlation%20analyses

2.1.3.4.4 A non-parametric alternative Wilcoxon non-parametric test

```
normtemp %>% wilcox_test(temperature ~ gender)
```

```
## # A tibble: 1 x 7
##   .y.         group1 group2    n1    n2 statistic      p
## * <chr>       <chr>  <chr>  <int> <int>     <dbl>  <dbl>
## 1 temperature 1      2         65    65      1637 0.0268
```

2.2 Correlation analysis

To estimate the **linear** association between two continuous variables: function `cor_test()` or `cor_mat()`.

- `cor_test()`: correlation test between two or more variables using Pearson, Spearman or Kendall methods.
- `cor_mat()`: compute correlation matrix with p-values (in case you have more than 2 numeric variables). Returns a data frame containing the matrix of the correlation coefficients. The output has an attribute named `pvalue`, which contains the matrix of the correlation test p-values.

To test for significant association

$H_0 : \rho = 0$
versus
$H_1 : \rho \neq 0$

Correlation analysis

```
normtemp %>% cor_test(temperature, hr)
```

```
## # A tibble: 1 x 8
##   var1        var2    cor statistic       p conf.low conf.high method
##   <chr>       <chr> <dbl>     <dbl>   <dbl>    <dbl>     <dbl> <chr>
## 1 temperature hr     0.25      2.97 0.00359   0.0852     0.408 Pearson
```

In case we want to use the non-parametric Spearman correlation

```
normtemp %>% cor_test(temperature, hr, method = "spearman")
```

```
## # A tibble: 1 x 6
##   var1        var2    cor statistic       p method
##   <chr>       <chr> <dbl>     <dbl>   <dbl> <chr>
## 1 temperature hr     0.28   263288. 0.00121 Spearman
```

3 Exercises

1. Use the `chol` data set. (import the *chol.txt* file from Toledo)
   a) Create a new variable `group` based on the variable `SMOKE` of the `chol` data set. The variable `group` has two possible values `nonsmoke` or `smoke`. In the group `smoke` we have the sigare and pipe smokers.
   Hint: Use the function `ifelse()` to create a new variable with the value `nonsmoke`/`smoke`.
   b) Make a grouped boxplot of `chol` value by `group`.
2. Generate, on the same graphical window, tho histograms for the `CHOL` values of these groups.
3. Use the data set `chol` to detect if there is a significant difference in average cholesterol (`CHOL`) between the two groups `smoke` and `nonsmoke`. Give comment on the methods you are using.

Chapter 9: Statistical inference for discrete data

Contents

1 Testing independence

1.1 Raw data: use table + chisq.test

Example *chol*
In order to test whether there is an association between the smoke behavior and the mortality, we use the chol data. Because we work here with raw data, we first have to construct a contingency table by the table function.

Importing the data: (data is available on Toledo)

```
Chol <- read.table(file=file.choose(), header=TRUE)
```

```
head(Chol, n=4)
```

```
## # A tibble: 4 x 7
##     AGE HEIGHT WEIGHT  CHOL SMOKE  BLOOD MORT
##   <dbl>  <dbl>  <dbl> <dbl> <chr>  <chr> <chr>
## 1    20    176     77   195 nonsmo b     alive
## 2    53    167     56   250 sigare o     dead
## 3    44    170     80   304 sigare a     dead
## 4    37    173     89   178 nonsmo o     alive
```

```
names(Chol)
```

```
## [1] "AGE"    "HEIGHT" "WEIGHT" "CHOL"   "SMOKE"  "BLOOD"  "MORT"
```

Null hypothesis:

H_0 : **mortality and smoke behavior is independent**

Alternative hypothesis:

H_1 : **There is an association between mortality and smoke behavior**

First constructing a contingency table:


```
tab.chol <- table(Chol$SMOKE, Chol$MORT)
tab.chol
```

```
##
##          alive dead
##   nonsmo    45    4
##   pipe      38    4
##   sigare    93   16
```

```
chisq.test(tab.chol)
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab.chol
## X-squared = 1.6677, df = 2, p-value = 0.4344
```

1.2 Summary data: use xtabs and chisq.test

In case you have summarized data, you work with a weight variable (often n).

By using the xtabs() function, you can use weight variables

```
xtabs(Freq ~ var1 + var2, DF)
```

Example *seatbelt*
In order to test whether wearing seat belts prevents for having fatal accidents we have following data *seatbelt.txt*.

Import seatbelts.txt

```
head(seatbelts)
```

```
##   seatbelts fatal   n
## 1       yes   yes   7
## 2       yes    no  89
## 3        no   yes  24
## 4        no    no 122
```

```
table <- xtabs(seatbelts$n ~ seatbelts$seatbelts + seatbelts$fatal)
table
```

```
##                    seatbelts$fatal
## seatbelts$seatbelts  no yes
##                 no  122  24
##                 yes  89   7
```

H_0 : **wearing seatbelts and having a fatal accident is independent**
versus
H_1 : **there is an association between wearing seatbelts and having a fatal accident**

```
chisq.test(table)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table
## X-squared = 3.558, df = 1, p-value = 0.05926
```

1.3 In case of very few observations

Example *tea*
tea-drinking lady: *tea.txt*

Someone claimed that, when drinking tea, she could distinguish whether milk or tea was added to the cup first. To test her claim, she tasted 8 cups of tea. 4 cups had milk added first, and the other had tea added first. The cups were presented in random order.

	Guess poured first	
Poured first	*milk*	*tea*
milk	3	1
tea	1	3

```
tea # after importing tea.txt
```

```
##   poured guess n
## 1   milk  milk 3
## 2   milk   tea 1
## 3    tea  milk 1
## 4    tea   tea 3
```

```
table <- xtabs(tea$n ~ tea$poured + tea$guess)
table
```

```
##           tea$guess
## tea$poured milk tea
##       milk    3   1
##       tea     1   3
```

```
fisher.test(table)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table
## p-value = 0.4857
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##    0.2117329 621.9337505
## sample estimates:
## odds ratio
##   6.408309
```

Remark:
In the package `rstatix`, you can as well use the functions `chisq_test()` and `fisher_test()`.

2 Some other functions for count data

Example

Two medications:

A: 121 deaths out of 1584 patients
B: 145 deaths out of 1998 patients

2.1 Test for two proportions

Test for two proportions: Do the two mortality rates differ significantly?

$H_0 : p_A = p_B$
versus
$H_1 : p_A \neq p_B$

```
trial.mort <- c(121, 145)
trial.siz <- c(1584, 1998)
prop.test(trial.mort, trial.siz)
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  trial.mort out of trial.siz
## X-squared = 0.13579, df = 1, p-value = 0.7125
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.01408481  0.02171744
## sample estimates:
##     prop 1     prop 2
## 0.07638889 0.07257257
```

2.2 Test for one proportion: `binom.test`

`binom.test` does not provide confidence intervals.
This function test hypotheses about the parameter p in a $Binomial(n, p)$ model given x, the number of successes out of n trials.

Syntax:

```
Binom.test(x, n, p)
```

Hypotheses:

$H_0 : p = 0.08$ versus $H_1 : p \neq 0.08$

```
binom.test(121, 1584, p = 0.08)
```

```
##
##  Exact binomial test
##
## data:  121 and 1584
## number of successes = 121, number of trials = 1584, p-value = 0.6432
## alternative hypothesis: true probability of success is not equal to 0.08
## 95 percent confidence interval:
##  0.06378725 0.09058599
## sample estimates:
## probability of success
##             0.07638889
```

Remark:
In the package `rstatix`, you can use the functions `prop_test()` and `binom_test()`.

3 Exercises

1. The data for this exercise can be found in *operations.txt*.

There are 4 groups of operations (A = Not serious, B, C, D = Very serious) and 3 groups of side effects (0 = No side effects, 1 = one side effect, 2 = two side effects).

Make a table of this cross-classified data in R.

```
##                      operations$side_effect
## operations$operation  0  1  2
##                    A 61 28  7
##                    B 68 23 13
##                    C 58 40 12
##                    D 53 38 16
```

Check if the row and column variables are independent of each other.

2. The data set used in this exercise is `tips` from the `reshape` package.

Make a table of day of the week and gender of the bill payer. Check whether these two variables are independent of each other.

Chapter 10: An example of a regression analysis

Contents

1 Regression analysis with usual R

```
lm(formula, data = ..., subset = ...)
```

General form of formula: **`response ~ expression`**

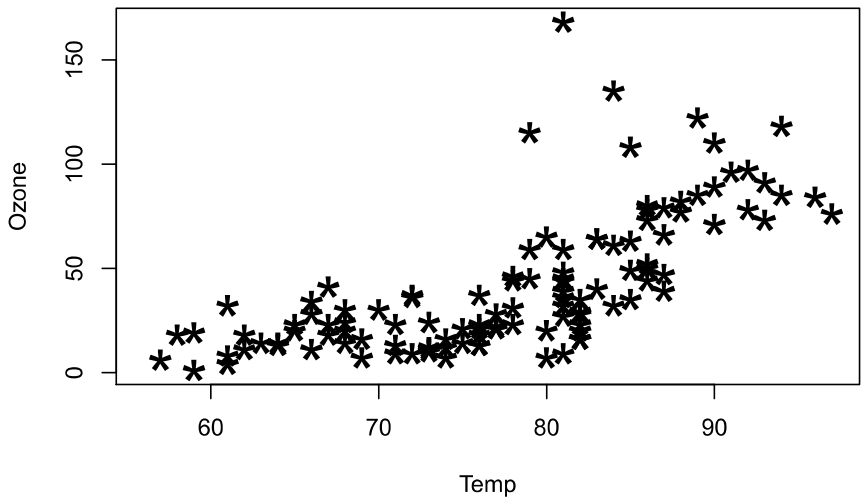
By default, the `lm()` function will print out the estimates for the coefficients. Much more is returned, but needs to be asked for.
Some useful extractors are:

- `summary()`
- `plot()`
- `coef()`
- `residuals()`
- `fitted()`
- `deviance()`
- `predict()`
- `anova()`
- `AIC()`

Example *chol*
We use the data set `airquality` from the package `datasets`.

We first make a scatterplot of `Ozone` versus `Temp`

```
plot(Ozone ~ Temp, data = airquality, pch = "*", cex = 3)
```



We perform a linear regression analysis (`lm(response ~ X)`).

```
res.lm <- lm(Ozone ~ Temp, data = airquality)
summary(res.lm)
```

```
##
## Call:
## lm(formula = Ozone ~ Temp, data = airquality)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -40.729 -17.409  -0.587  11.306 118.271
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -146.9955    18.2872  -8.038 9.37e-13 ***
## Temp           2.4287     0.2331  10.418  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.71 on 114 degrees of freedom
##   (37 observations deleted due to missingness)
## Multiple R-squared:  0.4877, Adjusted R-squared:  0.4832
## F-statistic: 108.5 on 1 and 114 DF,  p-value: < 2.2e-16
```

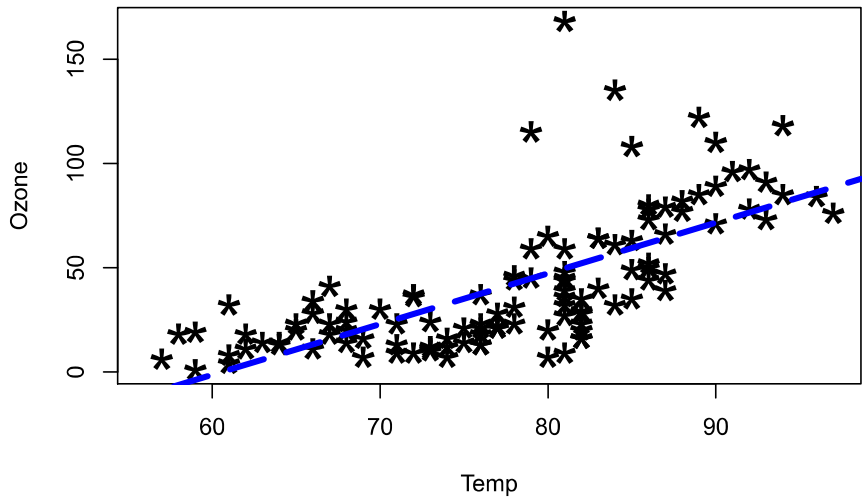
From the output:
Ozone = -147 + 2.43 Temp


```
names(res.lm)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "na.action"     "xlevels"       "call"          "terms"
## [13] "model"
```

We add the regression line to the plot

```
plot(Ozone ~ Temp, data = airquality, pch = "*", cex = 3)
abline(res.lm, lty = 5, col = 4, lwd = 4)
```



The argument **formula** in the function `lm` can take more complex forms than the example above. Some examples are

Formula regression model	Formula argument in R
$Z = \beta_0 + \beta_1 X + \beta_2 Y$	`Z ~ X + Y`
$Z = \beta_0 + \beta_1 X + \beta_2 X^2$	`Z ~ X + I(X^2)`
$Z = \beta_1 X$	`Z ~ X -1`

Note: Function `I()` is a conversion of object function. It means that the object X^2 should be treated as a new variable.
It can be done in another way: First, create a new variable $X_2 = X \cdot X$. Then use this variable in the regression model.

We will also try to apply a polynomial model to the data.

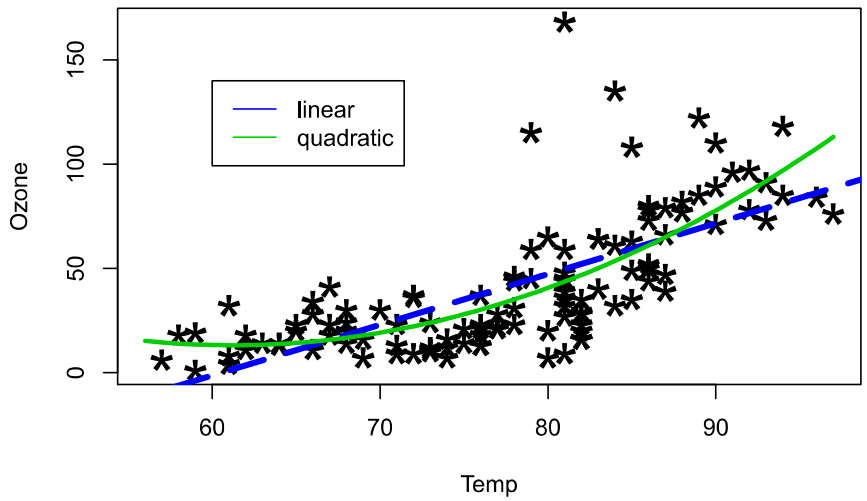
```
res.lm2 <- lm(Ozone ~ Temp + I(Temp^2), data = airquality)
summary(res.lm2)
```

```
##
## Call:
## lm(formula = Ozone ~ Temp + I(Temp^2), data = airquality)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.619 -12.513  -2.736   9.676 123.909
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 305.48577  122.12182   2.501 0.013800 *
## Temp         -9.55060    3.20805  -2.977 0.003561 **
## I(Temp^2)     0.07807    0.02086   3.743 0.000288 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.47 on 113 degrees of freedom
##   (37 observations deleted due to missingness)
## Multiple R-squared:  0.5442, Adjusted R-squared:  0.5362
## F-statistic: 67.46 on 2 and 113 DF,  p-value: < 2.2e-16
```

From the output:

Ozone = 305.49 - 9.55 Temp + 0.078 Temp^2

```
# One possibility
plot(Ozone ~ Temp, data = airquality, pch = "*", cex = 3)

# Add regression line
abline(res.lm, lty = 5, col = 4, lwd = 4)

# Add quadratic curve
curve(305.5 - 9.55*x + 0.078*x*x, add = T, col = 3, lwd = 3)

legend(60, 140, legend = c("linear", "quadratic"), lty = c(1,1), col = c(4,3))
```

Another possibility:

- *Step 1:*
  Compute the formula for polynomial regression

```
poly <- function(x, coefs)
{
  tot <- 0
  for (i in 1:length(coefs))
  {tot <- tot + coefs[i]*x^{i-1}}
  tot
}
```

```
coef(res.lm2)
```

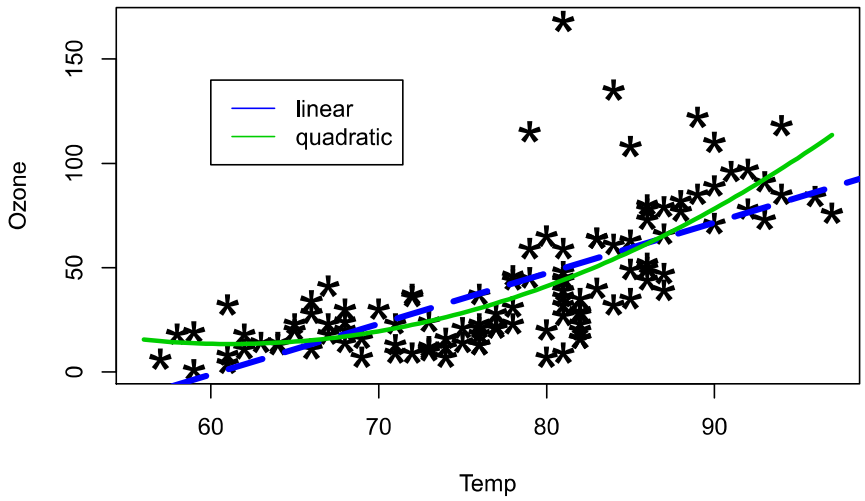
```
##  (Intercept)         Temp    I(Temp^2)
## 305.48576778  -9.55060391   0.07806798
```

What is happening in the `for` loop of `poly` for the polynomial regression model?

i $= 1 \rightarrow$	tot $= 0 + \text{coef}[1] \cdot 1$	$\rightarrow$ tot $= 305$
i $= 2 \rightarrow$	tot $= 305 + \text{coef}[2] \cdot x$	$\rightarrow$ tot $= 305 - 9.55 \cdot x$
i $= 3 \rightarrow$	tot $= 305 - 9.55 \cdot x + \text{coef}[3] \cdot x^2$	$\rightarrow$ tot $= 305 - 9.55 \cdot x + 0.078 \cdot x^2$

- *Step 2:*
  Produce the plot

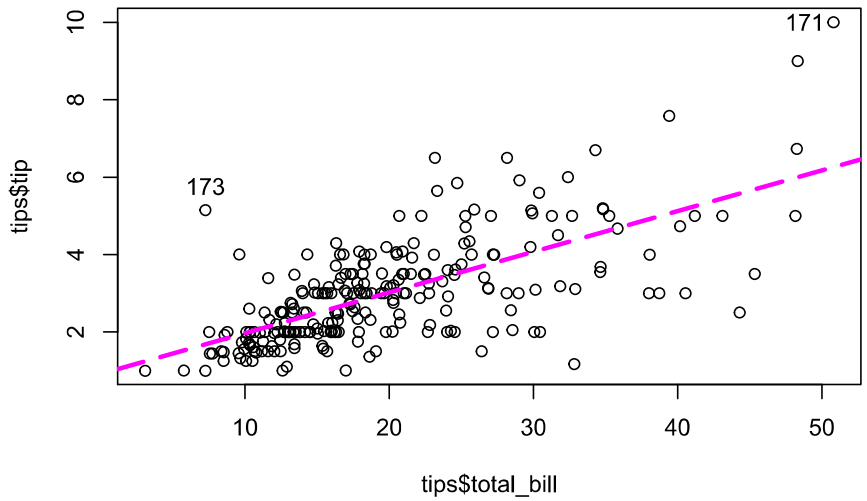
```r
# One possibility
plot(Ozone ~ Temp, data = airquality, pch = "*", cex = 3)

# Add regression line
abline(res.lm, lty = 5, col = 4, lwd = 4)

curve(poly(x, coef(res.lm2)), add = TRUE, col = 3, lwd = 3)

legend(60, 140, legend = c('linear', 'quadratic'), lty = c(1,1), col = c(4,3))
```



Remark:
The function `curve()` has an option to specify the function that should be plotted. In our case it is a function `poly(x, coef(res.lm2))`. In general, it can be any other function: `sin`, `cos`, `tan`

2 Exercises

Used data: `tips` from `reshape` package
Try to set up a regression model to predict the `tip` by `total_bill`. Add this regression line to the plot. Identify some special observations.



Chapter 11: Grammar of ggplot2

Contents

1 Introduction

1.1 What is ggplot2?

- Graphical package of R
- Works in layered fashion
- Each layer contains information about:
  - **Data**: should be data frame
  - **Mapping**: how your data corresponds to visual elements on your plot (=aesthetics).
  - **Geometric Objects**(points, lines...)
  - **Statistics**: how to summarize your data
  - **Facet**: how to break up the data

How to start?

```
install.packages("ggplot2")
library(ggplot2)
```

1.2 Documentation

ggplot2 documentation: http://ggplot2.org/

1.3 Grammar

Example *fish*
Data: *fish.xlsx* (Peck et all.)

The state of Maine conducted a field study of 115 lakes to characterize mercury levels in fish by measuring mercury (Hg) and other variables on lake characteristics.

This data frame contains information on the average mercury (Hg) level of fish in lakes in order to determine whether the fishes are safe to eat. The following variables are given:

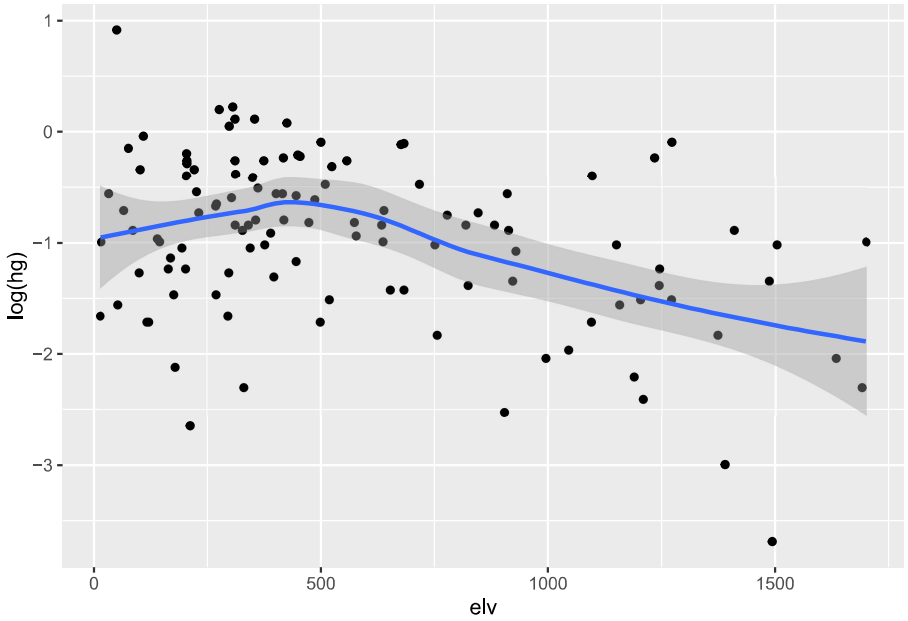
Name variable	Description
name	Name of the lake
hg	Mercury level, expressed in parts per million
number	Number of fish in the composite
elv	Elevation (feet)
sa	Surface area (acres)
z	Maximum depth (feet)
lt	Lake type: 1 = oligotrophic; 2 = eutrophic; 3 = mesotropic
st	Lake stratification indicator (1 = yes, 0 = no)
dam	Some lakes have a dam: 0 = the lake does not have a dam; 1 = the lake has a dam

```
head(fish)
```

```
## # A tibble: 6 x 15
##   name     hg number   elv    sa     z    lt    st   dam  lat1  lat2  lat3 long1
```

```
##   <chr> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ALLE~ 1.08       3   425    83    27     3     1     1    44    57    44    68
## 2 ALLI~ 0.025      2  1494    47    26     2     0     1    45    37    50    69
## 3 ANAS~ 0.57       5   402   568    54     2     1     0    44    25    13    70
## 4 BALC~ 0.77       5   557   704    44     2     1     0    43    37     0    70
## 5 BASK~ 0.79       5   417  6944    22     2     0     1    45    30    32    67
## 6 BAUN~ 0.75       4   205   200    29     2     1     0    43    21    46    70
## # ... with 2 more variables: long2 <dbl>, long3 <dbl>
```

```
ggplot(fish, aes(elv, log(hg))) + geom_point() + stat_smooth()
```



Layers:
Which part of this code belongs to which layer?

- `fish` → **Data**: should be data frame
- `aes(elv, log(hg))` → **Mapping**: tell R how your data corresponds to visual elements of your plot (= aesthetics). Use the function `aes()`.
- `geom_point()` → **Geom**etric Objects (point, lines...)
- `stat_smooth()` → **Stat**istics: how to summarize your data
- **Facet**: how to break up the data

2 Build a plot layer by layer by `ggplot()`

2.1 `ggplot()`

```
ggplot(data, mapping) + geom_XXX() + stat_YYY() + facet_ZZZ()
```

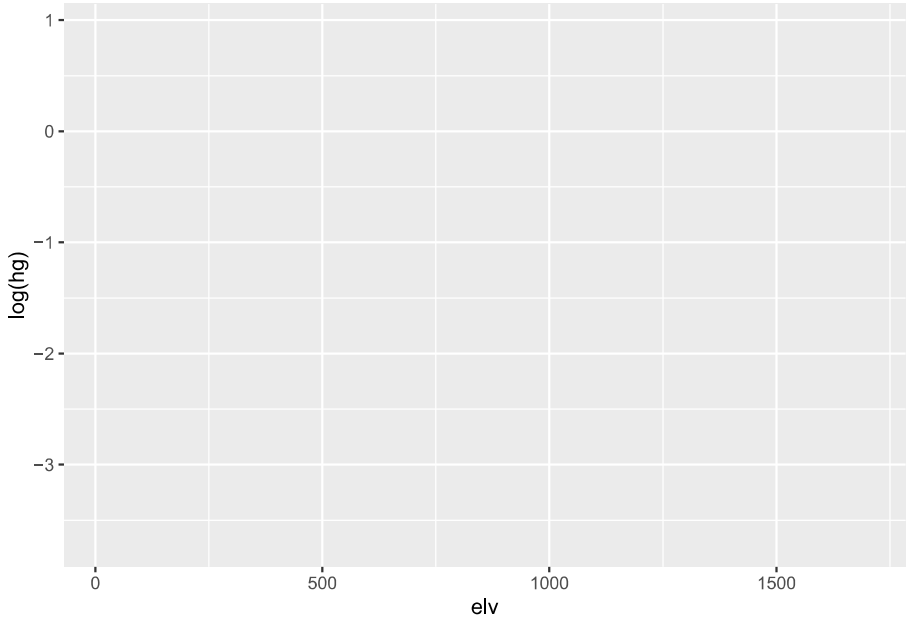

2.2 Data and mapping

1. **Data**: must be a data frame
2. **Mapping**: The aesthetic mappings describe the way that variables in the data are mapped to the plot. Therefore we use **aes** function.

Example *fish*
For data `fish`:

1. **Data** → `fish`
2. **Mapping** → `aes(x = elv, y = log(hg), colour = factor(dam))`

```
pl1 <- ggplot(fish, aes(x = elv, y = log(hg), colour = factor(dam)))
pl1
```



Remark:

1. Here, we map the *x position* to `elv`, the *y position* to `log(hg)` and *colour* to `dam`. The first two arguments can be left without names.
2. You should never refer to variables outside the data frame (e.g. `fish$hg`)
3. This code produces an empty plot! We have to add layers.

2.3 Layer Geoms

Layers define the basic "shape" of the elements on the plot. Layers can be added to plots created by `ggplot` or by `qplot`.

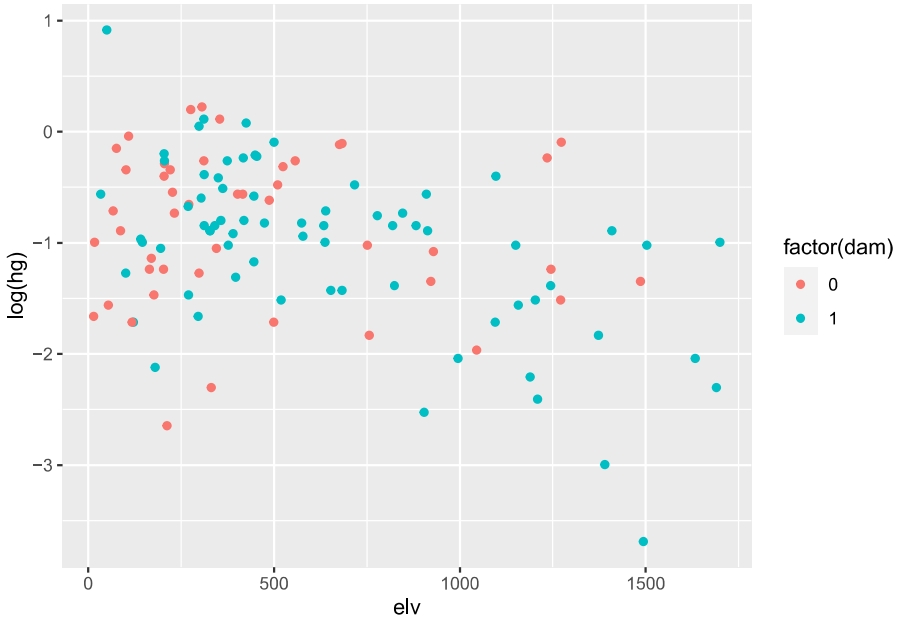
A geom defines the **layout** of a ggplot2 layer.

A selection of geoms and associated default stats:

geom	description	default stat
`geom_bar()`	Bar chart for categorical variable	`stat_bin()`
`geom_point()`	Scatterplot	`stat_identity()`
`geom_line()`	Line, connecting observations in ordered x value	`stat_identity()`
`geom_boxplot()`	Boxplot	`stat_boxplot()`
`geom_smooth()`	Fits a smoother to the data	`stat_smooth()`
`geom_histogram()`	Histogram for continuous variable	`stat_bin()`
`geom_density()`	Smooth density estimate	`stat_density()`

2.3.1 To create scatterplot: `geom_point`

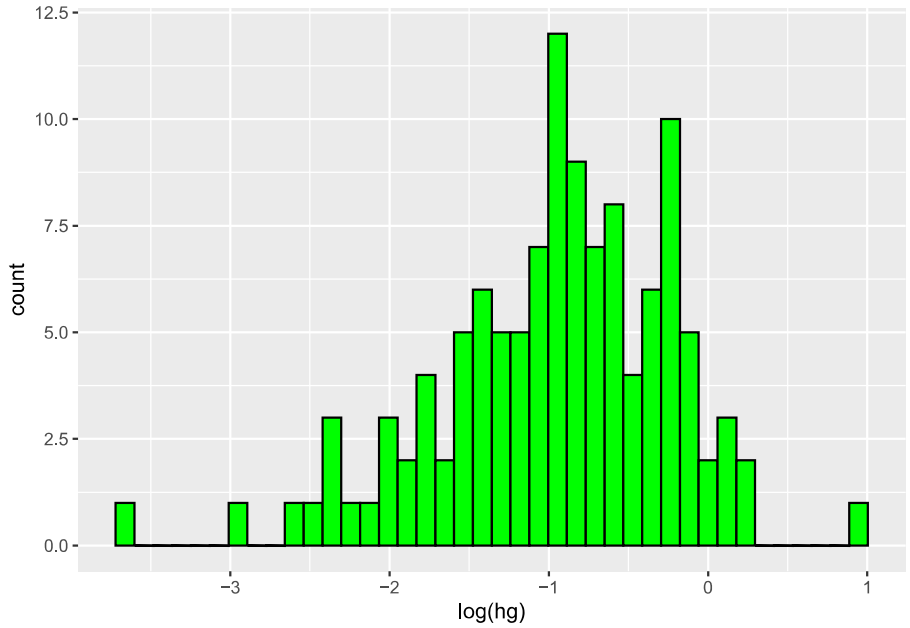
```
pl2 <- pl1 + geom_point()
pl2
```



2.3.2 To create a histogram: `geom_histogram`

To create a histogram for continuous variable `log(hg)`

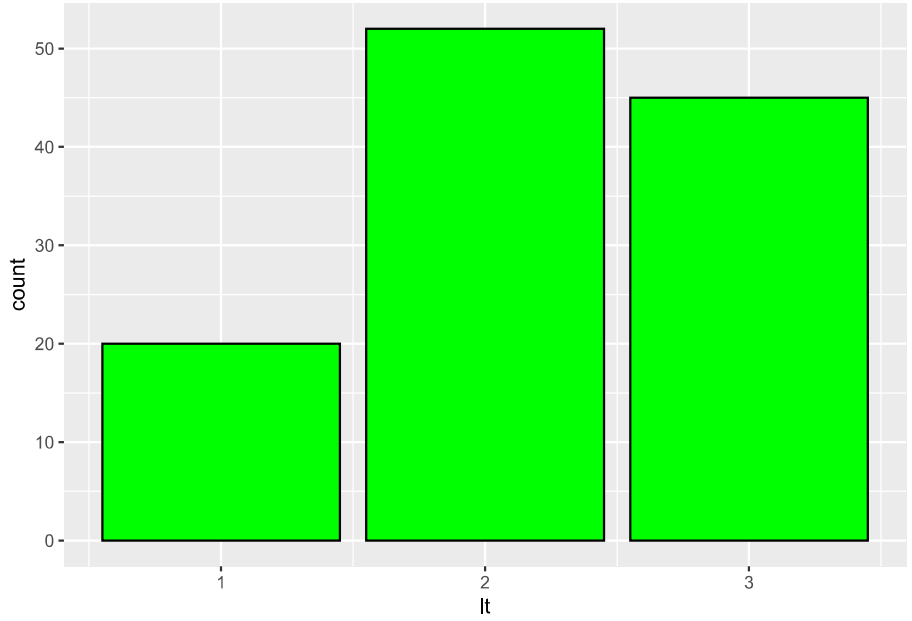
```
pl3 <- ggplot(fish, aes(log(hg))) + geom_histogram(bins = 40, colour = "black", fill = "green")
pl3
```

2.3.3 To create a bar chart for a categorical variable: `geom_bar`

To create a bar chart for categorical variable lake type (`lt`)

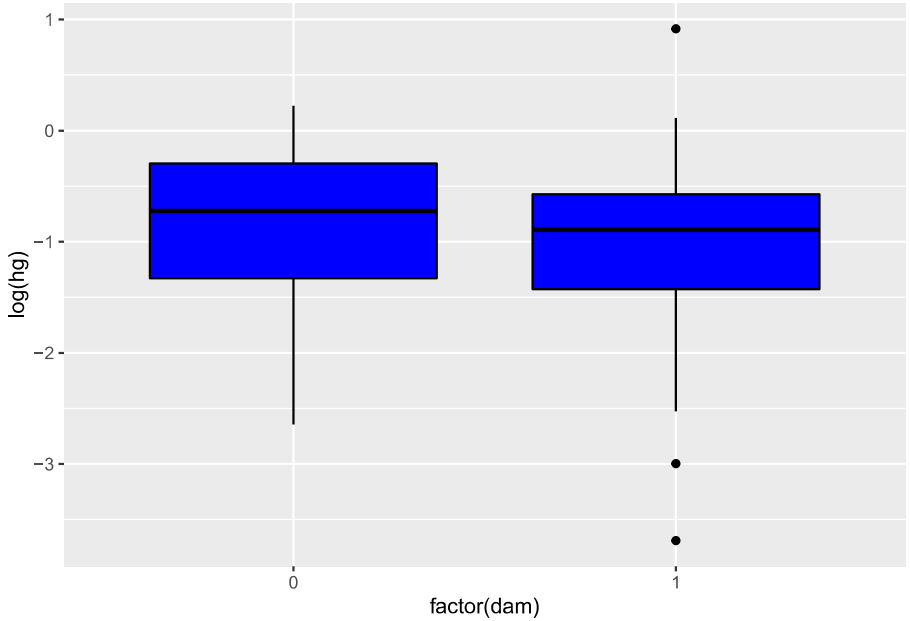
```
pl4 <- ggplot(fish, aes(lt)) + geom_bar(colour = "black", fill = "green")
pl4
```



2.3.4 To create a boxplot: `geom_boxplot`

To create a boxplot of `log(hg)` for lakes with and without `dam`

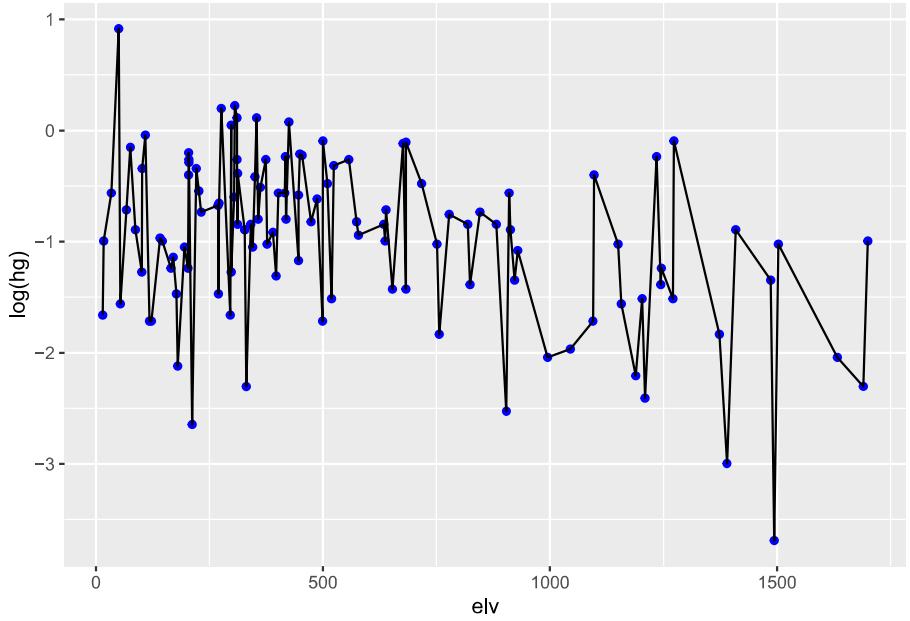
```
pl5 <- ggplot(fish, aes(x = factor(dam), y = log(hg))) +
  geom_boxplot(colour = "black", fill = "blue")
pl5
```

2.3.5 To produce a line: `geom_line`

Create a scatterplot and connect the dots by a line

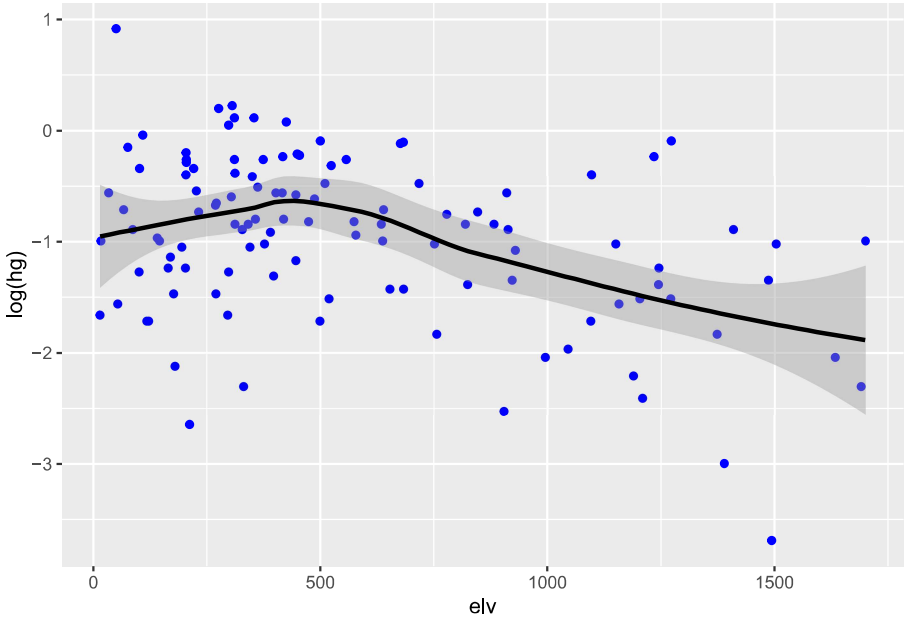
```
ggplot(fish, aes(x = elv, y = log(hg))) + geom_point(colour = "blue") +
  geom_line(colour = "black")
```



2.3.6 To produce a smooth trend line: `geom_smooth`

```
pl6 <- ggplot(fish, aes(x = elv, y = log(hg))) + geom_point(colour = "blue") +
  geom_smooth(colour = "black")
pl6
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

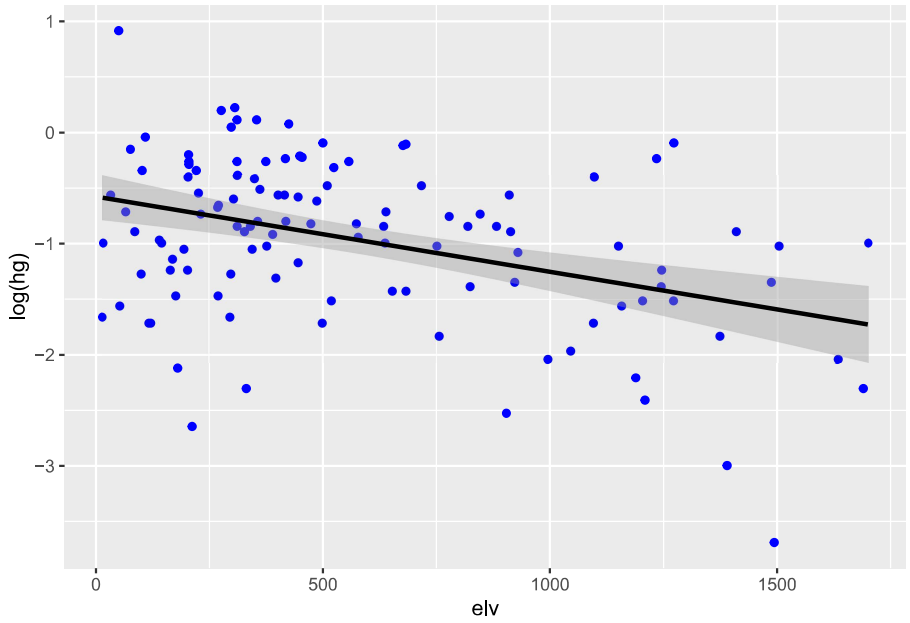



Remark:

- For small n (< 1000), a `loess` smoother is used by default.
- If you want to fit a linear model, you can change the method to `lm`.
- If you want a robust fitting line, you can use `method = rlm`. (You then first have to load the `MASS` package).

Add regression line

```
pl7 <- ggplot(fish, aes(x = elv, y = log(hg))) + geom_point(colour = "blue") +
  geom_smooth(colour = "black", method = "lm")
pl7
```



2.3.7 Remarks

Remark 1:

By default, `ggplot2` gives you the 95% confidence interval.

- In case you want to change the confidence level, use e.g., `level = 0.9`.
- In case you do not want to see the 95% confidence band, use `se = FALSE`.
- In case you want to have the 95% prediction interval: you can use the `geom_ribbon` function.

In case you want to see the prediction interval instead of confidence interval

```
# Fit a linear model
m.lm <- lm(log(hg) ~ elv, data = fish)
res.pred <- predict(m.lm, interval = "predict")
head(res.pred)
```

```
##          fit       lwr        upr
## 1 -0.8632111 -2.210718  0.4842960
## 2 -1.5869632 -2.959411 -0.2145157
## 3 -0.8476393 -2.195373  0.5000948
## 4 -0.9525800 -2.299410  0.3942505
## 5 -0.8577948 -2.205377  0.4897876
## 6 -0.7142631 -2.065263  0.6367366
```

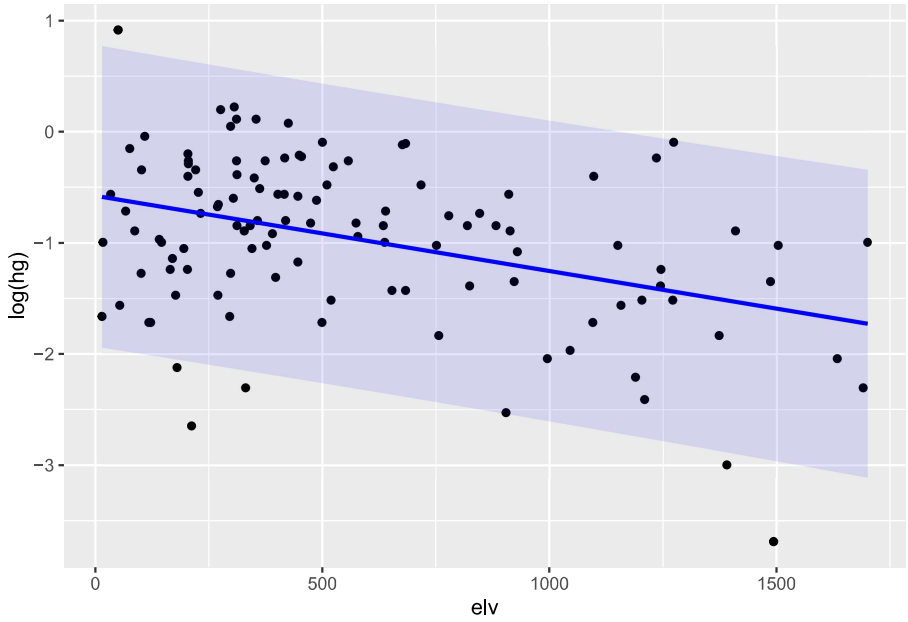
```
# cbind the predictions to fish
fish.pred <- cbind(fish, res.pred)
names(fish.pred)
```

```
##  [1] "name"   "hg"     "number" "elv"    "sa"     "z"      "lt"     "st"
```



```
##  [9] "dam"    "lat1"   "lat2"   "lat3"   "long1"  "long2"  "long3"  "fit"
## [17] "lwr"    "upr"
```

```
# Make now the plot
# remark that not all aesthetics are defined beforehand
pl7a <- ggplot(fish.pred, aes(x = elv)) +
  geom_point(aes(y = log(hg))) +
  geom_line(aes(y = fit), colour = "blue", size = 1)
pl7b <- pl7a + geom_ribbon(aes(ymin = lwr, ymax = upr), fill = "blue", alpha = 0.1)
pl7b
```



Remark 2:

Example *fish*

We use previous example as how you can **use multiple data frames** in one and the same `ggplot`. We here use the data frame `fish` and the data frame `pred_fish`.

1. Fit a linear model and create the data frame `pred_fish`

```
m.lm <- lm(log(hg) ~ elv, data = fish)
pred_fish <- cbind(elv = fish$elv, data.frame(predict(m.lm, interval = "prediction")))
names(pred_fish)
```

```
## [1] "elv" "fit" "lwr" "upr"
```

2. Make now the scatterplot of `elv` versus `log(hg)` **based on `fish` data frame**

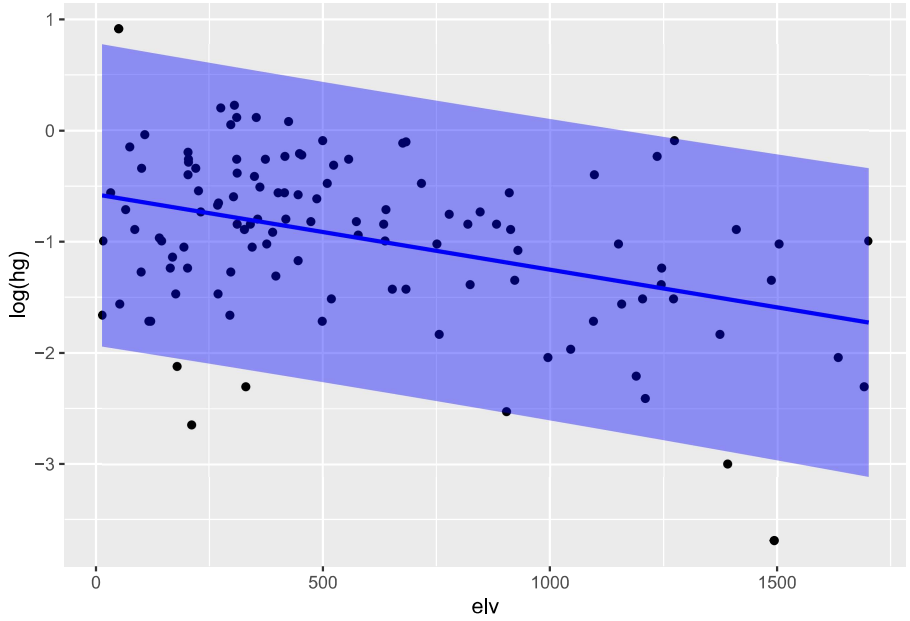
```
pl7d <- ggplot() + geom_point(data = fish, aes(x = elv, y = log(hg)))
```

3. Add now the fitted values **based on `pred_fish` data frame**

```
pl7e <- pl7d + geom_line(data = pred_fish, aes(x = elv, y = fit), colour = "blue",
                         size = 1)
```

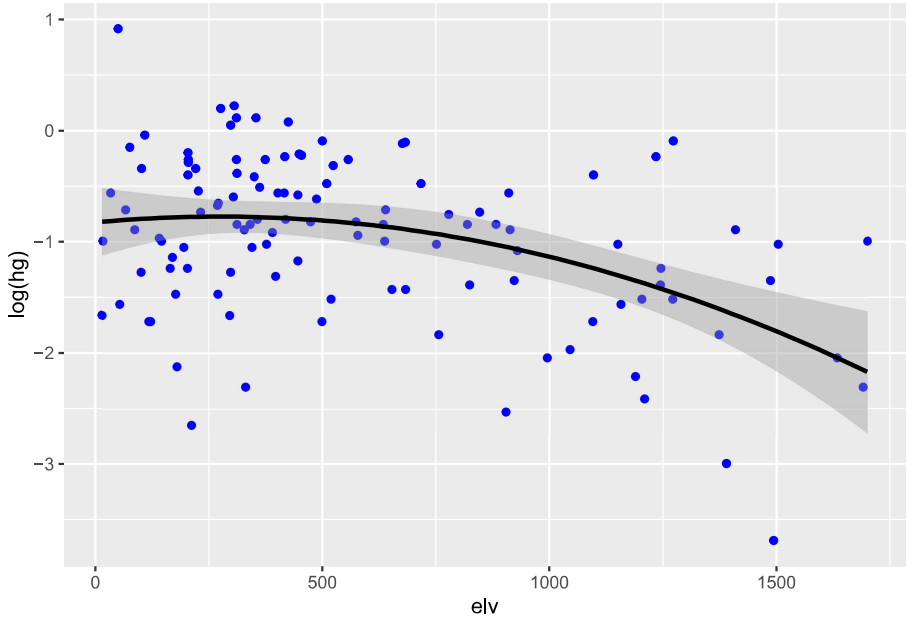
4. Add now the prediction limits **based on `pred_fish` data frame.
   Alpha expresses the density of the ribbon.

```
pl7f <- pl7e + geom_ribbon(data = pred_fish, aes(x = elv, ymin = lwr, ymax = upr),
                           fill = "blue", alpha = 0.4)
pl7f
```



Remark 3:
You can also specify the underlying model by for example `formula = y ~ x`

```
ggplot(fish, aes(x = elv, y = log(hg))) + geom_point(colour = "blue") +
  geom_smooth(colour = "black", formula = y ~ x + I(x^2), method = "lm")
```

2.4 Layer Stat

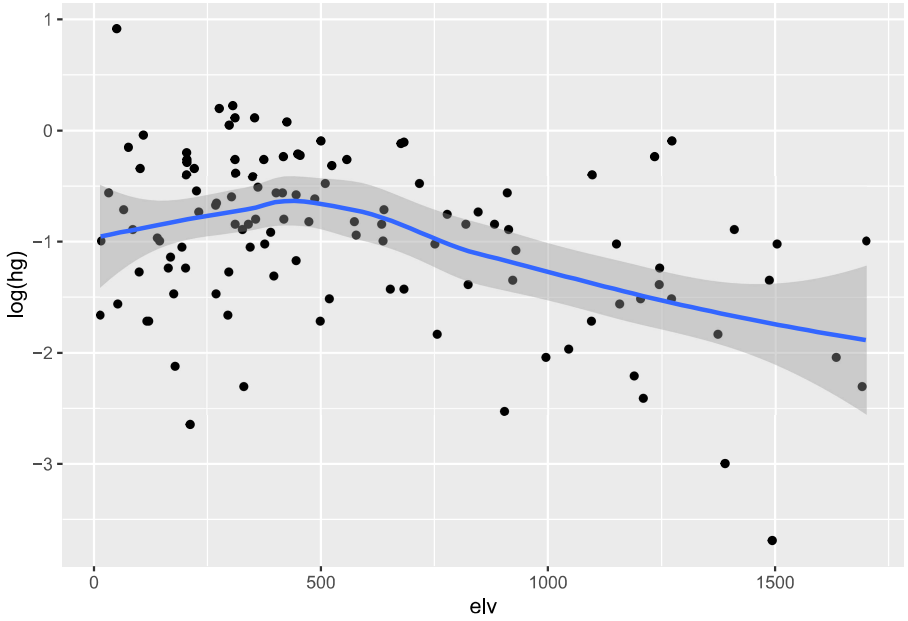
The *Stat* layer describes how the data should be summarized.

Some useful stats and default geoms:

stat	description	default geom
`stat_bin()`	Counts number of observations per bin	`geom_bar()`
`stat_smooth()`	Creates a smooth line	`geom_smooth()`
`stat_sum()`	Adds values	`geom_point()`
`stat_identity()`	No summary, plots data as it	`geom_point()`
`stat_boxplot()`	Summarizes data for boxplot	`geom_boxplot()`

2.4.1 Create a smooth line: `stat_smooth`

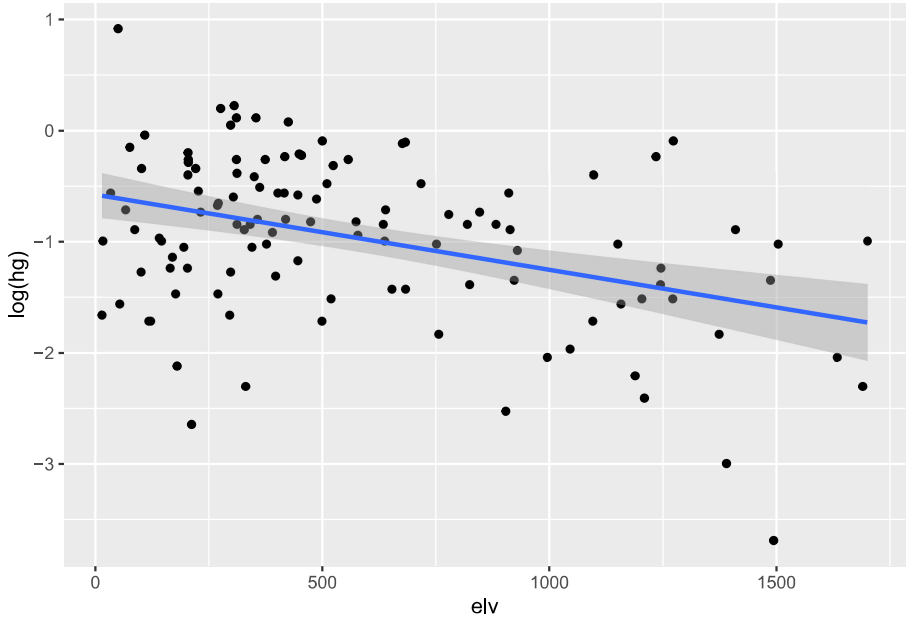
```
ggplot(fish, aes(elv, log(hg))) + geom_point() + stat_smooth()
```



- You first create a scatterplot by `geom_point()`
- You then add a smooth line with `stat_smooth()`: it uses `loess()` regression

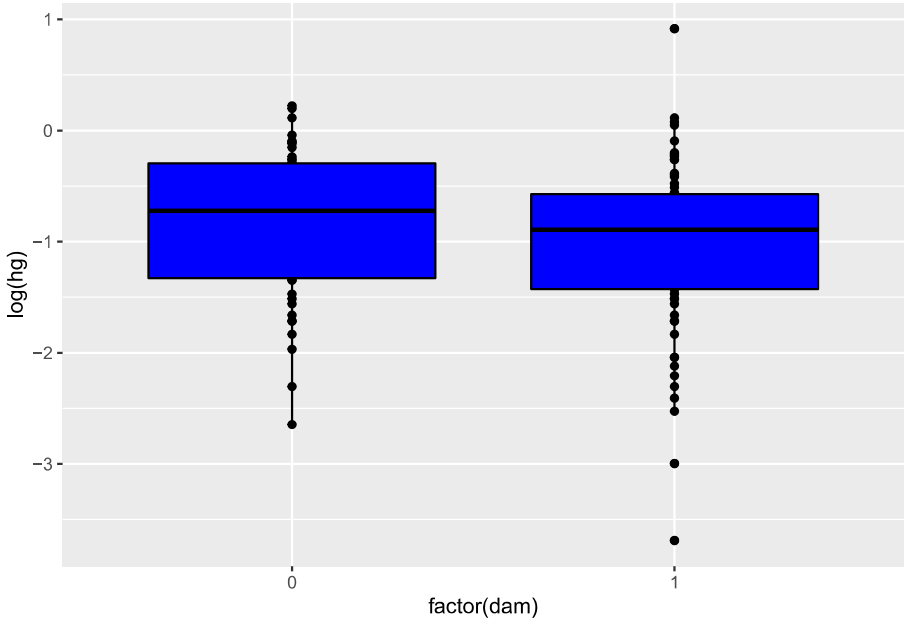
2.4.2 Create a regression line: `stat_smooth(method = 'lm')`

```
ggplot(fish, aes(elv, log(hg))) + geom_point() + stat_smooth(method = 'lm')
```

2.4.3 Add boxplot

```
ggplot(fish, aes(x = factor(dam), y = log(hg))) + geom_point(colour = "black") +
  stat_boxplot(colour = "black", fill = "blue")
```



2.5 Layer Facet

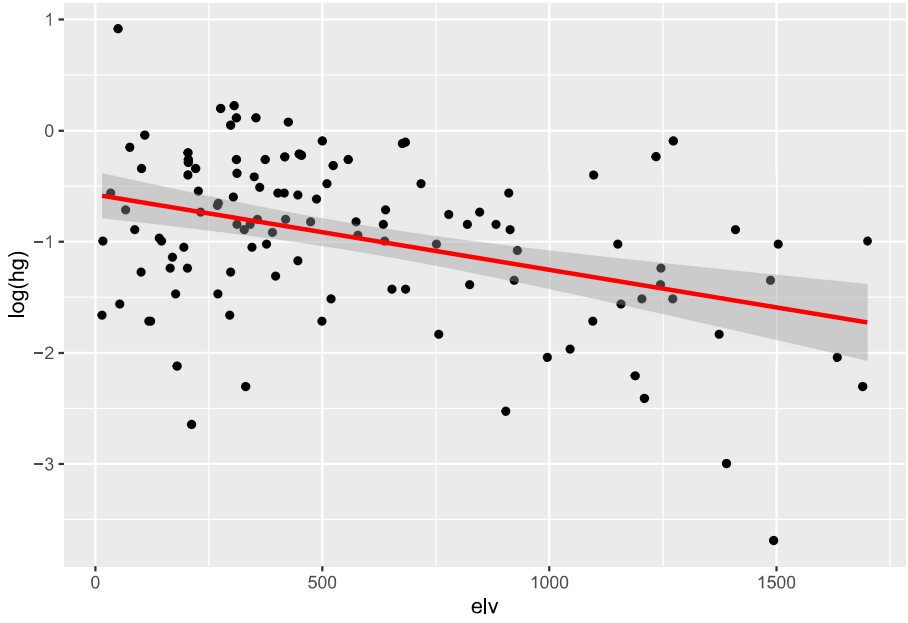
There are two types of faceting provided by ggplot2: **facet_grid** and **facet_wrap**

2.5.1 facet_grid

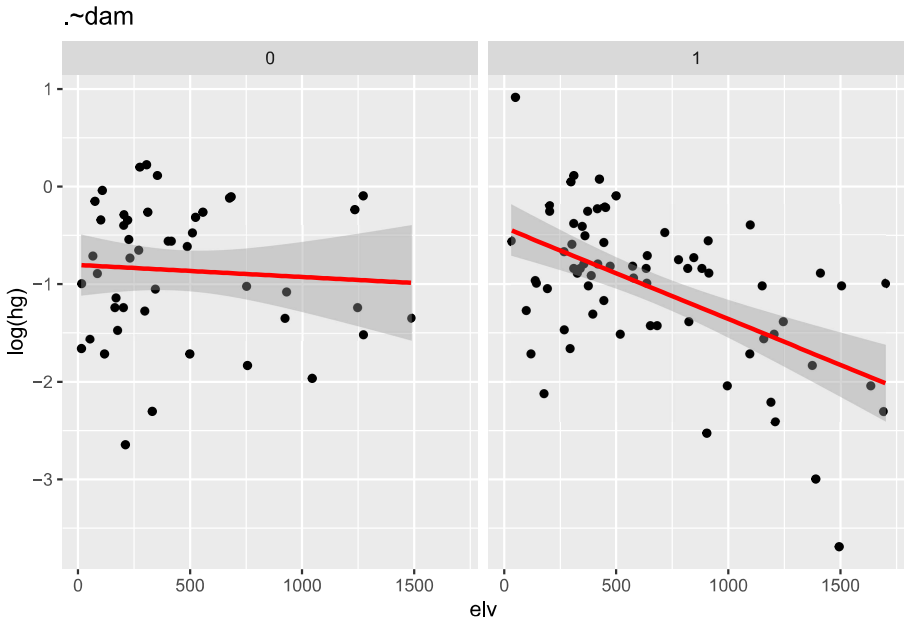
The specification of faceting variables is of the form (row ~ column)
. ~ a
a ~ .
a ~ b

Using the data set fish:

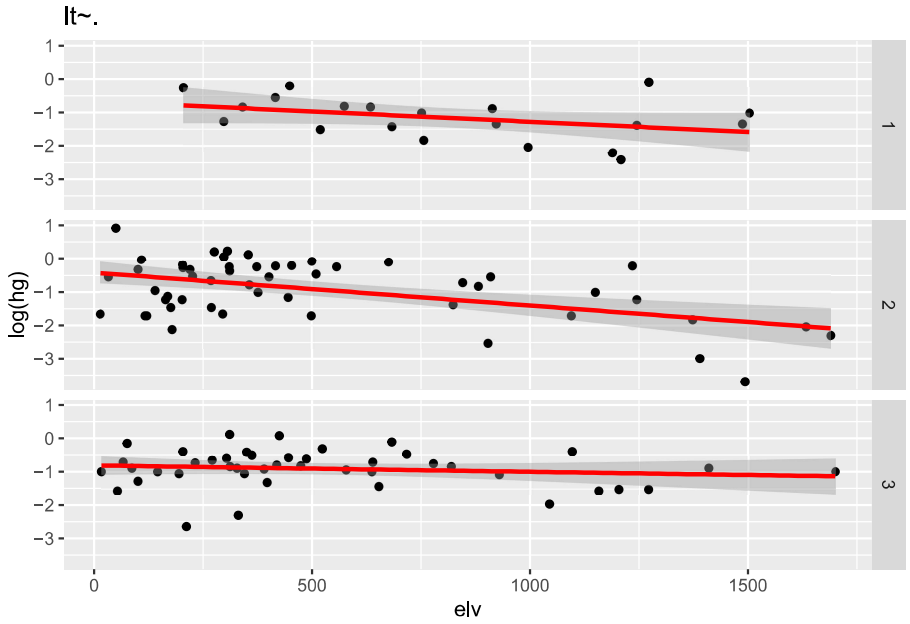
```
fac1 <- ggplot(fish, aes(elv, log(hg))) + geom_point() +
  geom_smooth(colour = "red", method = "lm")
fac1
```

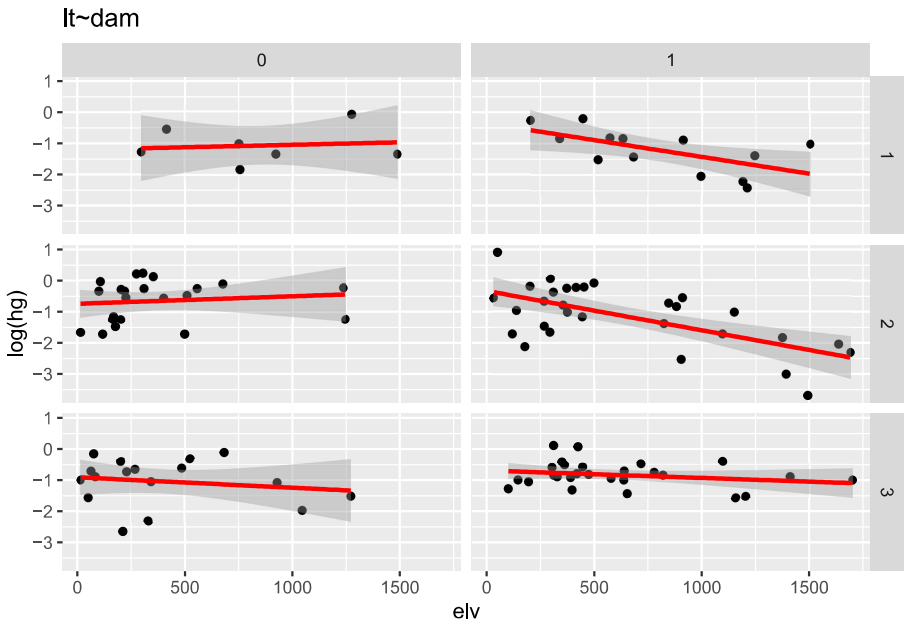

```
fac2 <- fac1 + facet_grid(. ~ dam) + labs(title = ".~dam")
fac2
```



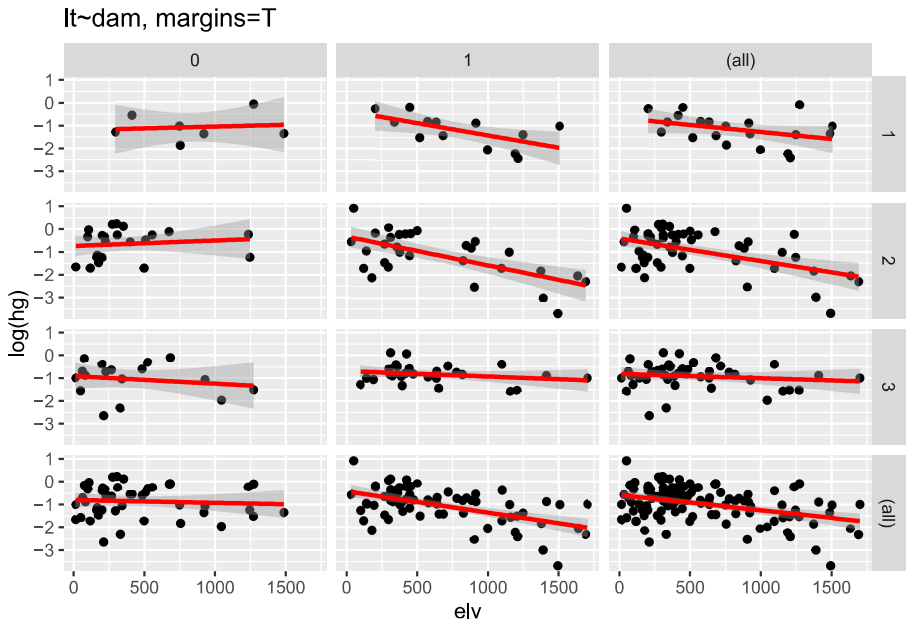

```
fac3 <- fac1 + facet_grid(lt ~ .) + labs(title = "lt~.")
fac3
```


```
fac4 <-  fac1 + facet_grid(lt ~ dam) +  labs(title = "lt~dam")
fac4
```



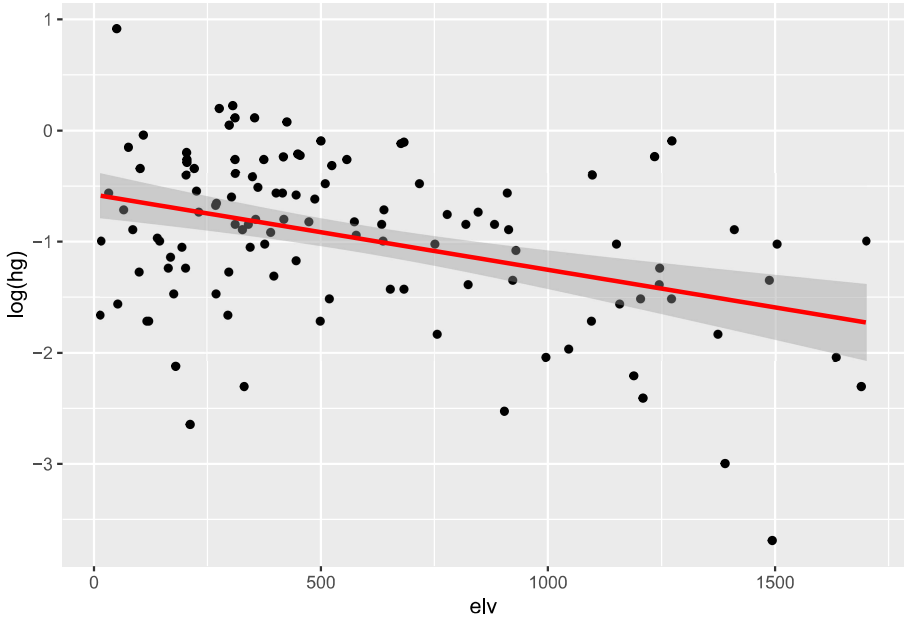

```
fac5 <-  fac1 + facet_grid(lt ~ dam, margins = T) + labs(title = "lt~dam, margins=T")
fac5
```

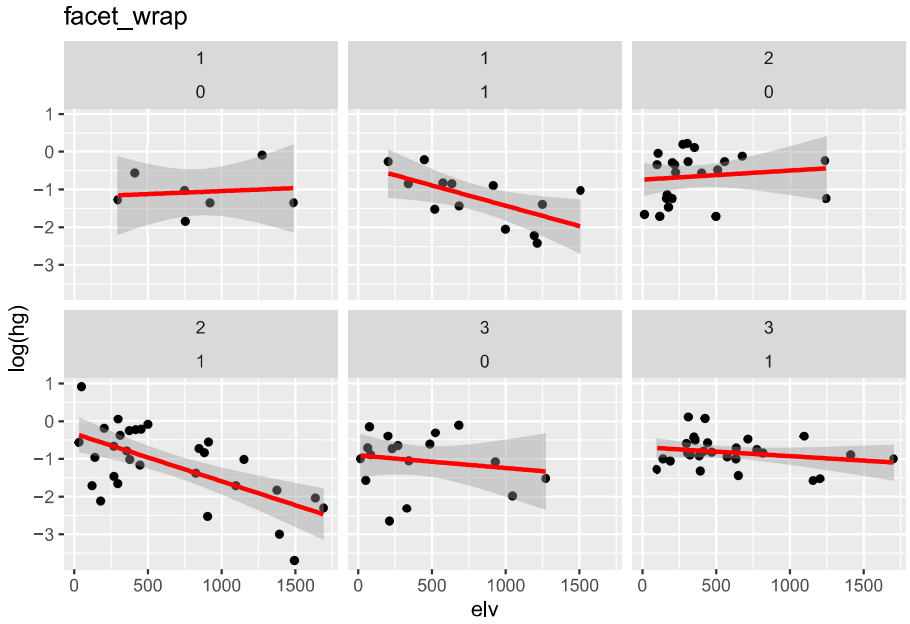
2.5.2 facet_wrap

The specification of faceting variables in facet_wrap is of the form ~ a + b

```
fac1 <- ggplot(fish, aes(elv, log(hg))) + geom_point() +
  geom_smooth(colour = "red", method = "lm")
fac1
```




```
fac2 <- fac1 + facet_wrap(~ lt + dam) + labs(title = "facet_wrap")
fac2
```

2.5.3 Difference between faceting and grouping

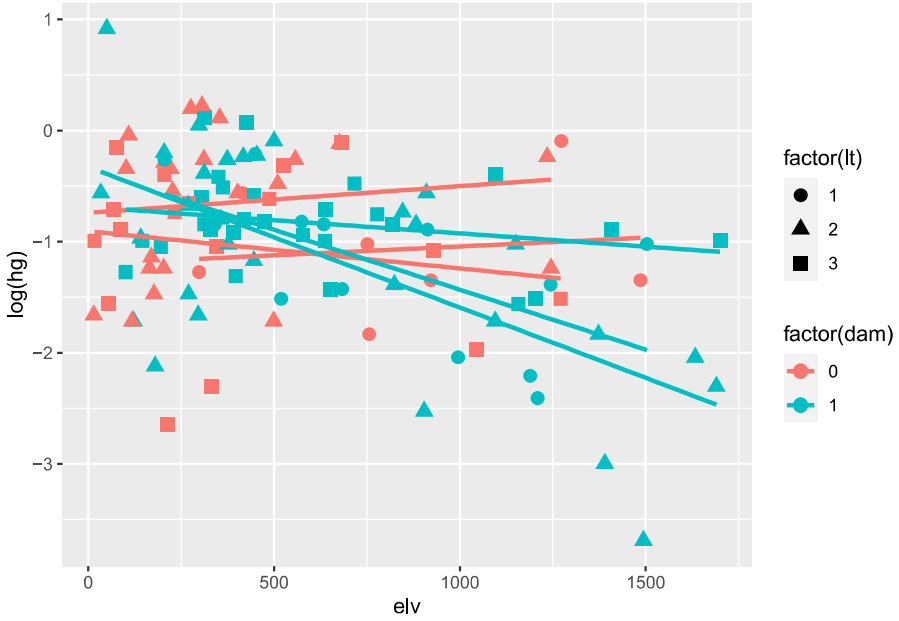
- With **faceting**: each group is quite far apart in its own panel, and there is no overlap between the groups. If there are small differences between groups, then these are harder to detect.
- When using **grouping**, the groups are close together and may overlap, but small differences are easier to detect.

With faceting, you can split in two dimensions and that is harder with grouping (using different colours and different symbols).

Example *fish*

Example of grouping:

```
ggplot(fish, aes(elv, log(hg), colour = factor(dam), shape = factor(lt))) +
  geom_point(size = 3) + geom_smooth(method = "lm", se = FALSE)
```



2.5.4 Faceting by continuous variables

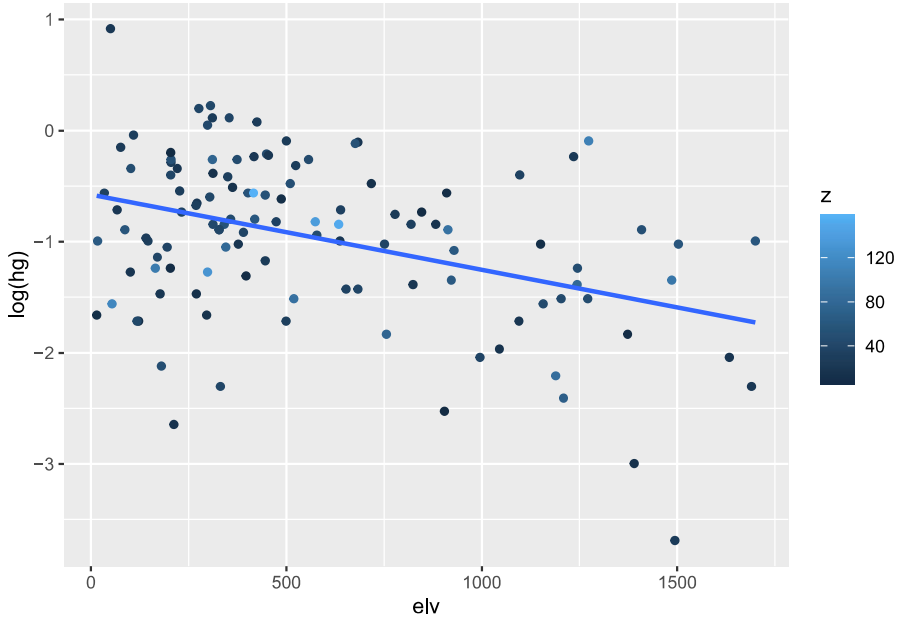
You first need to convert the continuous variables into discrete categories.

Assume that we want to see the scatterplot of `log(hg)` versus elevation **according to the value of z** (max. depth of the lake):

2.5.4.1 Method 1: Grouping by a continuous variable

Color points by value of continuous variable

```
ggplot(fish, aes(elv, log(hg), colour = z)) + geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

2.5.4.2 Method 2: Categorize your continuous variable

Step 1: Convert the continuous variable into a variable with discrete categories.

```
fish$z_cat1 <- cut_interval(fish$z, n=3)
# Or you can use
fish$z_cat2 <- cut_number(fish$z, n=3)

xtabs(~ fish$z_cat1)
```

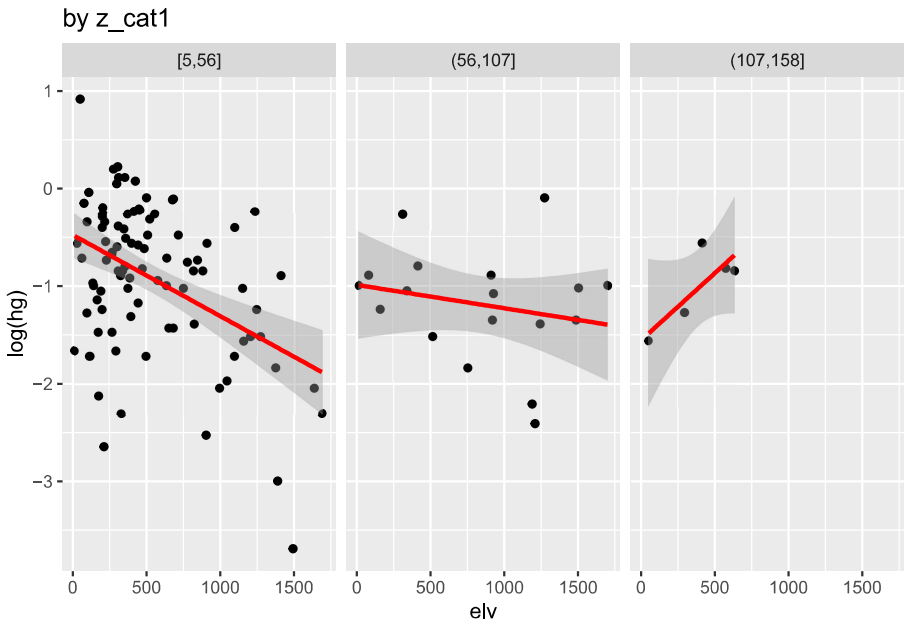
```
## fish$z_cat1
##    [5,56]  (56,107] (107,158]
##        94        18         5
```

```
xtabs(~ fish$z_cat2)
```

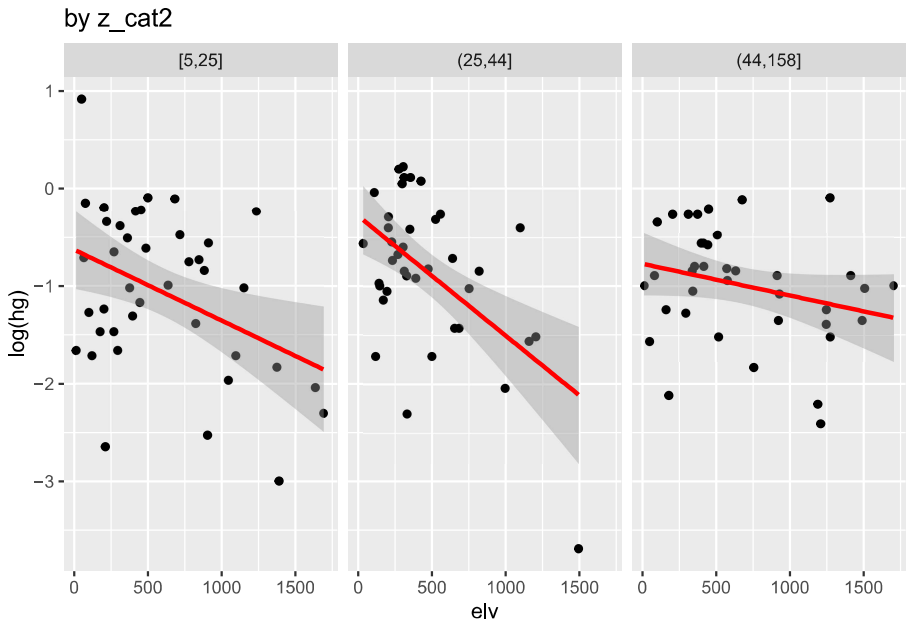
```
## fish$z_cat2
##   [5,25]  (25,44] (44,158]
##       40       39       38
```

Step 2: We now can use this variable for faceting

```
fac1 <- ggplot(fish, aes(elv, log(hg))) + geom_point() +
  geom_smooth(colour = "red", method = "lm")
fac2 <- fac1 + facet_wrap(~ z_cat1) + labs(title = "by z_cat1")
fac2
```




```
fac3 <- fac1 + facet_wrap(~ z_cat2) + labs(title = "by z_cat2")
fac3
```

3 Extras

3.1 Themes

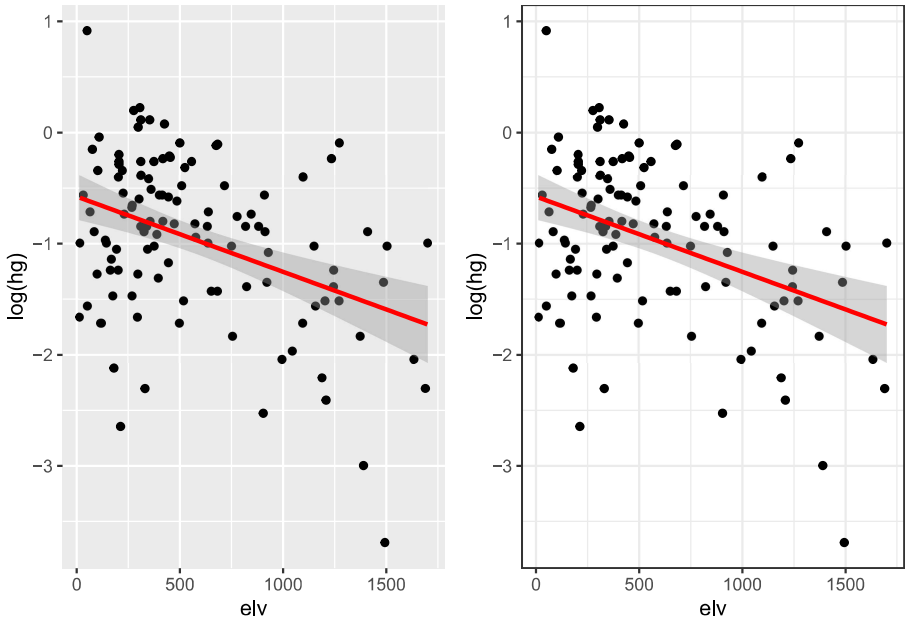
There are two built-in themes: `theme_gray()` and `theme_bw()`.

- The **default** `theme_gray()` uses a very light grey background with white grid lines.
- `theme_bw()` uses a white background with dark grey grid lines.

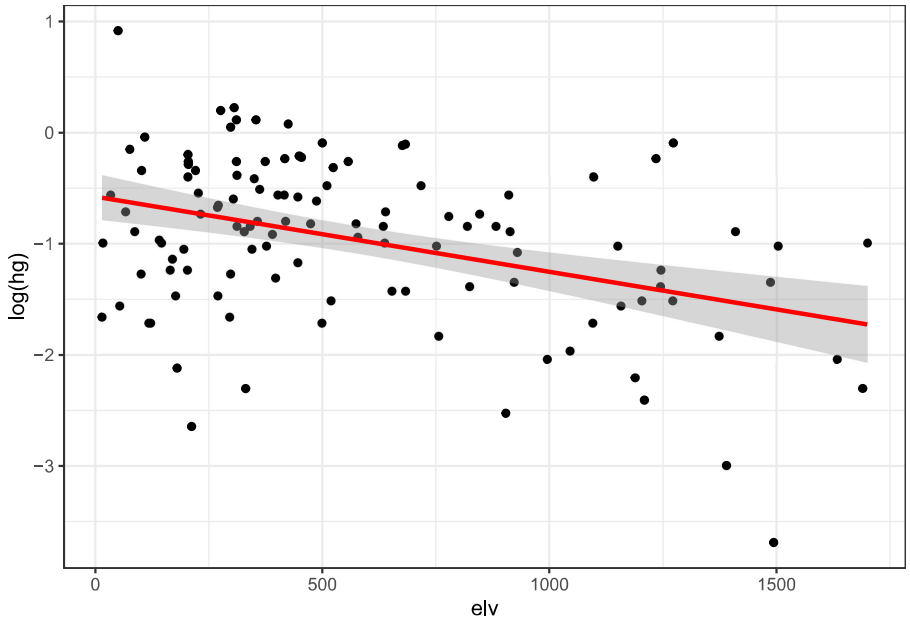
Example *fish*

Example: use of theme

```
fac1 <- ggplot(fish, aes(elv, log(hg))) + geom_point() +
  geom_smooth(colour = "red", method = "lm")
fac1
# You can override the theme for a single plot
fac2 <- fac1 + theme_bw()
fac2
```




```
# Affecting all plots
previous_theme <- theme_set(theme_bw())
fac1
```

3.2 Multiple plots on the same page

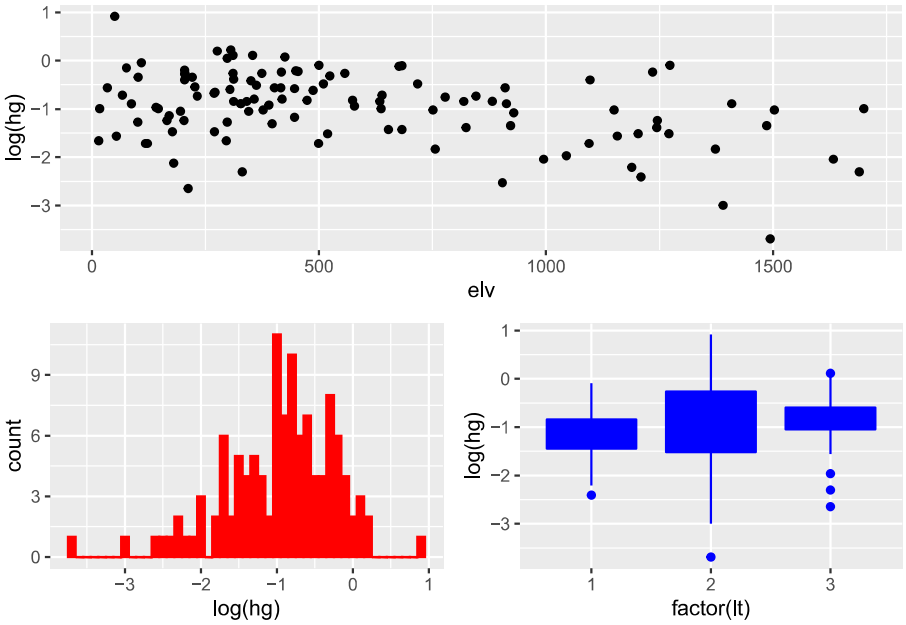
A viewport is a specified region in the entire plotting area. By customizing the viewport, you can arrange a set of plots.

First create the plots, assign them to objects and then plot the objects.

```
plot1 <- ggplot(fish, aes(elv,log(hg))) + geom_point()
plot2 <- ggplot(fish, aes(log(hg))) +
  geom_histogram(binwidth = 0.1, colour = "red", fill = "red" )
plot3 <- ggplot(fish, aes(factor(lt),log(hg))) +
  geom_boxplot(colour = "blue", fill = "blue" )
```

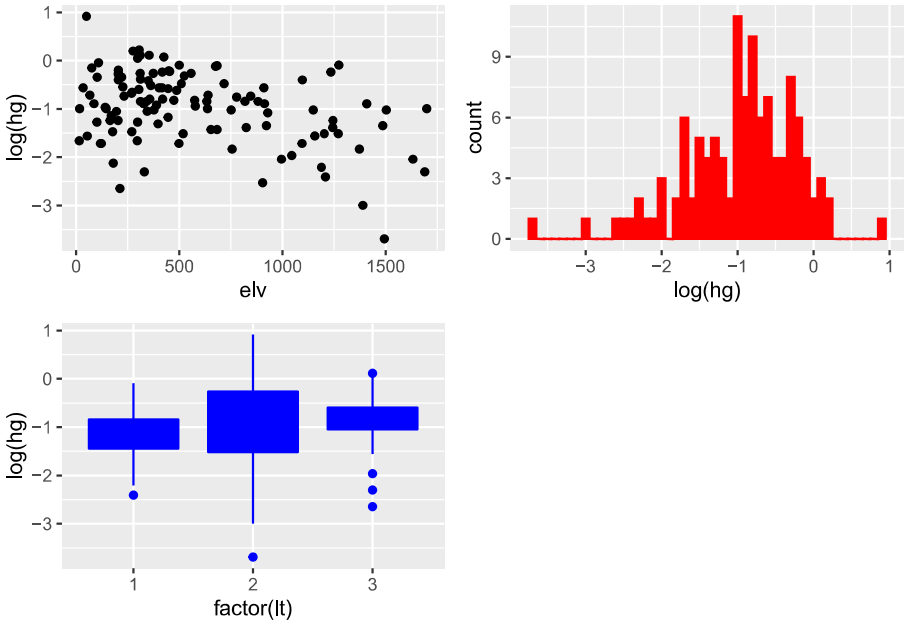
3.2.1 Use rectangular grids: use `grid.layout()`

```
grid.newpage()
pushViewport(viewport(layout=grid.layout(2,2)))
vplayout <- function(x,y){
  viewport(layout.pos.row=x, layout.pos.col=y)}
print(plot1, vp=vplayout(1,1:2))
print(plot2, vp=vplayout(2,1))
print(plot3, vp=vplayout(2,2))
```



Use rectangular grids: use ggarrange

```
ggarrange(plot1,plot2, plot3, ncol = 2, nrow = 2)
```

3.3 Save your output

Save your output to a *pdf* file

```
plot1 <- qplot(elv, log(hg), data = fish, geom = c("point"))
ggsave(file = "output1.pdf", plot = plot1)
```

Remark:
When you use *Latex*, it is recommended to save your work to a *.ps* file

Save your output to a *ps* file

```
ggsave(file = "output2.ps", plot = plot1)
```

```
## Saving 6.5 x 4.5 in image
```

4 Applications

4.1 Profile plots for visualizing longitudinal data

In this section, we are using the data set `Oxboys` from the package `nlme`.

```
install.packages("nlme")
library(nlme)
?Oxboys
```

Heights of Boys in Oxford

Description

The `Oxboys` data frame has 234 rows and 4 columns.

Format

This data frame contains the following columns:

Subject

an ordered factor giving a unique identifier for each boy in the experiment

age

a numeric vector giving the standardized age (dimensionless)

height

a numeric vector giving the height of the boy (cm)

Occasion

an ordered factor - the result of converting `age` from a continuous variable to a count so these slightly unbalanced data can be analyzed as balanced.

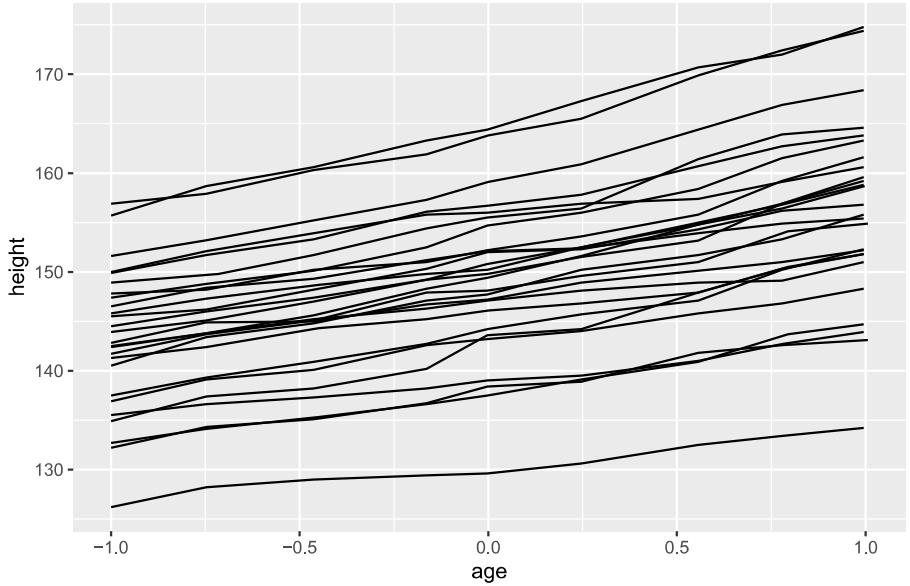
```
head(Oxboys, n = 12)
```

```
## Grouped Data: height ~ age | Subject
##    Subject     age height Occasion
## 1        1 -1.0000  140.5        1
## 2        1 -0.7479  143.4        2
## 3        1 -0.4630  144.8        3
## 4        1 -0.1643  147.1        4
## 5        1 -0.0027  147.7        5
## 6        1  0.2466  150.2        6
## 7        1  0.5562  151.7        7
## 8        1  0.7781  153.3        8
## 9        1  0.9945  155.8        9
## 10       2 -1.0000  136.9        1
## 11       2 -0.7479  139.1        2
## 12       2 -0.4630  140.1        3
```

- Individual profile plots: specify **subject as grouping variable**.

Visualize longitudinal data

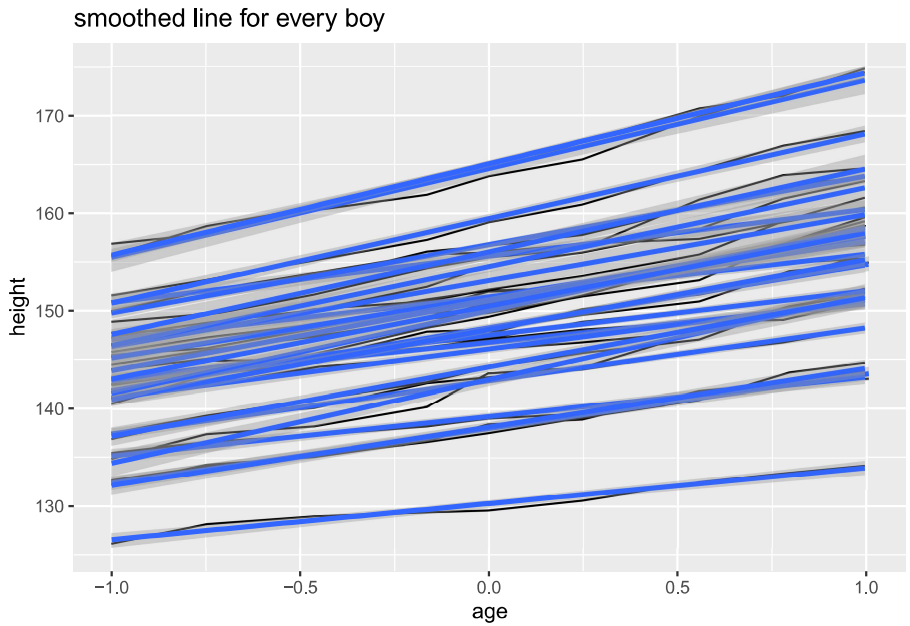
```
pl1 <- ggplot(Oxboys, aes(age, height, group = Subject))
pl2 <- pl1 + geom_line() + labs(title = "Individual profile plot")
pl2
```

- Individual profile plot with common trend

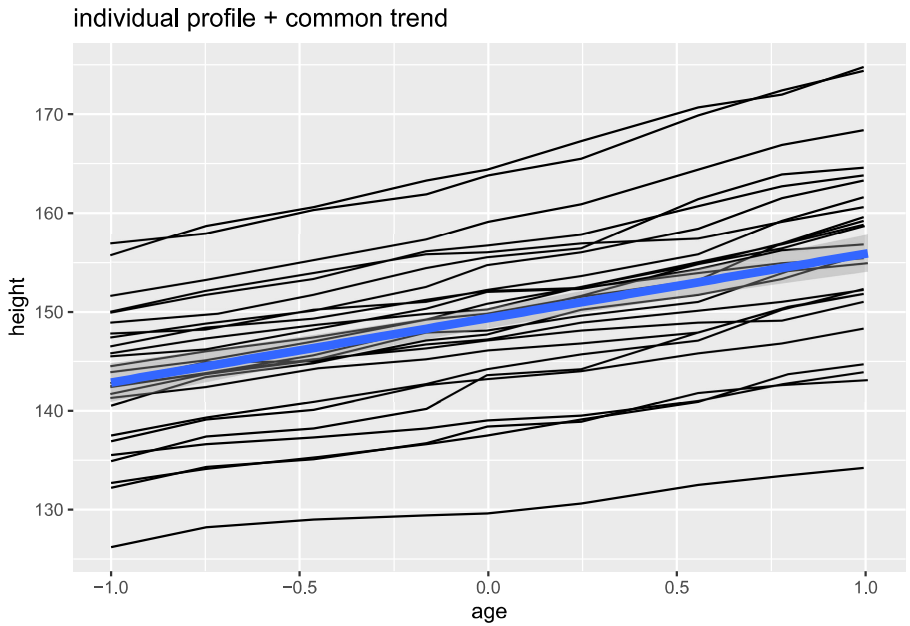
Adding a smoothed line for every boy

```
pl3 <- pl2 + geom_smooth(method = "lm", size = 1.2) +
  labs(title = "smoothed line for every boy")
pl3
```



Adding a smooth line based on ages and heights of all the boys.
Specifying `group = 1` indicates that you want a single line (and consider all data points as 1 group)

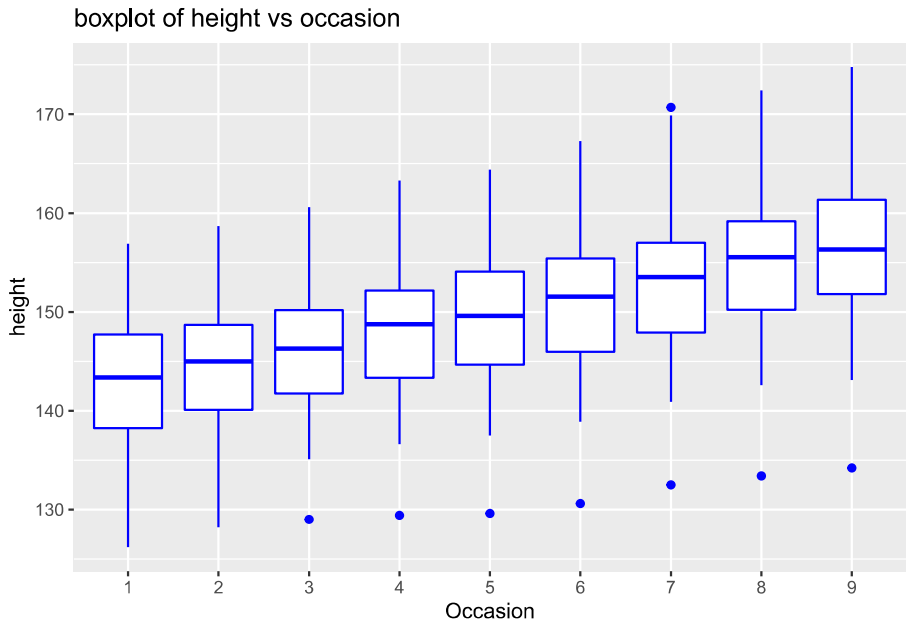
```
pl4 <- pl2 + geom_smooth(aes(group = 1), method = "lm", size = 2) +
  labs(title = "individual profile + common trend")
pl4
```

Connecting data points across groups

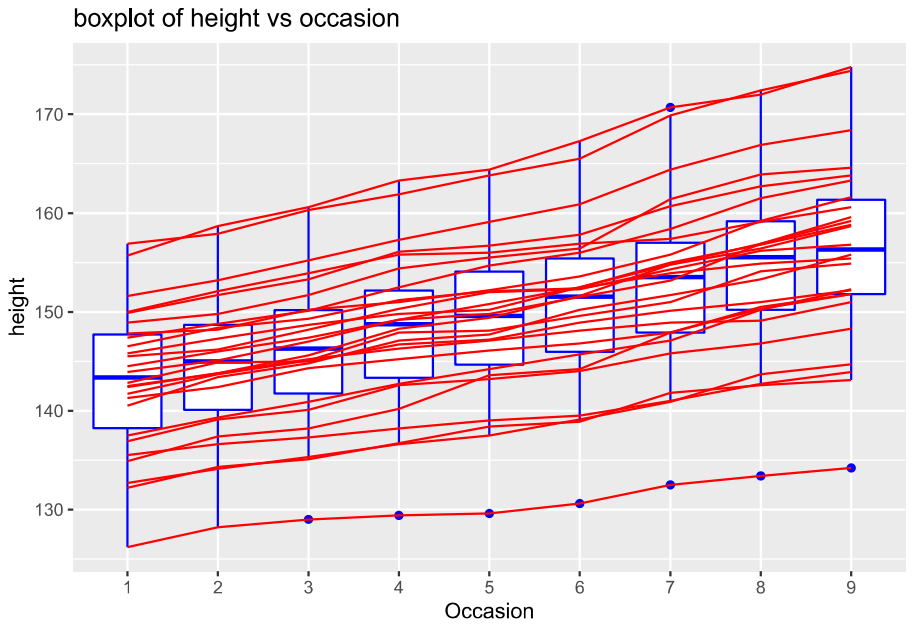
Construct a boxplot of `height` versus `Occasion`.

```
pl1 <- ggplot(Oxboys, aes(Occasion, height))
pl2 <- pl1 + geom_boxplot(colour = "blue") + labs(title = "boxplot of height vs occasion")
pl2
```



If you want to connect the values per boy over the several occasions

```
pl3 <- pl2 + geom_line(aes(group = Subject), colour = "red")
pl3
```

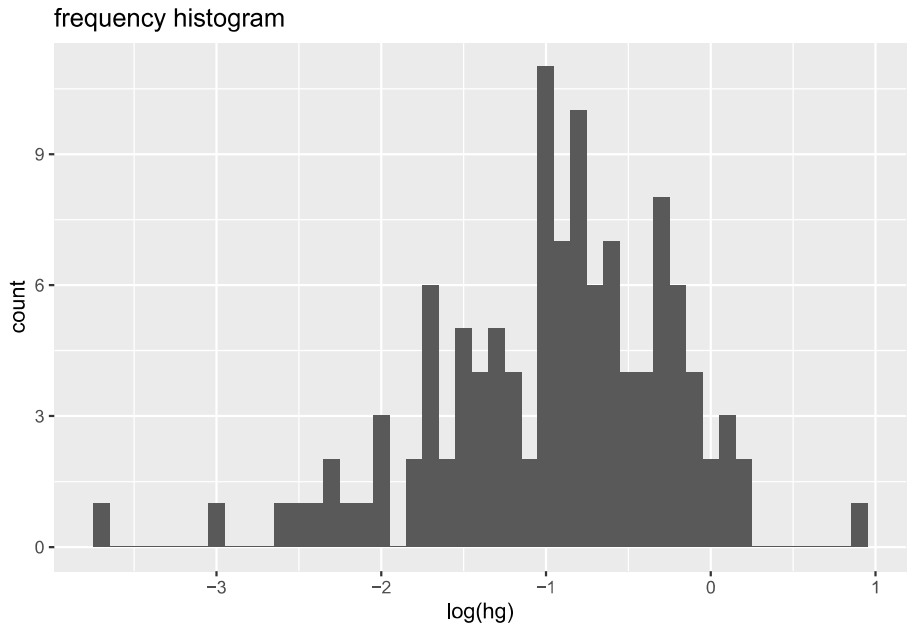



4.2 Create frequency histogram with density curve

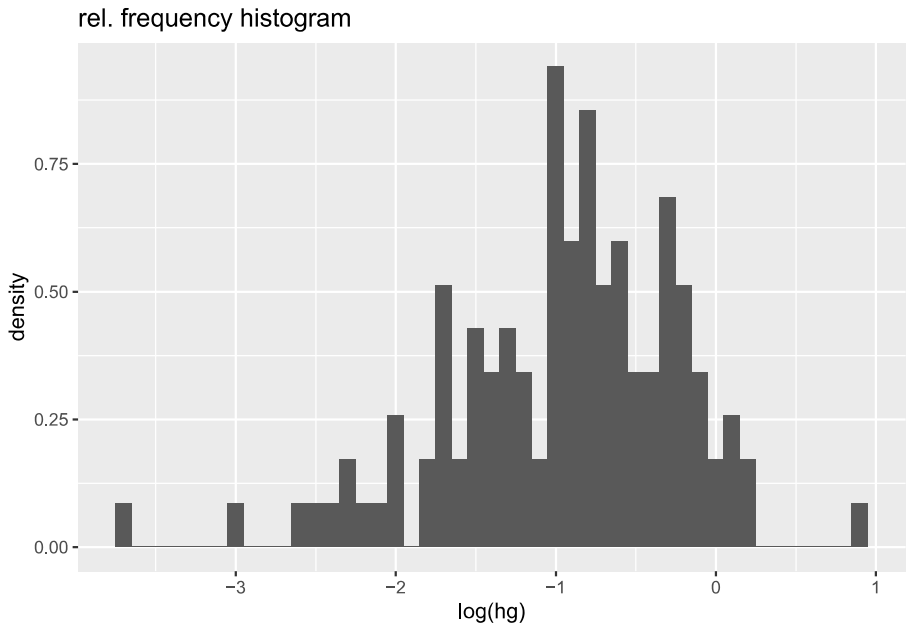
In this section, we use the data set `fish`.

Construct a frequency and relative frequency histogram overlayed with a density curve

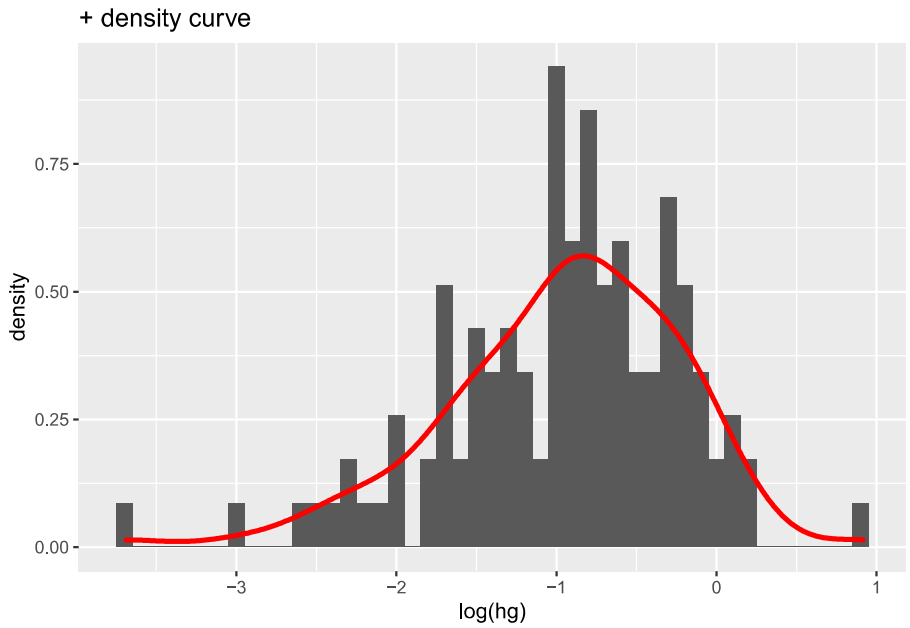
```
pl1 <- ggplot(fish, aes(log(hg)))
pl2 <- pl1 + geom_histogram(binwidth = 0.1) + labs(title = "frequency histogram")
pl2
```




```
pl3 <- pl1 + geom_histogram(aes(y = ..density..), binwidth = 0.1) +
  labs(title = "rel. frequency histogram")
pl3
```


```
pl4 <- pl3 + geom_density(colour = "red", size = 1.2) + labs(title = "+ density curve")
pl4
```



4.3 Visualize multiple variables on same plot (e.g. time series)

In this section, we use the data set `economics` from the package `ggplot2`.

US economic time series

Description

This dataset was produced from US economic time series data available from http://research.stlouisfed.org/fred2. `economics` is in "wide" format, `economics_long` is in "long" format.

Usage

```
economics

economics_long
```

Format

A data frame with 574 rows and 6 variables:

date

Month of data collection

pce

personal consumption expenditures, in billions of dollars, http://research.stlouisfed.org/fred2/series/PCE

pop

total population, in thousands, http://research.stlouisfed.org/fred2/series/POP

psavert

personal savings rate, http://research.stlouisfed.org/fred2/series/PSAVERT/

uempmed

median duration of unemployment, in weeks, http://research.stlouisfed.org/fred2/series/UEMPMED

unemploy

number of unemployed in thousands, http://research.stlouisfed.org/fred2/series/UNEMPLOY

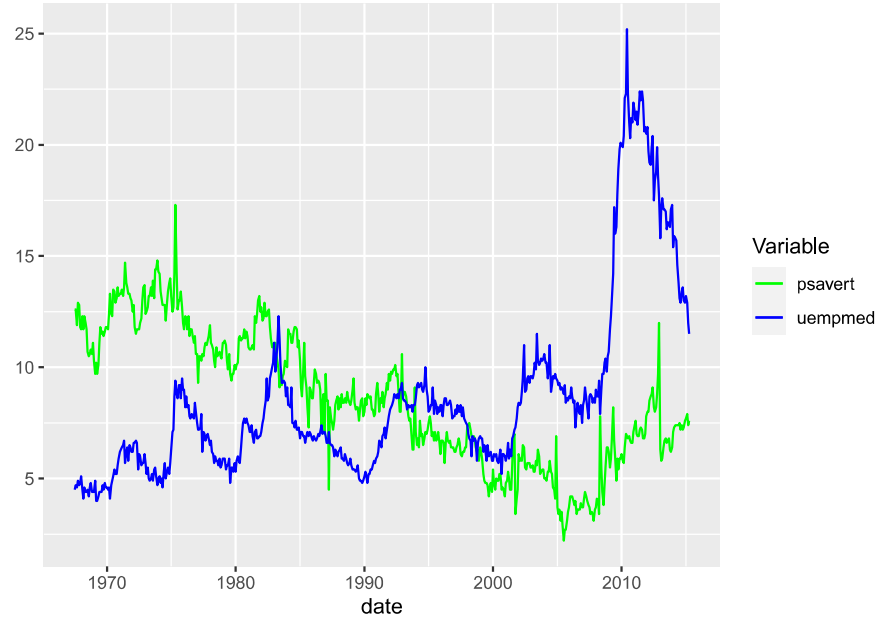
```
head(economics, 10)
```

```
## # A tibble: 10 x 6
##    date         pce    pop psavert uempmed unemploy
##    <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
##  1 1967-07-01  507. 198712    12.6     4.5     2944
##  2 1967-08-01  510. 198911    12.6     4.7     2945
##  3 1967-09-01  516. 199113    11.9     4.6     2958
##  4 1967-10-01  512. 199311    12.9     4.9     3143
##  5 1967-11-01  517. 199498    12.8     4.7     3066
##  6 1967-12-01  525. 199657    11.8     4.8     3018
##  7 1968-01-01  531. 199808    11.7     5.1     2878
##  8 1968-02-01  534. 199920    12.3     4.5     3001
##  9 1968-03-01  544. 200056    11.7     4.1     2877
## 10 1968-04-01  544  200208    12.3     4.6     2709
```

We want to visualize the personal savings rate (`psavert`) and median duration of unemployment (`uempmed`)

versus `date` (from 1970 till recent).

```
# Overlaying  lines
ts1 <- ggplot(economics, aes(date))
ts2 <- ts1 + geom_line(aes(y=psavert, colour="psavert")) +
  geom_line(aes(y=uempmed, colour="uempmed"))

# To omit the labeling of the Y axis
ts3 <-  ts2 + ylab(" ")

# To adapt the coloring of the lines + add nice heading to the legend
ts4 <- ts3 + scale_color_manual(name="Variable", values=c("green","blue"))
ts4
```



5 Adding statistical summaries

Using data set `fish`, visualize the `log(hg)` value and its average value for every lake type (`lt`).

Summarize y values at every value of x: **`stat_summary()`**

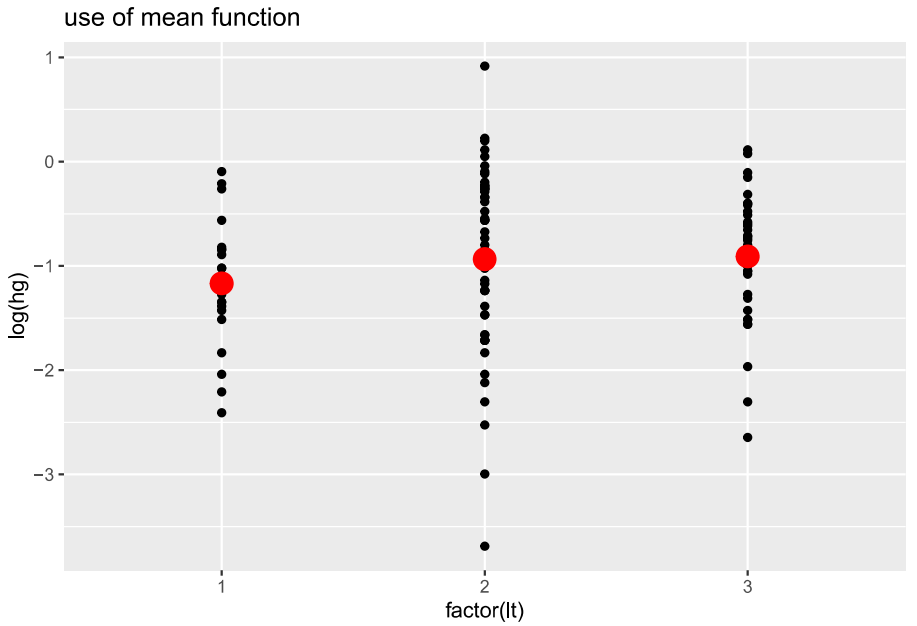
5.1 Individual summary functions

You can use the functions `fun` to create simple numeric summary functions. You can use e.g. `mean()`, `median()`,...

```
install.packages("Hmisc")
library(Hmisc)
```



```
pl1 <- ggplot(fish, aes(factor(lt), log(hg)))
pl2 <- pl1 + geom_point()
pl3 <- pl2 + stat_summary(fun = "mean", geom = "point", size = 5, colour = "red")
pl4 <- pl3 + labs(title = "use of mean function")
pl4
```

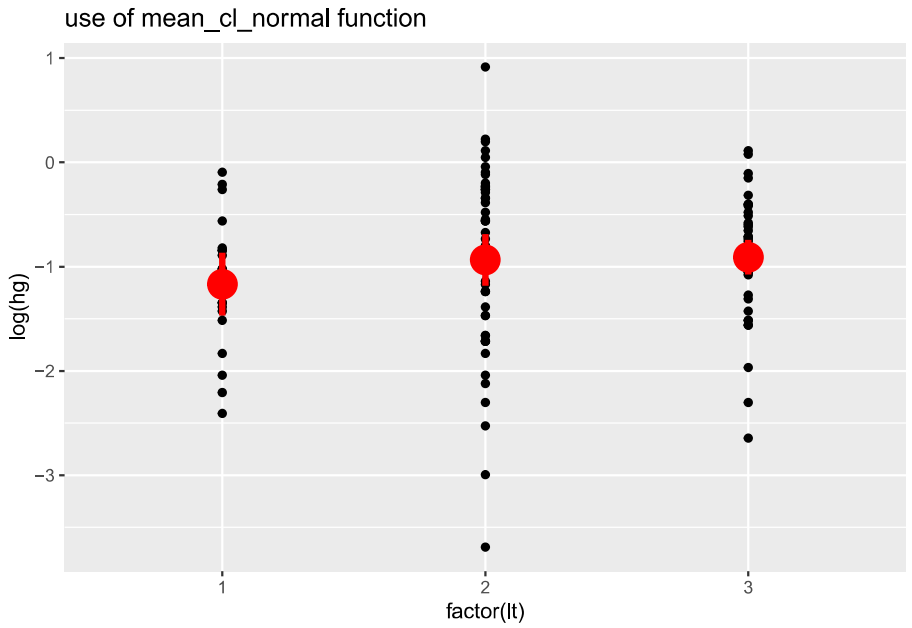


5.2 Single summary functions

`fun.data` can be used with more complex summary functions (you can write also your own summary function!).

`fun.data = ...`	middle	range
`mean_cl_normal()`	mean	se from normal approx.
`mean_cl_boot()`	mean	se from bootstrap
`median_hilow()`	median	25^{th} and 75^{th} percentile

```
pl1 <- ggplot(fish, aes(factor(lt), log(hg)))
pl2 <- pl1 + geom_point()
pl5 <- pl2 + stat_summary(fun.data = "mean_cl_normal", colour = "red", size = 1.5)
pl6 <- pl5 + labs(title = "use of mean_cl_normal function")
pl6
```

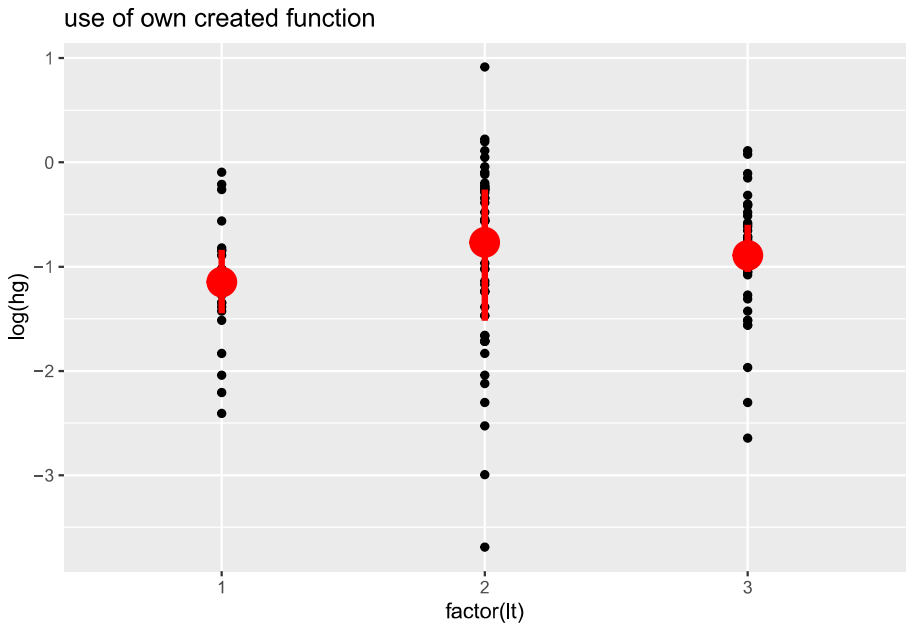


Write your own function:
Assume we want to show median with Q1 and Q3 as lower and upper bound

```
quant <- function(x)
{ q1 <- quantile(x, 0.25)
  q2 <- quantile(x, 0.50)
  q3 <- quantile(x, 0.75)
  qs <- c(q1, q2, q3)
  names(qs) <- c("ymin", "y", "ymax")
  qs}
tapply(log(fish$hg), fish$lt,quant)
```

```
## $`1`
##       ymin          y       ymax
## -1.4488692 -1.1473085 -0.8382227
##
## $`2`
##       ymin          y       ymax
## -1.5174398 -0.7662384 -0.2613648
##
## $`3`
##       ymin          y       ymax
## -1.0498221 -0.8915981 -0.5978370
```

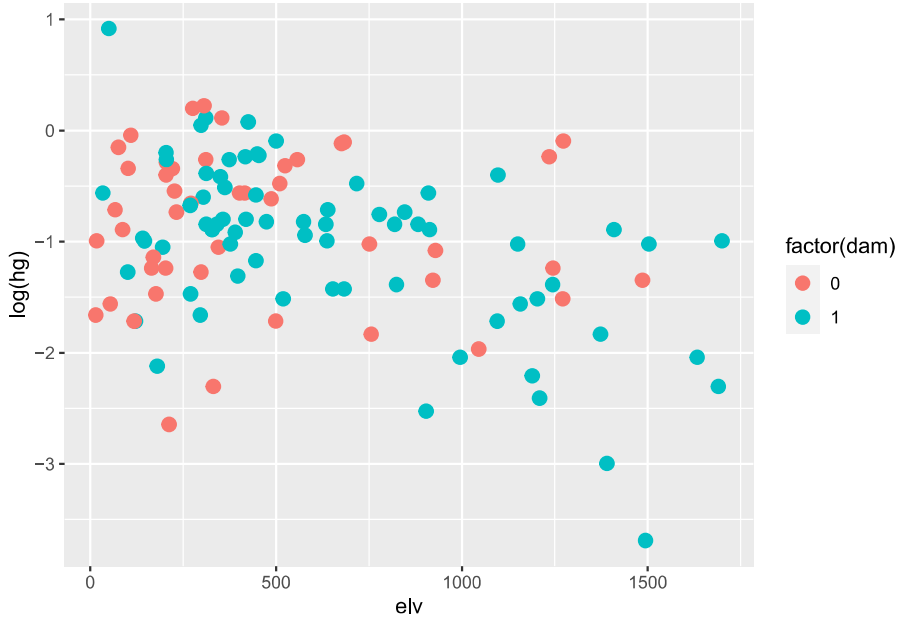
```
pl7 <- pl2 + stat_summary(fun.data = "quant", colour = "red", size = 1.5)
pl7 + labs(title = "use of own created function")
```

6 Animated graph

```
library(gganimate)
library(gifski)
```

```
g1 <- ggplot(fish, aes(elv, log(hg))) + geom_point(aes(colour = factor(dam)), size = 3)
g1
```




```
animo1 <- g1 + transition_states(factor(dam))
animo2 <- animo1 + enter_fade() + exit_shrink()
# The command 'animo2' will now give an animated graph.
```

7 Exercises

7.1 `tips` data

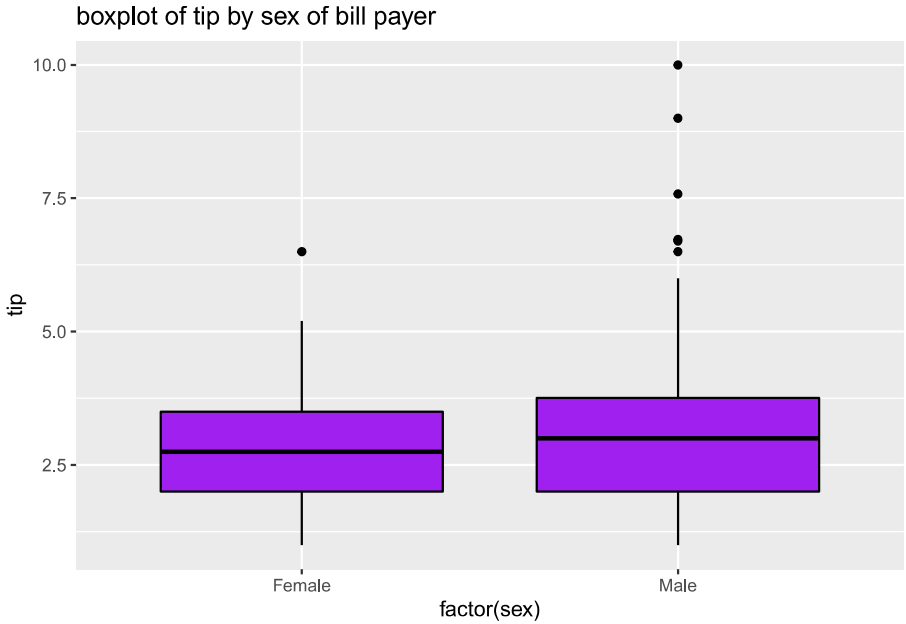
In this exercise, the data set `tips` from the package `reshape` will be used.

One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

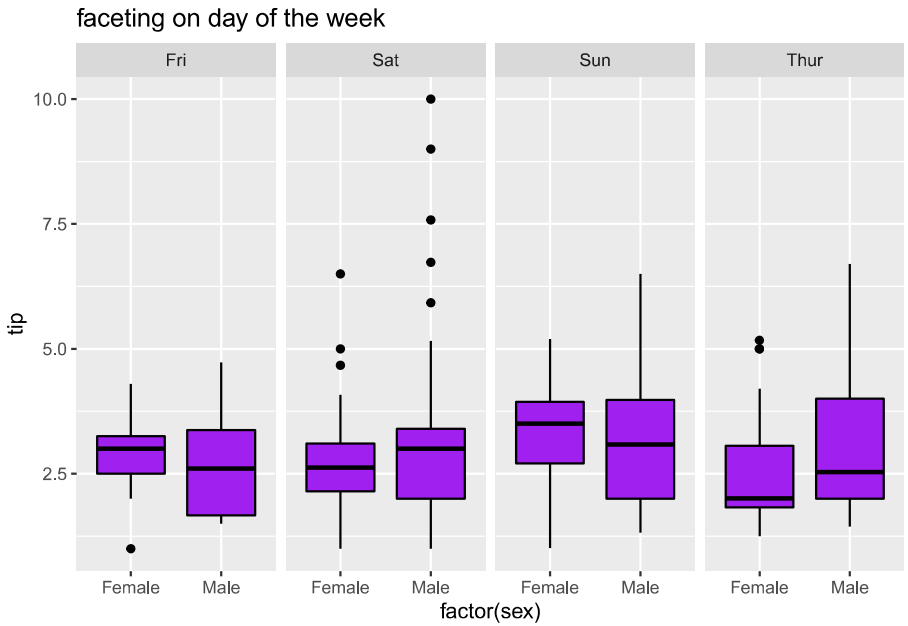
- `tip`: tip in dollars
- `total_bill`: bill in dollars
- `sex`: sex of the bill payer
- `smoker`: whether there were smokers in the party
- `day`: day of the week
- `time`: time of the day
- `size`: size of the party

In all he recorded 244 tips.

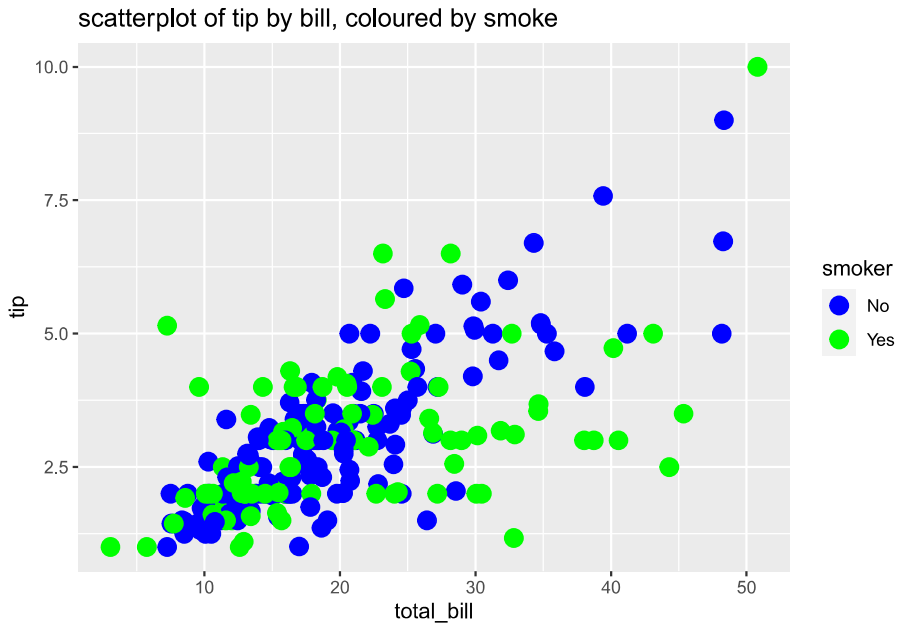
1. Create a boxplot for the tip by sex of the bill payer (use a nice color).



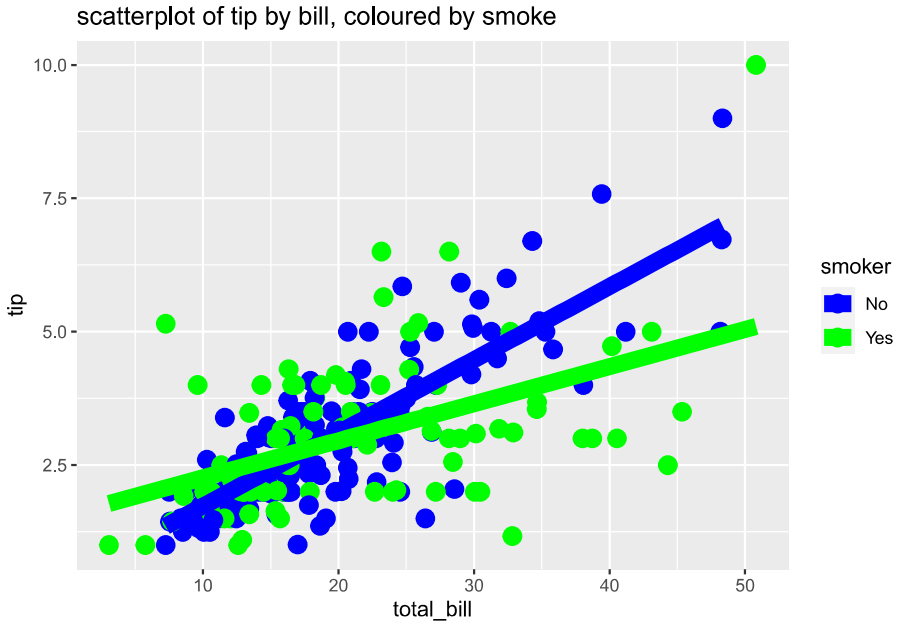
2. Create a matrix of panels of boxplots by using as faceting variable day of the week.



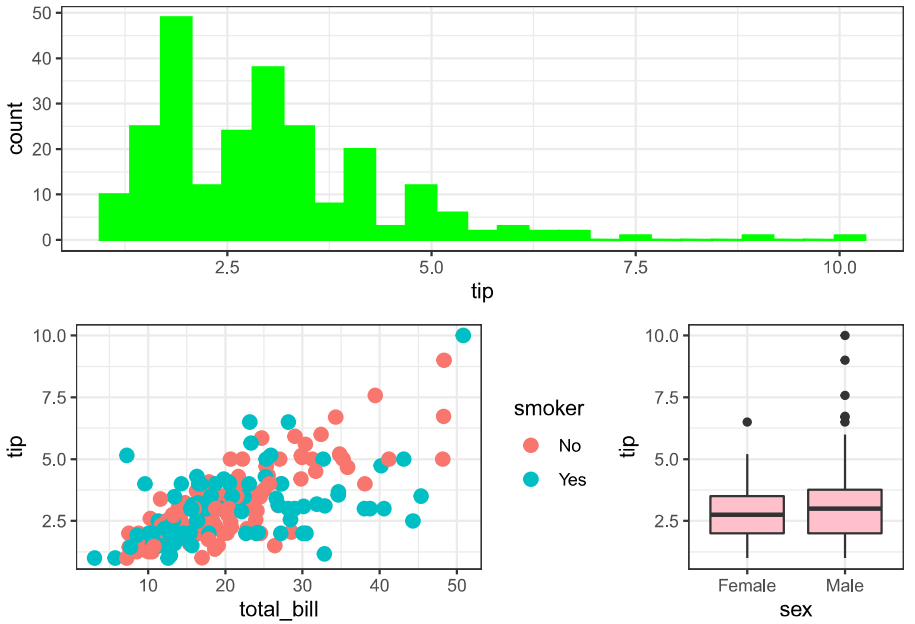
3. Create scatterplot of amount of tip by total amount of the bill. Use different colours for smokers (= green) and non-smokers (=blue).



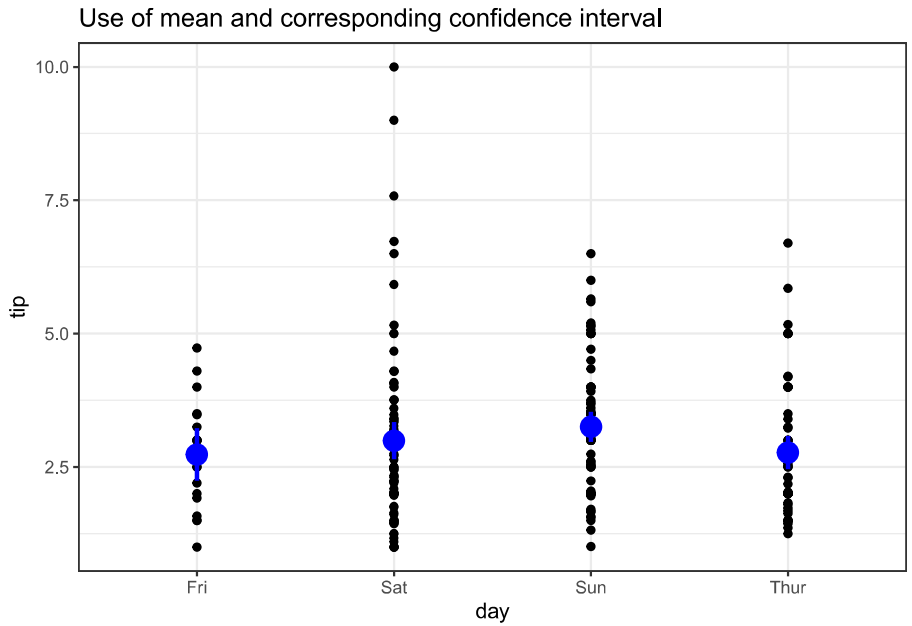
4. Based on previous scatterplot, add two regression lines.



5. Make
   - A histogram of `tip` (in green). Give this object the name `hist1`
   - A boxplot of `tip` by `sex`. Give this object the name `boxplot1`
   - Give the scatterplot of `tip` by `total_bill` and give it the name `scatter1`
   - Plot all of these 3 plots on one and the same graphical window as below



6.
   - Make a scatterplot of `tip` (on Y axis) versus `day` (on X axis).
   - Add the mean tip value and corresponding confidence interval (from bootstrap). Use a blue color.
   - Add a title 'Use of mean and corresponding confidence interval'.
   - Use black and withe background.



7.2 `galileo` data

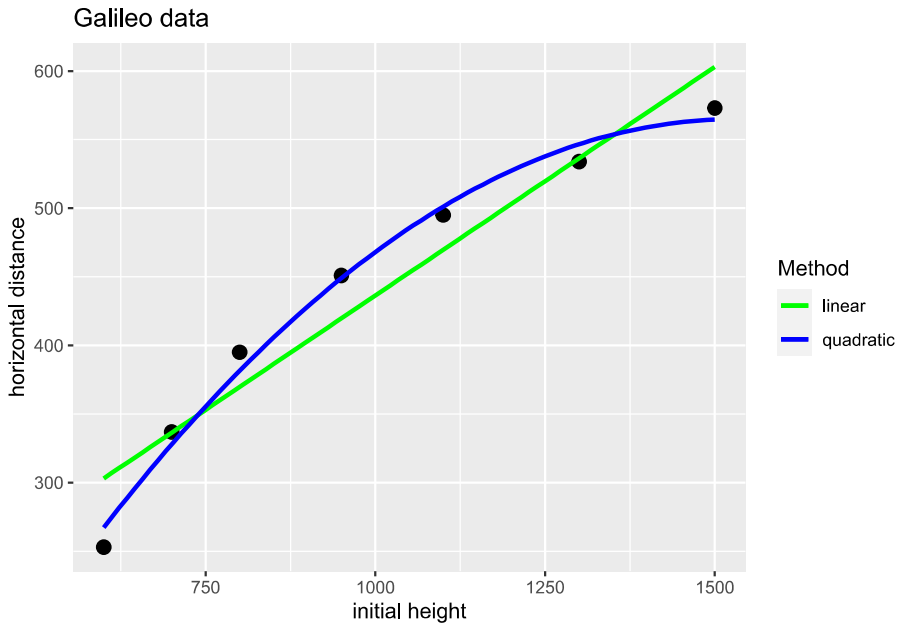
In this exercise, we use the `galileo` data set from the package `UsingR`.
This data frame has 7 observation on the following 2 variables

- `init.h`: Initial height of ball
- `h.d`: Horizontal distance traveled.

```
galileo
```

```
##   init.h h.d
## 1    600 253
## 2    700 337
## 3    800 395
## 4    950 451
## 5   1100 495
## 6   1300 534
## 7   1500 573
```

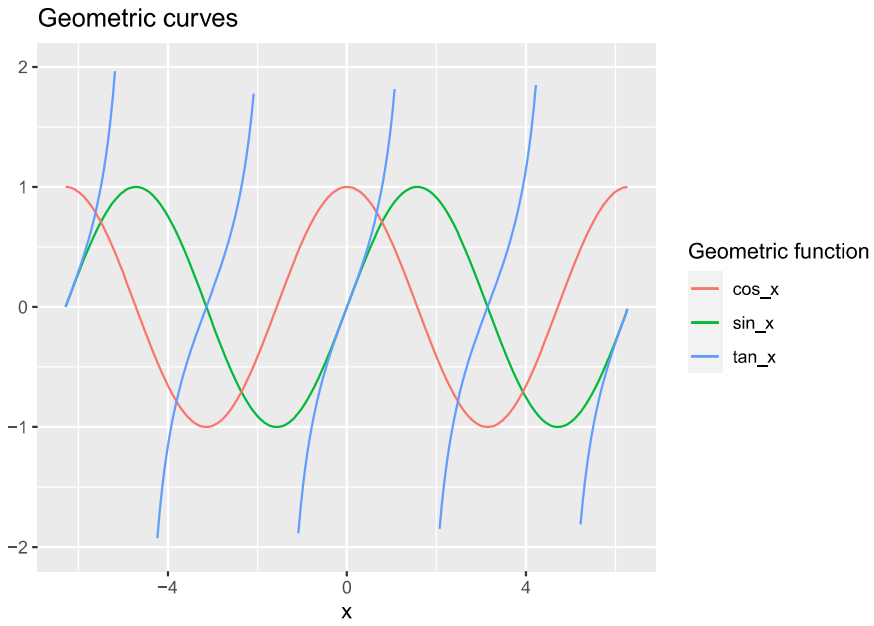
1. Make a scatterplot of initial height (X) versus horizontal distance (Y)
2. Add linear regression line
3. Add quadratic regression line (use *formula = y ~ x + I(x^2)* for this)
4. Add title, labels, legend in order to obtain the following graph



7.3 Geometric curves

We want to overlay a cosine, sine and tangent function for a sequence of x values (from -2π to $+2\pi$) and create the following plot:

- Generate a set of x values from $-2pi$ to $+2\pi$. (The value π is a known constant in R).
- Construct 3 vectors with the values for $\sin(x)$, $\cos(x)$ and $\tan(x)$
- Put all these vectors in one data frame.
- Create the plot


Geometric curves
2
1
0
−1
−2
−4
0
4
x
Geometric function
cos_x
sin_x
tan_x