(A small) R Guide for the Beginners with Applications for Decision Making

Intended to be used on the course
"Data Science for Business",
Aalto University, School of Business

Contents

0. Preparing the working space

As of now there are several open source environments to work with R and at least two of them are meant to work only with R. In any case you need to install the base system first.

0. Go to http://cran.r-project.org/mirrors.html and choose one of the mirrors. Usually, the closer the mirror to you geographically, the higher the download speed.

1. Click the mirror link you prefer. In the section "Download and Install R" you can choose the preferred version. Currently R is available for Mac, Windows, and Linux platforms.

2. Choose a version according to your platform.

    - If you are installing R for the first time on Windows, then the easiest way is to click "This is what you want to install R for the first time." Then click "Download R X.X.X for Windows"

    - In case you are using Mac, choose the topmost link "R-X.X.X.pkg". If you are using an older version of MacOS the other links as well. For Linux, follow the instructions found in /debian/ README.html

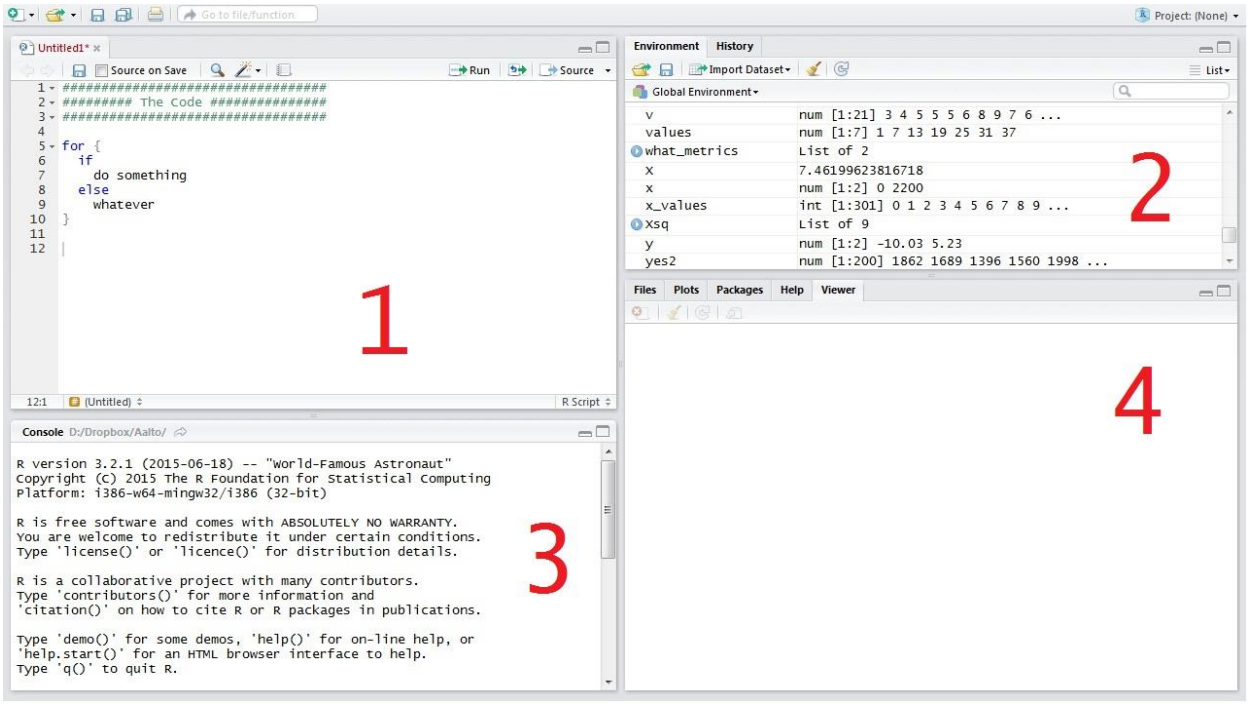
3. Unless you are using Linux, the file you download will include the GUI (graphical user interface) for R. So you can start using R immediately after the installation.

4. Now let's proceed with the installation of RStudio, which is a more user friendly software. Go to the page: https://www.rstudio.com/products/rstudio/download/ (you need to have the base of R to use RStudio)

5. In the section "Installers for Supported Platforms" choose the version that suits your platform (Mac, Windows or Linux). Download it and install it.

0.1. Using RStudio

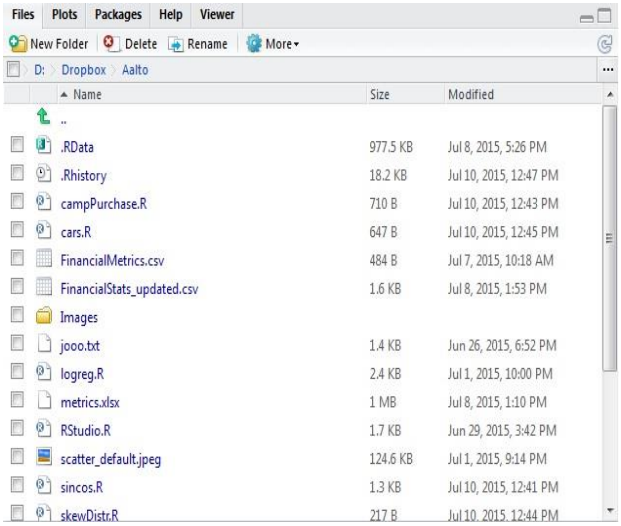
After installing RStudio the basic view should look like:

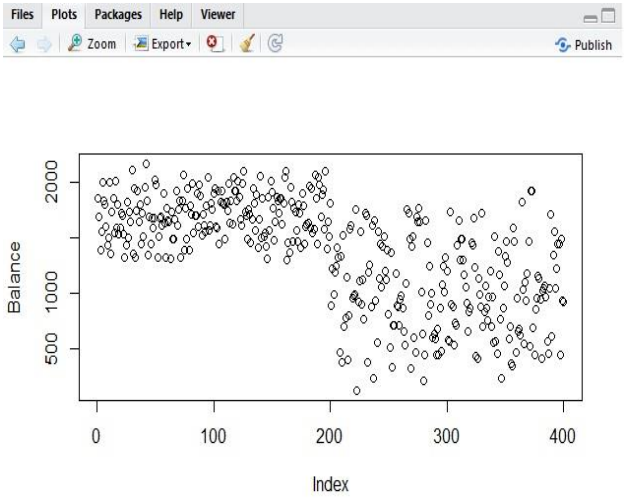


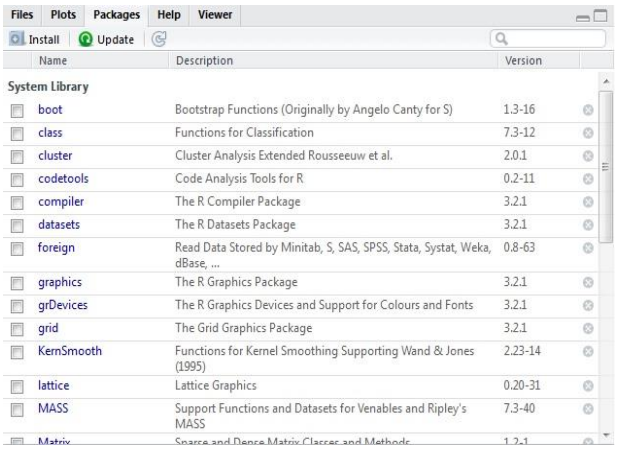
In the Section 1 you can write your code, functions and algorithms, and load saved code files. By default the file extension of the code files should be .R, but you can load code from the text files with different extensions, for instance .txt (caveat: unless the extension is .R you won't be able to use all the available features, one of them is code completion). In the Box 2 you can see all the data that has been created, loaded and modified previously. All the data and functions (at least in the RStudio version 0.99.465) can be viewed by clicking them and they open in the Box 1. In the Box 2 you can also view all the commands typed previously by choosing the tab "History". The values stored in the environment can be viewed by typing their names in the Box 3. In the Box 3 you can also type all the commands and write algorithms in the same way as in Box 1, even though it may be more practical to do that in the Box 1. Box 3 is also a log console; hence you can see all the actions being performed in the current session.

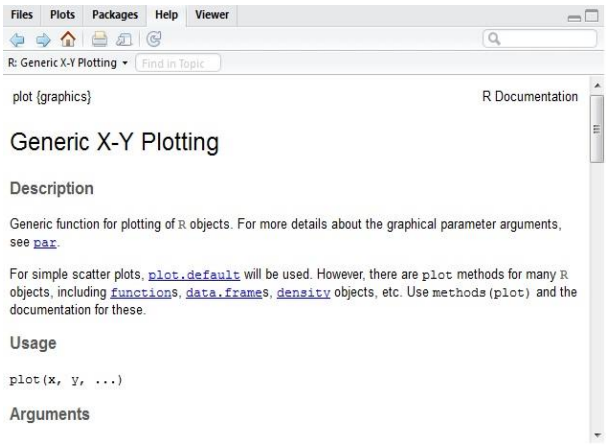
Box 4 is one of the biggest advantages of RStudio compared to the original R GUI. You can view files in your working directory, view the plots you created, manage the libraries that you are using, and surf

through the vast help documentation. It's a multifunctional section that is connected with the Box 3. If you feel unsatisfied with the default setup of the working interface, you can change it by choosing from the upper menu: "Tools" -> "Global Options..." -> "Pane Layout".









RStudio, just like R GUI is using a default working directory to load, save and store all the files that you need to use. To change the default working directory go to "Tools" -> "Global Options…" -> "General" -> "Default working directory". You can redefine the working directory later and change it if you create a new project. Creating new projects is simple and practical especially when one has multiple data sets preloaded. By creating separate projects you can ensure that you won't run out of memory. (Take this warning seriously in case you are working with bug datasets. Loading a dataset with the size of, say, 1 Gb, RStudio will reserve an equal amount of operative memory in your machine) Go to "File" -> "New Project…" to create new projects.

1. Data types, data classes and files

"R is a programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls, surveys of data miners, and studies of scholarly literature databases show that R's popularity has increased substantially in recent years." -Wikipedia.

This section is meant to introduce a reader to different types of data that can be processed in RStudio. Data types are introduced parsimoniously and one is encouraged to practice more independently.

1.1. Logical operators, comparison and truth values

In R there is a possibility to compare values with the following operators: "<" standing for less, ">" for more, ">=" for more or equal, "<=" for less or equal, and "==" for equal. Operator "=" is meant for inserting the values and therefore it can't be used for the comparison. The comparison operators print out TRUE or FALSE and the result can be stored in the variable. Also, logical operators such as "&" standing for AND, "|" for OR, and "!" for negation are in use. All logical and comparison operators can be used in the same sentence. All variables have different types depending on the values stored in them. You can always check the type of the variable with the function *class()*.

Example 1.....

```
> 0>1
[1] FALSE

> 0=1
Error in 0 = 1 : invalid (do_set) left-hand side to assignment

> 0==1
[1] FALSE

> a <- 0 != 0

> a
[1] FALSE

> a == FALSE
[1] TRUE
```

Example 2.....

```
> (1==0) & (1>0)
[1] FALSE

> (1==0) | (1>0)
[1] TRUE
> !(1==0)
[1] TRUE

> TRUE | FALSE
[1] TRUE

> !TRUE
[1] FALSE

> TRUE<FALSE
[1] FALSE

> TRUE>FALSE
[1] TRUE
```

1.2. Scalars

Scalars are the simplest data types and they can be input in R by simply typing the values.

Any variable can be assigned any value by using the operator "<-". Later the variable can be printed out by typing its name.

Example 3.....

```
> 42
[1] 42

> 0.42
[1] 0.42
```

Example 4.....

```
> a <- 42
> a
[1] 42

> b <- .42

> b
[1] 0.42

> c(a,b)
[1] 42.00  0.42
```

1.3. Arithmetic

In R it is possible to perform basic arithmetic operations such as addition, subtraction, multiplication, division, exponentiation, integer division and modulus calculation with real and complex numbers. Arithmetic operations can be performed with predefined existing variables.

Example 5.....

```
> c<-5
> d<-3

> c-d
[1] 2

> c+d
[1] 8

> e<-c+d

> e
[1] 8

> c-10
[1] -5

> c*d
[1] 15

> c/d
[1] 1.666667

> e%%d
[1] 2

> (e%%d)^d
[1] 8

> (e%%d)**d
[1] 8

> e%/%c
[1] 1


> 42%%5 #Division remainder calculator
[1] 2

> 42%/%5 #Division quotient calculator
[1] 8
```

1.4. Vectors

Vectors are one-dimensional matrices. Both scalars and vectors can be combined into new vectors with a combination function *c()*. A new vector variable can be saved into another variable in the same way as scalars. In fact, scalars are also vectors in R. Some simple data is better represented in vectors, which can be thought as values of a single variable. A function *seq()* can be used to create generic arithmetic sequences.

Example 6.....

```
> a<-c(0,2,4,6,8,10)
> a
[1]  0  2  4  6  8 10

> d<-c(12,3,4,a)
> d
[1] 12  3  4  0  2  4  6  8 10

> 0:10
 [1]  0  1  2  3  4  5  6  7  8  9 10

> seq(0,10,by=0.3)
 [1] 0.0 0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7 3.0 3.3
[13] 3.6 3.9 4.2 4.5 4.8 5.1 5.4 5.7 6.0 6.3 6.6 6.9
[25] 7.2 7.5 7.8 8.1 8.4 8.7 9.0 9.3 9.6 9.9

> 10:0
 [1] 10  9  8  7  6  5  4  3  2  1  0
```

Vector's cell can be referred to by giving its index in the square brackets after the name of the vector. N.B.: indexing starts from 1. Referring to an index that doesn't exist returns NA value. It is possible to refer to multiple indices at the same time by inputting a vector as an index. Function *length()* returns the length of a vector.

Example 7.....

```
> e<-c(5,10,5)
> e[2]
[1] 10

> e[10]
[1] NA

> length(e)
[1] 3

> e[length(e)]
```

```
[1] 5

> e[c(1,2)]
[1]  5 10

> e[c(2,3)]
[1] 10  5

> e[1:3]
[1]  5 10  5

> e[1:length(e)]
[1]  5 10  5

> e[1:4]
[1]  5 10  5 NA

> e[c(F,T,T)]
[1] 10  5

> e[c(-2)]
[1] 5 5
```

Changing all odd numbers to 42:

Example 8.....

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

> a%%2==0
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE

> a[!a%%2==0]=42
> a
 [1] 42  2 42  4 42  6 42  8 42 10

> a==42
 [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
```

1.5. Matrices

A matrix can be created from a vector or a set of vectors. For example a matrix $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$ can be created with a command *matrix()* in a following manner:

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9), ncol=3, nrow=3, byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

where the argument "byrow" takes values TRUE and FALSE depending on how we would like to organize the matrix, by rows or by columns. Argument "ncol" signifies the number of columns and "nrow" respectively the number of rows.

Matrices can be handled in same way as vectors. It is possible to refer to their cells, rows, and columns by using the square brackets after their name. Adding more rows or columns to the matrix can be done with the commands *rbind()* and *cbind()* respectively. Removing rows and columns can be done by using truth values in same way as vector operations.

```
> rbind(m, c(42,42,42))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   42   42   42

> cbind(rbind(m, c(42,42,42)), c(42,42,42,42))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3   42
[2,]    4    5    6   42
[3,]    7    8    9   42
[4,]   42   42   42   42
```

Example 9.....

```
> M <- array(1:10, dim=c(5,5))
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6    1    6    1
[2,]    2    7    2    7    2
[3,]    3    8    3    8    3
[4,]    4    9    4    9    4
[5,]    5   10    5   10    5
```

For example removing first three rows and columns can be done as follows

```
> M<- M[-1:-3,-1:-3]
> M
     [,1] [,2]
[1,]    9    4
[2,]   10    5
```

Or alternatively

```
> M<-M[c(F,F,F,T,T),c(F,F,F,T,T)]
> M
     [,1] [,2]
[1,]    9    4
[2,]   10    5
```

Hence, the result is

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

Multiplying matrices can be done using the operator "%*%", multiplication and division can be performed with "+" and "-", and the command *t()* transposes the matrix.

Multiplying:

Example 10......

```
> M1 <- matrix(c(1,2,3,4,5,6,7,8,9), ncol=3, nrow=3, byrow=T)
> M1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
> M2 <- matrix(c(1,2,3,4,5,6,7,8,9), ncol=3, nrow=3, byrow=T)
> M2
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> M3<-M1%*%M2
> M3
     [,1] [,2] [,3]
[1,]   30   36   42
[2,]   66   81   96
[3,]  102  126  150
```

Transposing:

Example 11. .....

```
> t(M3)
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

solve() function can be used to solve the systems of linear equations whose parameters are in the matrix form. The system of linear equations has to be in the form Ax=B. If B is not defined, then it is assumed to be the identity matrix, and the function will return the inverse of A.

Example 12. .....

Say we want to solve a system of equations as follows

$$\begin{cases} 3x + 2y - 4z = -4 \\ x - 6y + 3z = -4 \\ 2x - y + z = 5 \end{cases}$$

which corresponds to

$$\begin{bmatrix} 3 & 2 & -4 \\ 1 & -6 & 3 \\ 2 & -1 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 & 2 & -4 \\ 1 & -6 & 3 \\ 2 & -1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -4 \\ -4 \\ 5 \end{bmatrix}$$

```
> A<-matrix(c(3,1,2,2,-6,-1,-4,3,1),ncol=3,nrow=3,byrow=F)
> B<-matrix(c(-4,-4,5),ncol=1,nrow=3,byrow=F)
> A
     [,1] [,2] [,3]
[1,]    3    2   -4
[2,]    1   -6    3
[3,]    2   -1    1

> B
     [,1]
[1,]   -4
[2,]   -4
[3,]    5

> solve(A,B)
     [,1]
[1,]    2
[2,]    3
[3,]    4



> solve(A)
            [,1]        [,2]      [,3]
[1,]  0.06976744 -0.04651163 0.4186047
[2,] -0.11627907 -0.25581395 0.3023256
[3,] -0.25581395 -0.16279070 0.4651163
```

1.6. Characters

In addition to numeric variables and Booleans it is also possible to store character values that consist of letters and numbers as a text. Sequences of characters can be input in single or double quotes and they can be stored in the same way as other values.

Example 13......

```
> string <- "RStudio"
> string
[1] "RStudio"
```

```
> data_structures<-c("matrix", "table", "list")
> data_structures
[1] "matrix" "table"  "list"
```

1.7. Factors

Factors are the type of values that are meant to be used with classifies variables, such as the sex “M” or “F”. A variable can be converted to a factor variable using the function *as.factor().*

Example 14......

```
> answer<-c("Y","N","N","N","Y")
> answer
[1] "Y" "N" "N" "N" "Y"

> answer<-as.factor(answer)
> answer
[1] Y N N N Y
Levels: N Y
```

1.8. Data frames

Data frames are used as the fundamental data structure by most of R’s modeling software. They are tightly coupled collections of variables which share many of the properties of matrices and lists. For instance, combining the numeric and character vectors will produce a data frame as an output. Data frames can be created using a function *data.frame()* and an argument “stringAsFactors = FALSE/TRUE” defines whether string value will be stored as characters or as factors. In fact, strings are characters in R terminology. This is another confusing part in R terminology apart from library/package thing.

Example 15......

```
> pizza <- c('Margherita', 'Kebab', 'Quattro stagioni','Frutti di
mare','Hawaii')
> isItalian <- c('Y','N','Y','Y','N')
> price <- c(4,5,5,6,4.5)
> pizzeria <- data.frame(pizza,isItalian,price, stringsAsFactors=TRUE)
> pizzeria
             pizza isItalian price
1       Margherita         Y   4.0
2            Kebab         N   5.0
3 Quattro stagioni         Y   5.0
4   Frutti di mare         Y   6.0
5           Hawaii         N   4.5N     4.5
```

```
> str(pizzeria)
'data.frame':     5 obs. of  3 variables:
 $ pizza    : Factor w/ 5 levels "Frutti di mare",..: 4 3 5 1 2
 $ isItalian: Factor w/ 2 levels "N","Y": 2 1 2 2 1
 $ price    : num  4 5 5 6 4.5
```

1.9. List

List is a data structure that provides a flexible method to store the variables of different types, classes, and lengths, which is not possible in a vector or a table. Trying to combine the variables from different classes in a vector converts them to the same class.

"If you are using a data frame the data types must all be the same otherwise they will be subjected to type conversion. This may or may not be what you want, if the data frame has string/character data as well as numeric data, the numeric data will be converted to strings/characters and numerical operations will probably not give what you expected. " Source: r-bloggers.com

Example 16......

```
> vector <- c('test', 1:5, 42)
> vector
[1] "test" "1"    "2"    "3"    "4"    "5"    "42"

> class(vector)
[1] "character"
```

Example 17......

```
> pizza <- c('Margherita','Quattro stagioni','Frutti di mare')
> ingredients <- c('Tomato sauce', 'Cheese', 'Basilica','Olive oil')
> price <- c(4:10)
> owner_height <- sqrt(32400)
> pizzeria <- list(pizza,ingredients,price,owner_height,3.14, c(T,F,T))
> pizzeria
[[1]]
[1] "Margherita"       "Quattro stagioni" "Frutti di mare"

[[2]]
[1] "Tomato sauce" "Cheese"       "Basilica"     "Olive oil"

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

```
[[4]]
[1] 180

[[5]]
[1] 3.14

[[6]]
[1]  TRUE FALSE  TRUE

> pizzeria[3]
[[1]]
[1]  4  5  6  7  8  9 10

> pizzeria[1:3]
[[1]]
[1] "Margherita"       "Quattro stagioni" "Frutti di mare"

[[2]]
[1] "Tomato sauce" "Cheese"       "Basilica"     "Olive oil"

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

Subgroups of lists are lists as well:

Example 18. .....

```
> second <- pizzeria[2]
> second
[[1]]
[1] "Tomato sauce" "Cheese"       "Basilica"     "Olive oil"

> second[1]
[[1]]
[1] "Tomato sauce" "Cheese"       "Basilica"     "Olive oil"
> class(second)
[1] "list

> is.list(second)
[1] TRUE

> second[2]
[[1]]
NULL

> second <- pizzeria[[2]]
> second
[1] "Tomato sauce" "Cheese"       "Basilica"     "Olive oil"
```

```
> second[2]
[1] "Cheese"

> class(second)
[1] "character"
```

Example 19......

```
> cars <- list(brand=c("Ford"), model=c("Mustang", "Taurus", "Tourneo",
"Transit"), body=as.factor(c("passenger","van")),
intro_year=c(1964,1986,1995,1965), generation=1:6,
serialno=sample(x=10000:10000000, size=4))
> cars
$brand
[1] "Ford"

$model
[1] "Mustang" "Taurus"  "Tourneo" "Transit"

$body
[1] passenger van
Levels: passenger van

$intro_year
[1] 1964 1986 1995 1965

$generation
[1] 1 2 3 4 5 6

$serialno
[1] 8181771  879340 1885626 9812503

> cars[[2]]
[1] "Mustang" "Taurus"  "Tourneo" "Transit"

> cars$model[2]
[1] "Taurus"

> cars$model <- c(cars$model, "Focus")
> cars$model
[1] "Mustang" "Taurus"  "Tourneo" "Transit" "Focus"

> cars$engine <- "V12"
> cars$engine
[1] "V12"

> class(cars$engine)
[1] "character"

> cars$engine <- as.factor(c(cars$engine, "V4", "V6", "V8", "V10"))
> cars$engine
```

```
[1] V12 V4  V6  V8  V10
Levels: V10 V12 V4 V6 V8

> class(cars$engine)
[1] "factor"
```

Removing an object from the list can be done in a following manner.

Example 20......

```
> remove(cars$engine)
Error in remove(cars$engine) :
  ... must contain names or character strings

> cars$engine <- NULL
> cars$engine
NULL

> cars <- cars[-7]
> cars$engine
NULL
```

1.10. Loading the data from an external file

By default R software is using a predefined directory for loading and storing the files, this directory can be checked with the command *getwd()*. A new default directory can be set using a command *setwd()* or by clicking "File" -> "Change dir" in the default R GUI and "Tools" -> "Global Options…" -> "General" in RStudio.

A data file can be loaded into R in many different ways, for example using the following command:

```
X <- read.table("exampledata.txt", header = TRUE, stringsAsFactors=FALSE)
```

Where X will be the name of the data frame and the argument header defines whether to use the default names of the variables/columns that are already in the file being loaded. If the data being loaded doesn't have column names or you set "header = FALSE", then R will use the default column names look like "V1", "V2",… and so on. By default, strings in the data are treated as factors. Setting the option "stringsAsFactors=FALSE" will make sure that the strings are being loaded as characters. A column can be converted to factors afterwards by typing:

```
X$Column <- factor(X$Column)
```

where "Column" is any variable in the data set.

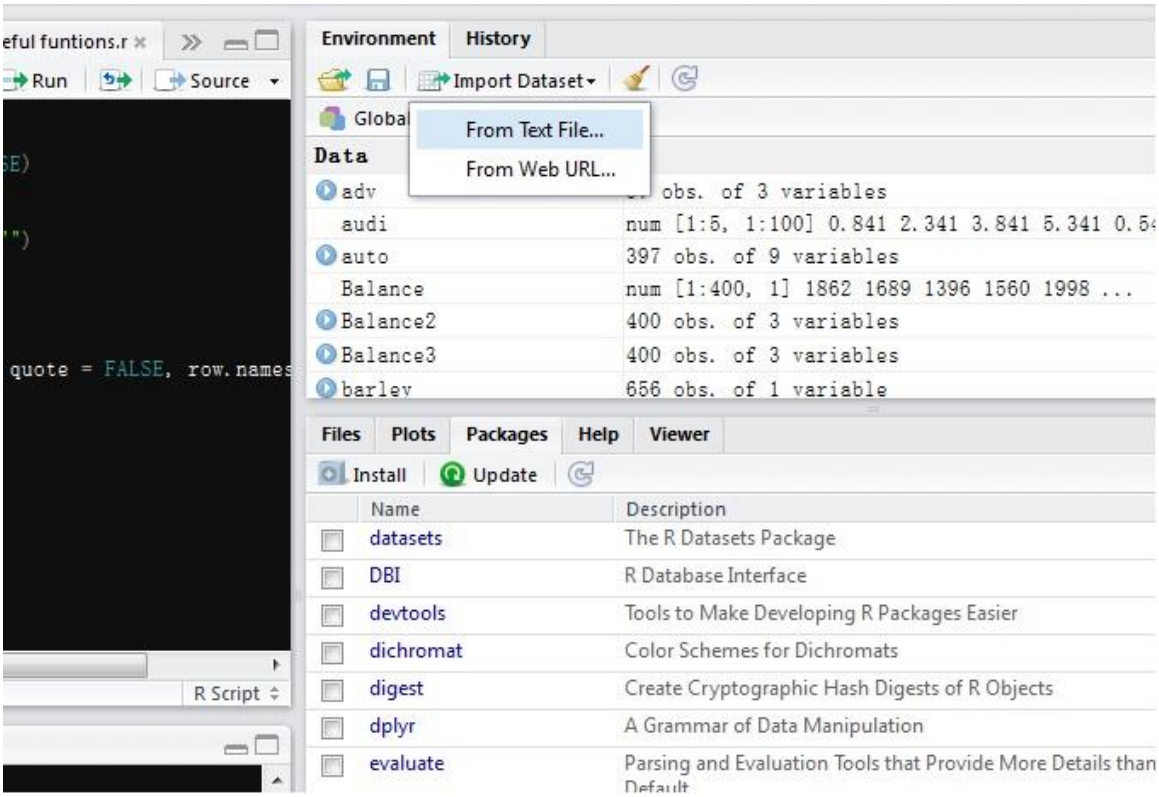
The files can also be loaded from outside of the working directory in the following way:

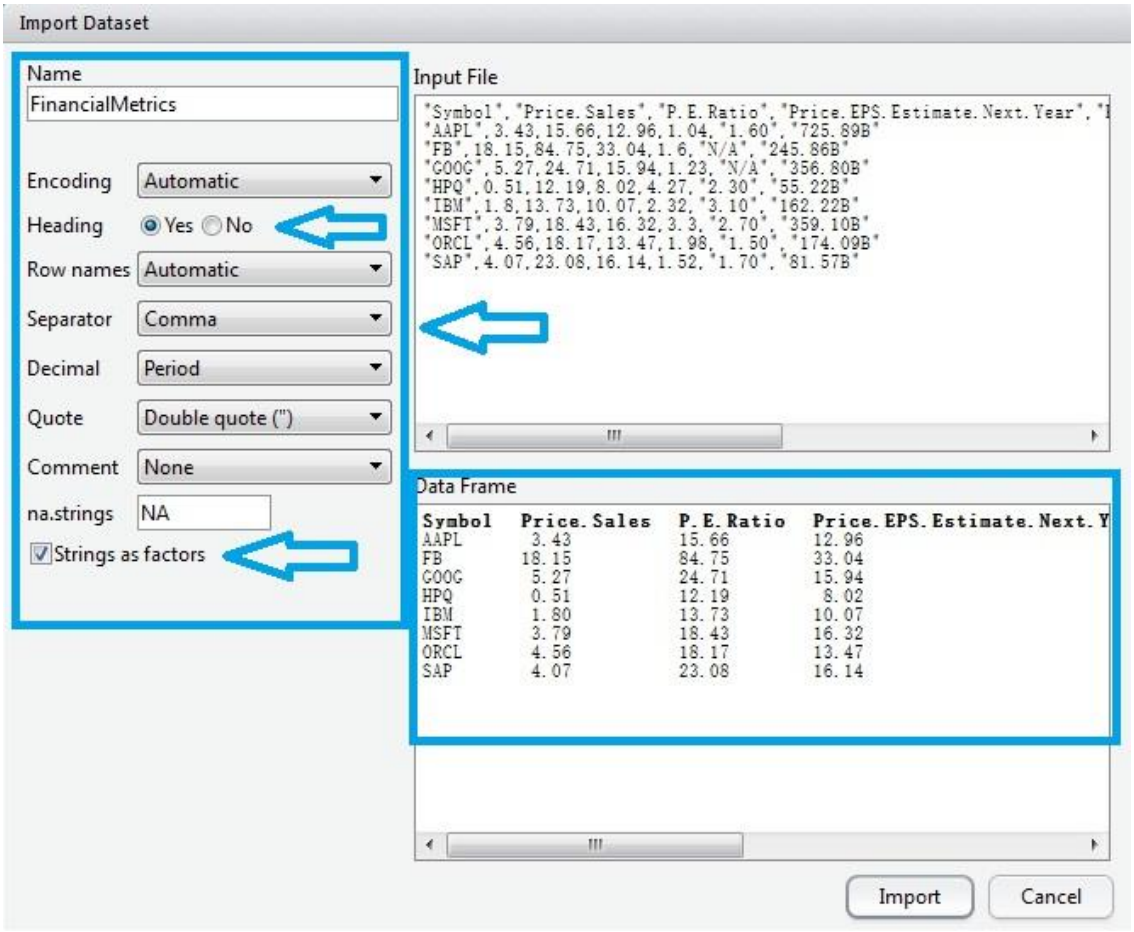
```
X <- read.table(file.choose(), header = TRUE)
```

After that the software will ask you to choose a file manually from the system.
For the CSV (comma-separated values) files use the following format instead:

```
X <- read.csv(file.choose(), header = TRUE)
```

In RStudio you can load the file interactively clicking: “Environment” -> “Import Dataset” -> “From Text File…” and by choosing a file from a directory. After choosing a file, you will be provided with several options to import the file. The good thing is that you can see all the changes you make before the import in the window “Data Frame”.





Default data extensions in R are .csv and .txt. If you want to load .xlsx files you have to import them into R using special importing functions. The 'xlsx' package has a function *read.xlsx()* for reading Excel files:

```
install.packages("xlsx")
library (xslx)
data <- read.xlsx ("data.xlsx", 1)  #instead of "data.xlsx" you can write
                                    #file.choose() to choose a file from a
                                    #different directory. The second argument
                                    #defines the number of sheet being loaded
```

Package 'xlsx' depends on a package 'rJava', which is being loaded automatically. If you get an error loading 'rJava', it is probably due to the outdated version of Java you are using. Head to http://www.java.com/en/download/ to download the latest version. If it seems like RStudio can't find where Java is installed use the following code to point it (change the location if needed):

```
Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre7') # for 64-bit version
Windows
Sys.setenv(JAVA_HOME='C:\\Program Files (x86)\\Java\\jre7') # for 32-bit
version Windows
```

Package ‘gdata’ has a method to load older .xls files:

```
install.packages("gdata")
library(gdata)
data <- read.xls("datafile.xls")    #reads the first sheet by default.
                                    #Otherwise use argument sheet to define
                                    #the number of sheet needed
```

Also in excel software it is possible to save the file in .csv or .txt format.

Loading data from other files is possible as well. Package ‘foreign’ can handle files from SPSS and other sources:

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

Use following functions to load files depending on your source.

- SPSS: read.spss()
- Stata: read.stata()
- MATLAB and Octave: read.octave()
- SAS: read.xport()

Example 21......

data: Auto.csv from http://www-bcf.usc.edu/~gareth/ISL/data.html

```
> getwd()
[1] "C:/WINDOWS/system32"

> setwd("C:/R")
> getwd()
[1] "C:/R"

> data = read.csv(file.choose(), header = TRUE, stringsAsFactors=FALSE)
> data[1:5,1:5]
  mpg cylinders displacement horsepower weight
1  18         8          307        130   3504
2  15         8          350        165   3693
3  18         8          318        150   3436
4  16         8          304        150   3433
5  17         8          302        140   3449
```

1.11. Referring to cells, rows and columns, choosing partial data

A data frame's cells, rows and columns can be pointed in the same way as the ones of matrices. The columns of the data frame are vectors and they can be referred using numbers or their names (headers). The partial data can be chosen using different conditions. When applying a condition to a specific vector/column in the table the operator $ should be used for referring to this column.

Example 22......

data: Auto.csv from http://www-bcf.usc.edu/~gareth/ISL/data.html

```
> data[5,4]
[1] "140"

> data[1:5,4]
[1] "130" "165" "150" "150" "140"

> data[5,]
   mpg cylinders displacement horsepower
5   17         8          302        140

> data$weight[1:5]
[1] 3504 3693 3436 3433 3449

> View(data)
> class(data$cylinders)
[1] "integer"

> class(data$acceleration)
[1] "numeric"

> class(data$name)
[1] "character"

> data$name[data$cylinders==8 & data$weight<3300]
[1] "buick estate wagon (sw)" "chevrolet monza 2+2"     "ford mustang ii"
"ford futura"

> data$horsepower/100
Error in data$horsepower/100 : non-numeric argument to binary operator

> class(data$horsepower)
[1] "character"

> data$horsepower <- as.numeric(data$horsepower)
> data$horsepower[1:5]/100
```

```
[1] 1.30 1.65 1.50 1.50 1.40
```

Example 23.

```
> model <- c("Mustang", "Taurus", "Tourneo", "Transit")
> body <- c(1,NA,2,NA)
> body <- factor(body,labels=c("passenger","van"))
> intro_year <- c(1964,1986,1995,1965)
> serialno <- sample(x=10000:10000000, size=length(model))
> ford <- data.frame(model, body, intro_year, serialno)
> ford
    model      body intro_year serialno
1 Mustang passenger       1964   901917
2  Taurus      <NA>       1986  2676266
3 Tourneo       van       1995   660185
4 Transit      <NA>       1965  5949726
```

Example 24.

```
> ford[ford$body=='passenger',]
       model      body intro_year serialno
1    Mustang passenger       1964   901917
NA      <NA>      <NA>         NA       NA
NA.1    <NA>      <NA>         NA       NA

> nrow(ford[ford$body == 'passenger', ])
[1] 3

> subset(ford, body == 'passenger')
    model      body intro_year serialno
1 Mustang passenger       1964   901917

> nrow(subset(ford, body == 'passenger'))
[1] 1
```

Sometimes empty or NA values in the data can produce false results, especially if we want to dismiss the NA values completely. In such cases *which()* or *subset()* function can fix the problem.

Example 25.

```
> v < 4
[1]  TRUE  TRUE    NA FALSE  TRUE    NA

> v[v<4]
[1]  1  2 NA  3 NA
```

```
> length(v[v<4])
[1] 5

> sum(v<4, na.rm=T)
[1] 3
```

Example 26......

```
> which(v<4)
[1] 1 2 5

> v[which(v<4)]
[1] 1 2 3

> length(v[which(v<4)])
[1] 3

> length(which(v<4))
[1] 3
```

Example 27......

```
> subset(v, v<4)
[1] 1 2 3

> length(subset(v, v<4))
[1] 3
```

Example 28......

```
> M42 <- data.frame(matrix(1:42, nrow=7))
> colnames(M42) <- c("V1", "V2", "V3",
+                    "V4", "V5", "V6")
> M42
  V1 V2 V3 V4 V5 V6
1  1  8 15 22 29 36
2  2  9 16 23 30 37
3  3 10 17 24 31 38
4  4 11 18 25 32 39
5  5 12 19 26 33 40
6  6 13 20 27 34 41
7  7 14 21 28 35 42

> M42.345 <- subset(M42, select=c("V3", "V4", "V5"))
```

```
> M42.345
  V3 V4 V5
1 15 22 29
2 16 23 30
3 17 24 31
4 18 25 32
5 19 26 33
6 20 27 34
7 21 28 35

> M42.345.over10 <- subset(M42, V2 > 10, select=c("V3", "V4", "V5"))
> M42.345.over10
  V3 V4 V5
4 18 25 32
5 19 26 33
6 20 27 34
7 21 28 35
```

Function *paste()* can be useful if there's a need to create several columns that have consequent generic names.

Example 29......

```
> paste("column", 1:3)
[1] "column 1" "column 2" "column 3"

> paste("column", 1:3, sep="")
[1] "column1" "column2" "column3"

> paste("column", 1:3, sep="_")
[1] "column_1" "column_2" "column_3"

> paste("column", 1, letters[1:5], sep="")
[1] "column1a" "column1b" "column1c" "column1d"
[5] "column1e"

> paste("column", 1, LETTERS[1:5], sep="")
[1] "column1A" "column1B" "column1C" "column1D"
[5] "column1E"
```

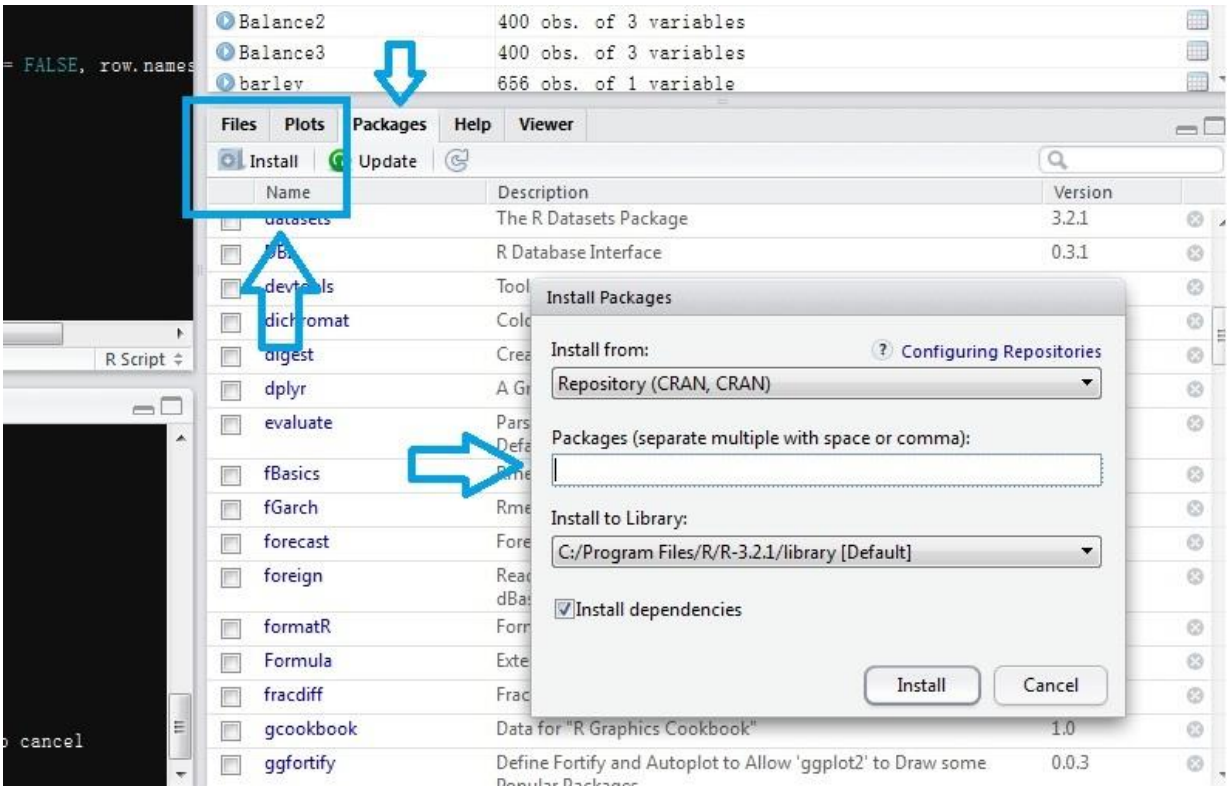
Example 30......

```
> M42.345 <- subset(M42, select=paste("V",3:5, sep=""))
> M42.345
  V3 V4 V5
1 15 22 29
2 16 23 30
3 17 24 31
4 18 25 32
```

```
5 19 26 33
6 20 27 34
7 21 28 35

> M42.135 <- subset(M42, select=paste("V", seq(1,6, by=2), sep=""))
> M42.135
  V1 V3 V5
1  1 15 29
2  2 16 30
3  3 17 31
4  4 18 32
5  5 19 33
6  6 20 34
7  7 21 35

> M42.345.v2even <- subset(M42, V2%%2==0, select=paste("V", seq(3,5,by=1),
sep=""))
> M42.345.v2even
  V3 V4 V5
1 15 22 29
3 17 24 31
5 19 26 33
   7  21 28 35
```

2. Statistical functions

There is a vast collection of functions for data analysis in R. The functions are included in different libraries which tend to have different purposes. The standard libraries are always included in R package, the rest can be downloaded by clicking “Packages” -> “Load package” in the standard R framework or simply by typing *library()* with the name of the package in the brackets. If you don’t have a package you need to use, then you have to install it first. You can do it by typing *install.packages()* in the dialog box with the name of the package in the brackets. Also you can do it interactively by choosing “Packages” -> “Install”:



You may be prompted to select a download mirror. You can either choose the one geographically closest to you, or, if you want to make sure you have the most up-to-date version of your package, choose the Austrian site, which is the primary CRAN server. Get back to the dialog box and pick the Austrian repository by typing:

```
setRepositories()
```

and by choosing a preferred action from the dialog box.

When you tell R to install a package, it will automatically install any other packages that the first package depends on. The content (all the functions) available in a package can be checked using a function *ls()*:

```
ls("package: the_name_of_the_package")
```

Please be aware that in addition to methods, tests and functions many packages include data sets which can be very useful for practicing.

Although one has to use the *library()* function to load a package, a package is not a library. A library is a directory that contains a set of packages. You might, for example, have a system-wide library as well as a library for each user.

2.1. Descriptives

There are plenty of built-in functions to get the descriptive statistics in R. Some of them are introduced below.

Example 31......

data: Auto.csv from http://www-bcf.usc.edu/~gareth/ISL/data.html

```
> mean(data$weight)
[1] 2970.262
> median(data$weight)
[1] 2800
> max(data$weight)
[1] 5140
> min(data$weight)
[1] 1613
> sd(data$weight)
[1] 847.9041
```

You may find a package *"pastecs"* useful as it includes a function for a table of comprised descriptive statistics.

Example 32......

```
> library(pastecs)
> options(digits=2)
> options(scipen=100)
> stat.desc(data$weight, basic=F)

 median         mean      SE.mean CI.mean.0.95          var
2800.00      2970.26        42.56        83.66    718941.40

std.dev     coef.var
 847.90         0.29
```

2.2. Basic functions for normal distribution

Example 33......

```
> rnorm(mean = 0, sd=1, n=10) #Simulating a normal distribution with zero
                              #mean,the standard deviation (sd) of one, n=10

 [1]  0.1611  0.0080 -0.0831  0.1452 -1.2257 -1.2628  0.0092  0.0437 -0.5330
-0.6463

> rnorm(mean = 0, sd=1, n=10)
 [1]  1.09 -1.16  0.14  0.92  1.31 -0.55 -0.39  1.46  1.50  0.61

> pnorm(q = -1, mean=0, sd = 1) #Cumulative distribution function

[1] 0.1586552539              #the point at quantile -1

> qnorm(mean=0, sd = 1, p = 1/3, lower=T) #quantile functions
[1] -0.43                     #1/3 of the probability mass is found to the
                              #left of this point

> dnorm(x=1, mean=0, sd = 1)  #Density functions
[1] 0.24                      #the solution of the equation f(x) with x=1
```

2.3. Apply() function and its sister functions

apply() function is especially practical in case of matrices or data frames. It allows to run operations row by row or column by column for the matrices.

apply() function has a following syntax:
apply(X, MARGIN, FUN, ...), where X is the data being elaborated, MARGIN indicates row if equal to 1 and columns if equal to 2, FUN is the function being used, … may include optional arguments for the function FUN.

lapply() performs the similar actions for the lists or vectors. The main difference between *apply()* and *lapply()* is what these two different functions print out. *lapply()* has a following syntax:
lapply(X, FUN, ...), where arguments correspond to the same purpose as in the case of *apply()*.

Descriptives of the partial data (especially factors) from a matrix or a data frame can be most easily done using the function *tapply()* which a following syntax: tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE), where X is an array, typically a column of the matrix, INDEX is an argument for factors, typically another column. If "simplify=FALSE", the output is a list, otherwise the output is a scalar. FUN is a function being applied.

Example 34. .....

```
> m <- matrix(rep(1:3,3),ncol=3)
```

```
> m
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3

> apply(m,1,sum)
[1] 3 6 9

> a <- apply(m,2,sum)
> a
[1] 6 6 6

> class(a)
[1] "integer"
```

Example 35......

```
> l <- list(1:3,4:6,7:9)
> l
[[1]]
[1] 1 2 3

[[2]]
[1] 4 5 6

[[3]]
[1] 7 8 9

> lapply(l,mean)
[[1]]
[1] 2

[[2]]
[1] 5

[[3]]
[1] 8
```

Example 36......

```
> ford$weight <- c(3139, 2700, 2600, 3200)
> ford$body[ford$model=="Taurus"] <- 'passenger'
> ford$body[ford$model=="Transit"] <- 'van'
> ford
    model      body intro_year serialno weight
1 Mustang passenger       1964   901917   3139
2  Taurus passenger       1986  2676266   2700
3 Tourneo       van       1995   660185   2600
```

```
4 Transit       van       1965  5949726   3200

> tapply(ford$weight, ford$body, mean)
passenger       van
     2920      2900
```

2.4. Creating functions

Even though R has a big set of functions, sometimes one may need to combine these functions. Writing own functions in R can be done most conveniently in the script environment. You can create a new script

- In RStudio: File -> New File -> R Script or Ctrl+Shift+N
- In R console: File -> New Script

Example 37.

```
theTruth <- function(N) { #A function that returns an N amount of 42's
  x <- rep(42,N)
  return(x)
}

> theTruth(10)
 [1] 42 42 42 42 42 42 42 42 42 42
```

Example 38.

```
gm_mean <- function(a){ #a function to calculate a geometric average
      prod(a)^(1/length(a))
}


> x<-seq(1:10)

> gm_mean(x)

[1] 4.528729
```

2.5. Algorithms

2.5.1. For loops

For loops can be used to create repetitive algorithms in R. It is important to keep in mind that *for* is a function in R like any other function and therefore it outputs a value at the end of its execution. Therefore it is important to use *return()* function to print out the value of the for function when it's needed. A misuse of the *for* function can lead to errors without the *return()* function. Let's use a simple *for* structure to demonstrate its action.

Example 39.

```
> for(i in 1:5){
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

In the previous example *for* function repeats itself 5 times according to the instructions in braces. Hence, as long as the index *i* is in the range from 1 to 5 (that is the condition for *for* function to continue the execution), the function prints it out. The instructions in braces don't necessarily have to be related to the index *i* itself. We could instead create the following algorithm as well:

```
> for(i in 1:5){
+ print(paste("This is number 42, it is being printed ", i ," times"))
+ }
[1] "This is number 42, it is being printed  1  times"
[1] "This is number 42, it is being printed  2  times"
[1] "This is number 42, it is being printed  3  times"
[1] "This is number 42, it is being printed  4  times"
[1] "This is number 42, it is being printed  5  times"
```

Example 40.

```
totSum <- function(data){ #A function that calculates the sum of an array of
                          #numbers
  sum=data[1]
  N=length(data)-1
  for (i in 1:N){
    sum=sum+data[i+1]
  }
  return(sum)
}
```

Calculating the total account balance of all customers (data: Credit.csv from http://www-bcf.usc.edu/~gareth/ISL/data.html):

```
> totSum(credit$Balance)
[1] 208006
```

For loops work well with the matrices and vectors, but sometimes it is more practical to use *sapply()* function instead, when the output is in a matrix or a vector form. The next example demonstrates the trick.

Example 41.

```
M42 <- matrix(1:42, nrow=6)
# "values" is the empty vector for the minimum values
values <- rep(0,7)
min_value <- function(x) { #The algorithm to calculate the minimum values of
                           #the matrix columns
  return(min(M42[,x]))
}
for(i in 1:10){
  values[i] <- min_value(i)
}
```

```
> M42
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    7   13   19   25   31   37
[2,]    2    8   14   20   26   32   38
[3,]    3    9   15   21   27   33   39
[4,]    4   10   16   22   28   34   40
[5,]    5   11   17   23   29   35   41
[6,]    6   12   18   24   30   36   42
```

```
> values
[1]  1  7 13 19 25 31 37
```

```
> sapply(1:7, function(x) min_value(x))
[1]  1  7 13 19 25 31 37
```

2.5.2.If else structure

Conditional sentences can be programmed using if else structure.

Example 42.

```
numCompare <- function(x,y)
 if(x > y) {
 cat(x,"is bigger than",y)
 } else {
```

```
 cat(x,"is not bigger than", y)
   }

#OR

numCompare <- function(x,y){
  if(x > y) cat(x,"is bigger than",y) else cat(x,"is not bigger than", y)
  }

> numCompare(10,12)
10 is not bigger than 12

> numCompare(12,10)
12 is bigger than 10
```

Example 43......

```
 numCompare2 <- function(x,y)
      if(x > y) {
            cat(x,"is bigger than", y)
       } else if(x<y){
            cat(x,"is smaller than", y)
       } else {
            cat(x,"is equal to", y)
       }

#OR

numCompare2 <- function(x,y){
  if(x > y) cat(x,"is bigger than", y) else if(x<y) cat(x,"is smaller than",
y) else cat(x,"is equal to", y)
}

> numCompare2(10,10)
      10 is equal to 10
```

3. Methods of numerical inference

3.1. Cross-tabulating

The function *table()* returns the amount of occurrences that each value gets in the data.

Example 44......

```
> v <- c(3,4,5,5,5,6,8,9,7,6,4,2,78,90,6,1,3,5,7,4,23)
> table(v)
```

```
v
 1  2  3  4  5  6  7  8  9 23 78 90
 1  1  2  3  4  3  2  1  1  1  1  1

> table(ford$body)

passenger       van
        2         2
```

Using the same function *table()* there is a chance to cross tabulate two variables. *Apply()* function and its sisters can be used in the same way for the crosstabs and the function *prop.table()* is a quick way to printout the proportions of the values row-wise or column-wise. The second argument in *prop.table()* defines the method of proportions, where 1 stands for the rows, 2 for the columns.

Example 45.

```
> table(ford$body, ford$intro_year)

           1964 1965 1986 1995
  passenger    1    0    1    0
  van          0    1    0    1

> table(ford$body, ford$intro_year)[2,2]
[1] 1

> table(ford$body, ford$intro_year)[2,]
1964 1965 1986 1995
   0    1    0    1

> cross.ford <- table(ford$body, ford$intro_year)
> cross.ford[1,1]
[1] 1

> cross.ford[,1]
passenger       van
        1         0
```

Example 46.

```
> apply(cross.ford, 1, sum)
passenger       van
        2         2
>
> prop.table(cross.ford, 1)

           1964 1965 1986 1995
  passenger  0.5  0.0  0.5  0.0
  van        0.0  0.5  0.0  0.5
>
> prop.table(cross.ford, 2)
```

```
           1964 1965 1986 1995
  passenger    1    0    1    0
  van          0    1    0    1
```

3.2. Classification of the data

The classification of the data is especially useful in case of continuous variables. *Cut()* function is one of the options that can be used. It returns the input as factors.

Example 47.

```
> cut(M42$V2, breaks=c(0,10,14))
[1] (0,10]  (0,10]  (0,10]  (10,14] (10,14]
[6] (10,14] (10,14]
Levels: (0,10] (10,14]

> table(cut(M42$V2, breaks=c(0,10,14)))

 (0,10] (10,14]
      3       4

> cut(M42$V2, breaks=c(0,4,8,12))
[1] (4,8]  (8,12] (8,12] (8,12] (8,12] <NA>
[7] <NA>
Levels: (0,4] (4,8] (8,12]

> table(cut(M42$V2, breaks=c(0,4,8,12)))

 (0,4]  (4,8] (8,12]
     0      1      4
```

Tapply() function can be used to perform operations for the columns that are being classified with the classes from another column.

Example 48.

```
> class <- cut(M42$V2, breaks=c(0,10,14))
> tapply(M42$V3, class, mean)
 (0,10] (10,14]
   16.0    19.5
```

3.3. T-test and confidence intervals

Example 49.

```
> norm <- rnorm(mean = 0, sd=1, n=1000)
> t.test(norm, conf.level=0.99)

      One Sample t-test

data:  norm
t = -0.1961, df = 999, p-value = 0.8446
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 -0.08721620  0.07490038
sample estimates:
   mean of x
-0.006157911

> t.test(norm, conf.level=0.99)$conf.int
[1] -0.08721620  0.07490038
attr(,"conf.level")
[1] 0.99

# replicate confidence interval test 100 times
confidence <- replicate(100, t.test(rnorm(n=100, mean=0, sd=1))$conf.int)
```

3.4. F-test (incomplete)

3.5. Tests for the variance (independence)

Example 50. .....

Let's assume a hypothetical situation where a company wants to test the efficiency of it's marketing method. In the first approach the company is using a conventional marketing method to reach its customers and in another approach it is using a new method. After trying both methods the company divides customers into two groups based on the marketing applied on them and picks 1000 random people from both groups. As a result the manager of the company gets the following data: in the group of a new marketing method 745 committed to buy a product and 255 didn't, while in another group 690 committed and 310 didn't. Assuming all other factors being fixed and both groups of people being homogeneous, how can we test whether there has been a significant positive impact using a new method?

Let's create a matrix with values as follows:

```
> sales <- matrix(c(745,255,690,310), ncol=2)
```

Naming the rows and the columns:

```
> dimnames(sales) <- list(Result = c("Commit", "No commit"),
+                         Marketing_method = c("New","Old"))
```

```
> sales
           Marketing_method
Result      New Old
  Commit    745 690
  No commit 255 310
```

Applying Pearson's Chi-squared test:

```
> chisq.test(sales, corr=F)

      Pearson's Chi-squared test

data:  sales
X-squared = 7.462, df = 1, p-value = 0.006302
```

Here the argument "corr" means Yates' continuity correction. As we can see the p-value is very small, less than 1%. The null hypothesis is that there's no difference in distributions of two groups and hence the small p-value speaks against the null hypothesis. Therefore the manager has a strong support for the new marketing method.

3.6. Bayesian inference

Let's augment the previous marketing example to calculate the conditional probability of purchasing the product and being reached by the new marketing campaign. Let's mark B=1 if a person has purchased the product and B=0 if she hasn't. Next let's mark M=1 if the person has been reached by a new marketing campaign (new and conventional combined) and M=0 if she has been reached by the old campaign. Based on the marketing data (customers reached by both campaigns) our manager estimated that a person buys the product when he has been reached by a new marketing campaign with a probability of 74.5%, in other words $P(B=1|M=1)=0.745$. And he didn't buy the product when he wasn't reached by the new campaign (but he has been reached by the conventional campaign) with a probability of 31%, in other words $P(B=0|M=0)=0.31$.

Example 51. .....

Assume that 12345 people were reached by the campaigns and out of them 34% were reached by the new campaign. Hence, 4197 out of customers reached by the campaigns were reached by the new campaign. We pick a random person from the customers who were reached by the marketing campaigns and notice that this person has purchased the product, what is the probability that this person has been reached by the new campaign?

You want to calculate the following probability:

$$P(M = 1|B = 1) = \frac{\sum P(M = 1 \cap B = 1)}{\sum P(B = 1)}$$

```
n <- 12345
 newCamp <- rbinom(n=n, size=1, prob=0.34)
 purchased <- numeric(n)
 for(i in 1:n) {
  if(newCamp[i] == 1) {
    purchased[i] <- rbinom(n=1, size=1, prob=0.745)
  } else {
    purchased[i] <- rbinom(n=1, size=1, prob=1-0.31)
  }
 }
 >
 > sum(newCamp * purchased) / sum(purchased)
 [1] 0.3586286
```

NB: in real life you wouldn't have to simulate the distribution. You could instead take the binary vector of the purchasing customers and run the function:

```
> sum(newCamp * purchased) / sum(purchased)
```

Example 52......

Based on the whole pool of potential customers (e.g. segmented by age or income) our manager has estimated that 11% of them have purchased the product. This group also includes people not being reached by the marketing campaign. The manager estimates that as much as 23% of the people from the whole pool of customers have been reached by a marketing campaign. We pick a random person the customer pool and notice that this has not purchased a product. What is the probability that this person has been reached by a marketing campaign?

Let's mark the probability that the person has been reached by a marketing campaign $P(M = 1)$ and $P(M = 0)$ if she hasn't. $P(B = 1)$ and $P(B = 0)$ respectively for the purchases. Now we want calculate the following probability:

$$P(M = 1|B = 0) = \frac{\sum P(M = 1 \cap B = 0)}{\sum P(B = 0)}$$

We know that $P(B = 0) = 1 - 0.11 = 0.89$ and $P(M = 1) = 0.23$. And the whole customer pool is $12345/0.23 = 53673$. In addition, based on the data the manager estimated that if customers are not reached by a marketing campaign they purchase the product with 37% probability.

```
total <- 53673
 camp <- rbinom(n=total, size=1, prob=0.23)
 no_purchase <- numeric(total)
 for(i in 1:total) {
   if(camp[i] == 1) {
     no_purchase[i] <- rbinom(n=1, size=1, prob=((12345-
sum(purchased))/12345))
   } else {
     no_purchase[i] <- rbinom(n=1, size=1, prob=1-0.37)
   }
 }
 >sum(camp * no_purchase) / sum(no_purchase)
[1] 0.1230405
```

4. Graphics

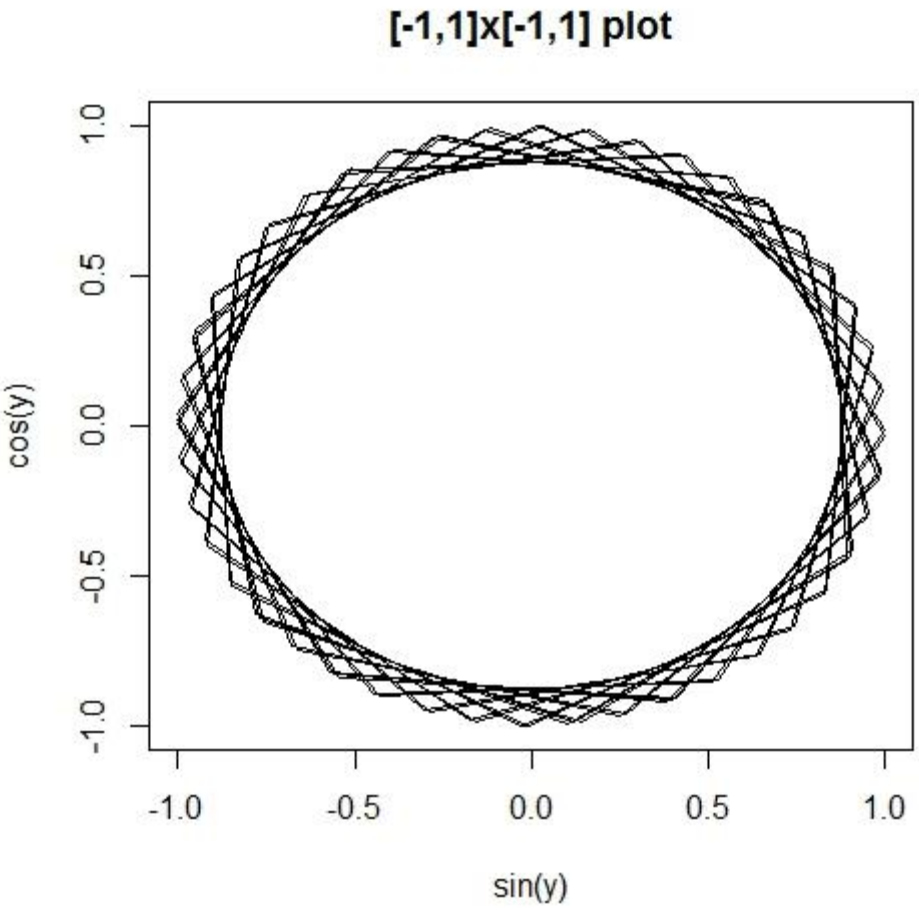
4.1. Basic graphics

You will never feel yourself alone with the collection of graphical tools in R.

Example 53......

Let's draw a graph of trigonometric functions as follows.

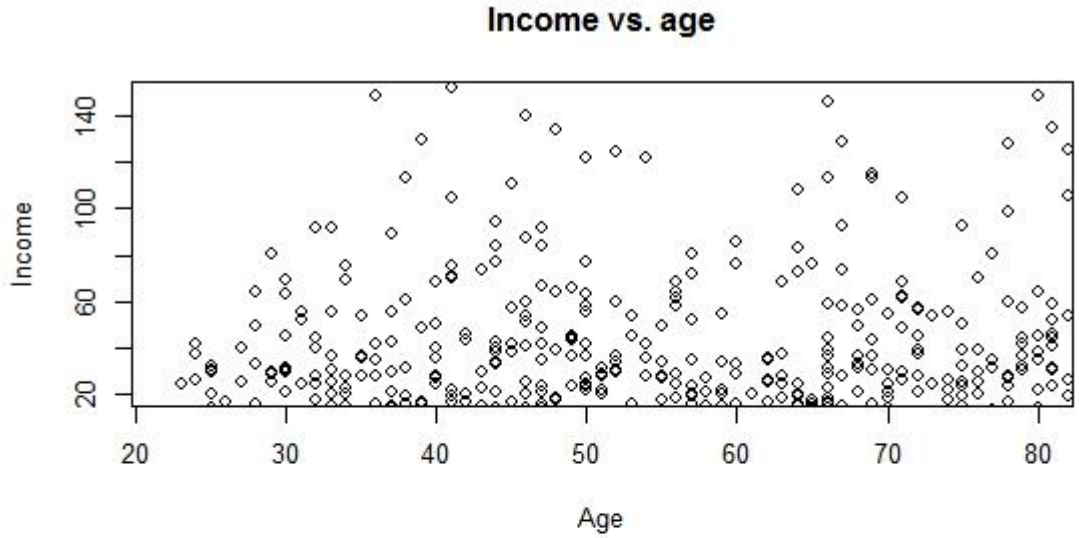
```
> plot(NULL, xlim=c(-1,1), ylim=c(-1,1), main="[-1,1]x[-1,1] plot",
xlab="sin(y)", ylab="cos(y)")
> y <- sapply(1:100, function(y) c(sin(y), cos(y)))
> lines(y[1,],y[2,])
```

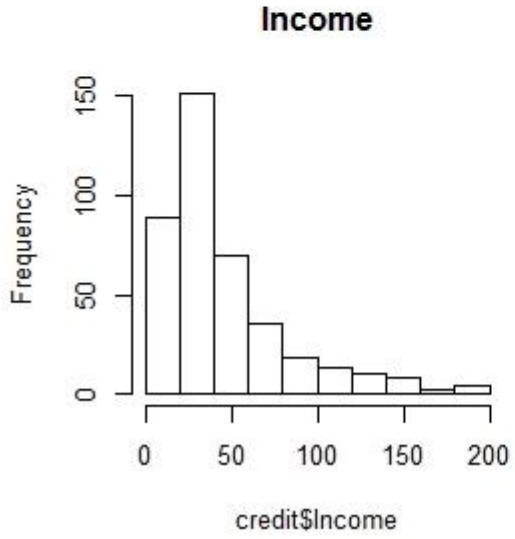


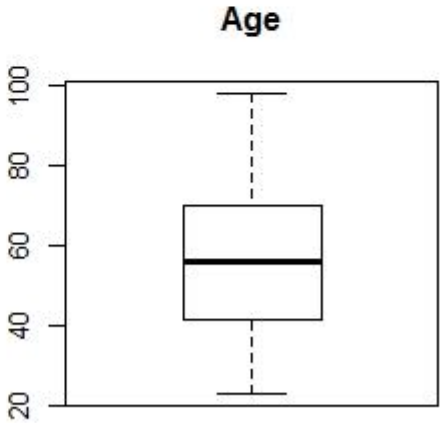
Example 54......

Data from: http://www-bcf.usc.edu/~gareth/ISL/Credit.csv

```
> credit=read.csv(file.choose(),header=TRUE,stringsAsFactors=F)
> View(credit)
>
> layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
>
> plot(credit$Age, credit$Income, main="Income vs. age", xlab="Age",
ylab="Income", xlim=c(22, 80), ylim=c(20,150))
>
> hist(credit$Income, main="Income")
>
> boxplot(credit$Age, main="Age")
```







4.2. Ggplot2

Start by installing the 'ggplot2', you will make yourself a big favor.

```
install.packages("ggplot2")
```

Ggplot2 is probably one of the most useful, intuitive and varied package for drawing graphics in R. It is based on the package 'lattice' and it's very useful for the multilayered plots.

Ggplot2 has a very good documentation here: http://docs.ggplot2.org/current/ and there's no way we could cover everything in this material as ggplot2 includes more than 250 functions and hundreds of other parameter variables. Instead, we'll go through the most commonly used functions of ggplot2.

4.3. Plotly (incomplete)

5. Decision-making methods for business

5.1. Simple linear regression

Linear regression is one of the most applied methods in statistical modeling. It's been in use for years, but still powerful enough in the modern business environment and its applications. Simple linear regression allows to estimate the relationship between two variables. Let's assume that we want to estimate how the budget consumed on TV ads affects sales of a company, in other words how much the sales change when spending on TV ads increases. The important prerequisite of the relationship estimation is that relationship is indeed linear. In case of two variables this can be checked graphically by plotting both variables in a 2-dimensional plane. Graphically the linear relationship between the variables would represent a straight line going up and right (in case of a positive relationship) or down and right (in case of a negative relationship). Mathematically the relationship can be represented as

$$\boldsymbol{Y} = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \boldsymbol{X}$$

Where $\boldsymbol{Y}$ is a (dependent) vector that represents data points of the variable of sales (the one that we want to explain with TV ads), X represents the vector that includes the data points of the (explanatory) variable of the consumption on TV ads, $\boldsymbol{\beta}_0$ and $\boldsymbol{\beta_1}$ are unknown, but fixed coefficients, intercept and slope, that we want to estimate with our model. Estimating these coefficients leads to the following model:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

or

$$\boldsymbol{Y} = \widehat{\boldsymbol{\beta}}_0 + \widehat{\boldsymbol{\beta}}_1 \boldsymbol{X} + \boldsymbol{\varepsilon}$$

where $\hat{\beta}_0$ and $\hat{\beta}_1$ are the estimators of the unknown β_0 and β_1, and $\hat{y}$ is the predictor of Y on the basis of x using pairs of observations

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

and $\boldsymbol{\varepsilon}$ is a vector of error terms, which geometrically simply represents the distance between the observations (points) and the straight line.

In other words

$$sales = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \times TV.$$

This model can be represented graphically as a straight line:

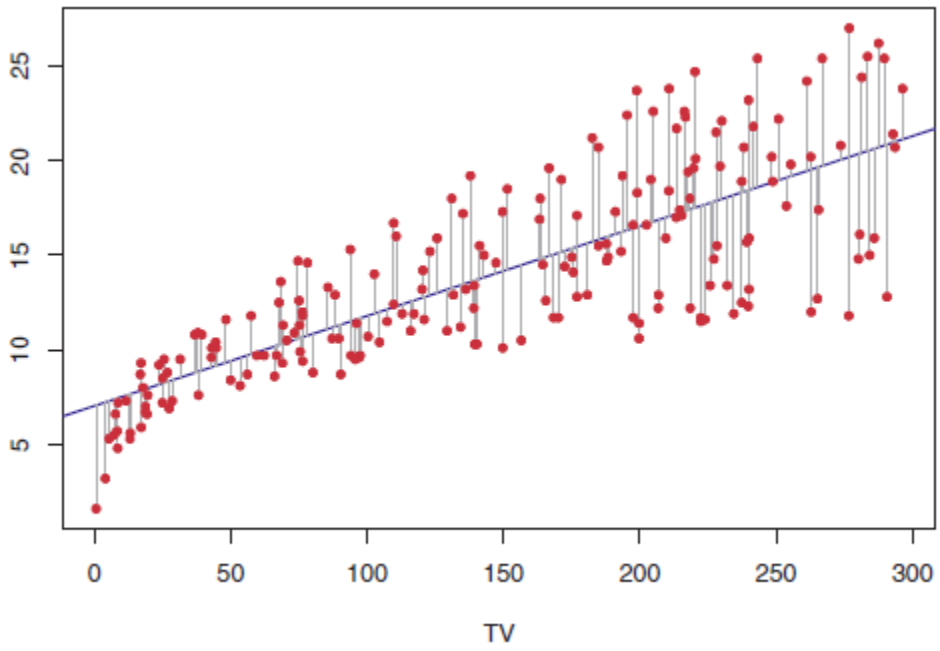


Figure 1. Source: Hastie et al.(2015)

There are many methods to estimate the coefficients of the model. But probably the most used one is by minimizing the distance between the observations and the straight line with ordinary least squares (OLS), where β_0 is simply the point where the straight line cuts the vertical axis. Therefore changing β_0 would mean moving the line up or down.

Apart from the graphical inspection we could set up a statistical hypothesis framework to test whether there is a (linear) relationship between the variables. Our hypothesis framework takes the following form:

$$H_0: There\ is\ no\ relationship\ between\ X\ and\ Y$$

versus

$$H_1: There\ is\ a\ relationship\ between\ X\ and\ Y.$$

Mathematically this corresponds to

$$H_0: \beta_1 = 0$$

versus

$$H_1: \beta_1 \neq 0,$$

since if $\beta_1 = 0$ then our model reduces to $Y = \beta_0 + \varepsilon$.

In practice $\hat{\beta}_1$ can be assessed by computing the t-statistic given by

$$t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)},$$

which simply measures how far away (how many standard deviations) $\hat{\beta}_1$ is from 0 (recall that the normal distribution is has a bell shape). From t-statistic we can deduct a p-value which is the probability of attaining such a coefficient estimate simply by chance. Hence, the smaller the p-value, the higher there is a chance that there is a relationship between the variables in case. Typically we would reject the null hypothesis when the p-value is lower than 0.05. Recall that correlation

$$Corr(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

is also a measure of the linear relationship between the variables.

If $\hat{\beta}_1$ turns out to be statistically significant one can move further and assess the model we created. Again, graphical inspection is one of the methods that should work for the beginning. The most common statistical units for model estimation are residual standard error (RSE) and R squared (R^2) . Residual standard error measures the average amount that the response will deviate from the true regression line and it is computed using the formula

$$RSE = \sqrt{\frac{1}{n-2}RSS} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

where

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

And R squared is defined as

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS},$$

where

$$TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

is the total sum of squares.

In a simple linear regression $R^2 = [Corr(X,Y)]^2$. However this doesn't necessarily apply in the framework of the multiple linear regression.

N.B. The linear model is always linear as long as its coefficients remain linear. For instance, a model

$$Y = \beta_0 + \beta_1 X^2$$

is also linear.

Example 55. .....

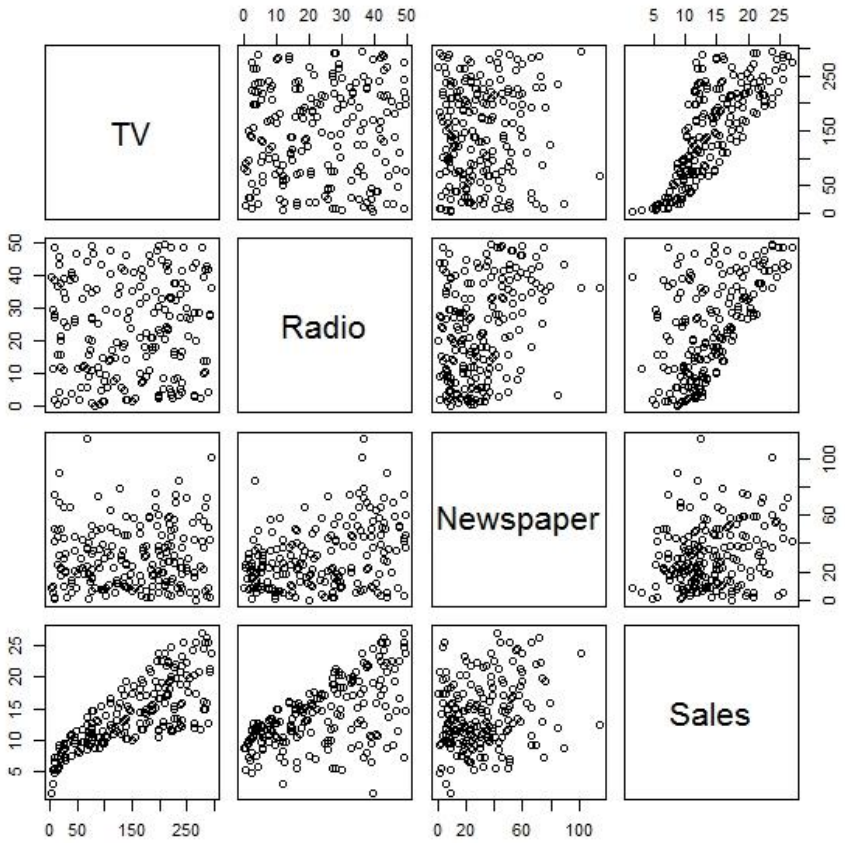
We start by loading the data into R:

```
>adv=read.csv(file.choose(), header=TRUE)
```

This is the data set provided by Gareth James, you can download it here (Advertising.csv): http://www-bcf.usc.edu/~gareth/ISL/data.html

```
>head(adv, n=5)
      TV Radio Newspaper Sales
1  230.1  37.8      69.2  22.1
2   44.5  39.3      45.1  10.4
3   17.2  45.9      69.3   9.3
4  151.5  41.3      58.5  18.5
5  180.8  10.8      58.4  12.9
```

```
>plot(adv)
```



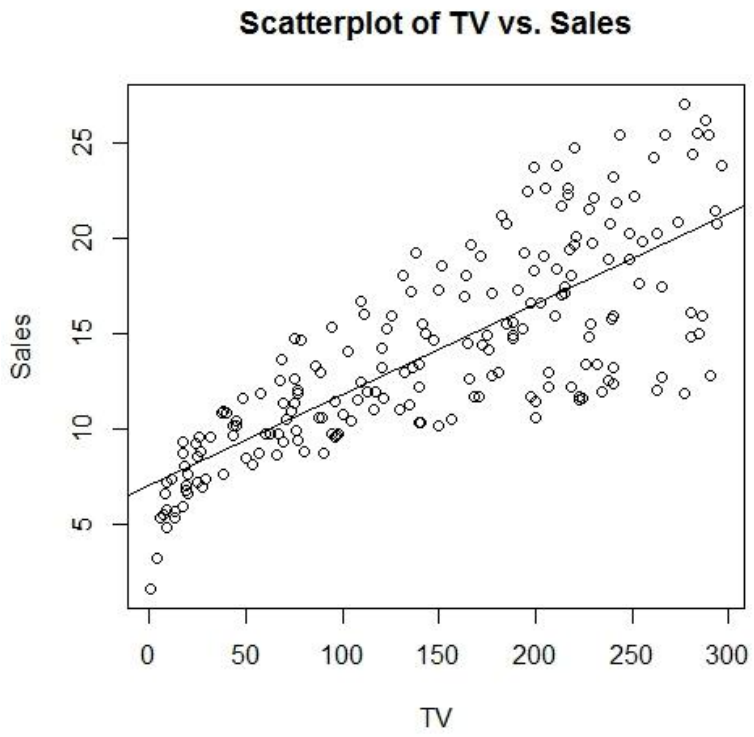
We want to explore how the variables "TV" and "Sales" are correlated to each other. To do that we'll run a linear OLS regression:

$$sales = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \times TV.$$

But first, let's plot TV vs. Sales in a scatterplot.

```
>plot(adv$TV,adv$Sales, xlab="TV", ylab="Sales", main="Scatterplot of TV vs.
Sales")
```

```
>abline(lm(adv$Sales ~ adv$TV))
```



As you can see, there is some positive (why?) correlation. And now the regression statistics:

```
>SalesTV=lm(Sales~TV, adv)
>summary(SalesTV)
Call:
lm(formula = Sales ~ TV, data = adv)

Residuals:
    Min      1Q  Median      3Q     Max
-8.3860 -1.9545 -0.1913  2.0671  7.2124

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 7.032594   0.457843   15.36   <2e-16 ***
TV          0.047537   0.002691   17.67   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.259 on 198 degrees of freedom
Multiple R-squared:  0.6119,  Adjusted R-squared:  0.6099
F-statistic: 312.1 on 1 and 198 DF,  p-value: < 2.2e-16
```

Both β_0 and β_1 are statistically significant at the 99,9% level. Hence, we can conclude that there is a correlation based on OLS regression.

Using the maximum likelihood estimators which are $\beta_0 = 7.033$ and $\beta_1 = 0.048$ we can draw the regression line manually using *lines()* function:

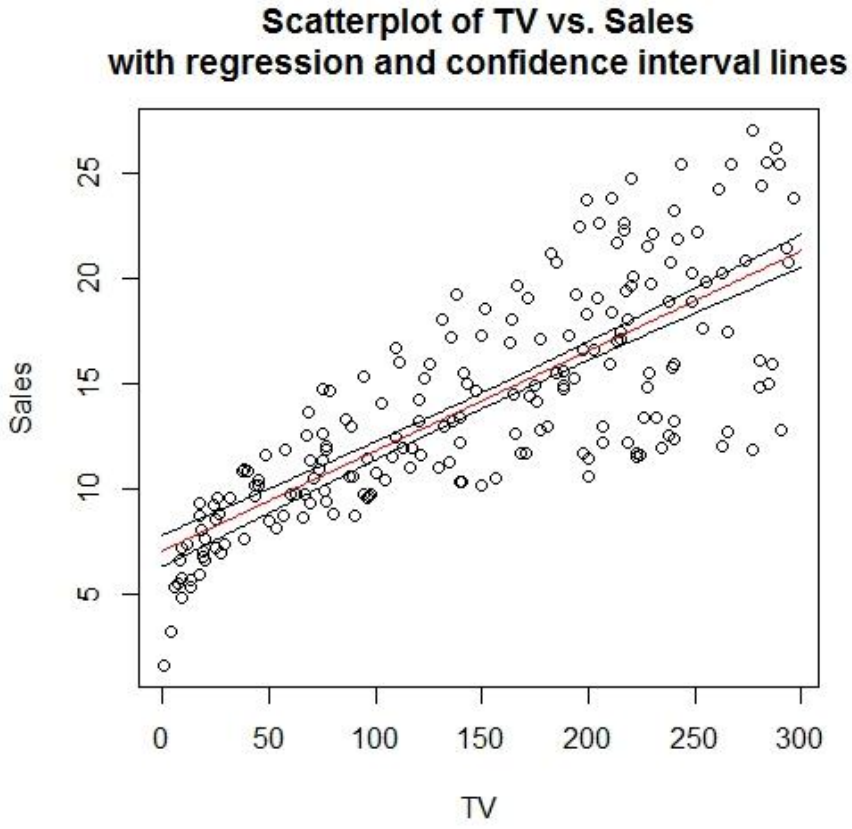
```
> x <- c(0,300)
> y <- 7.033 + 0.048*x
> lines(x,y, col="red")
```

Or *curve()* function:

```
> curve(7.033 + 0.048*x, from=0, to=50, add=T, col="blue")
```

Let's put 90% confidence intervals for the regression line:

```
newx <- seq(0,300)
pred <- predict(SalesTV, data.frame(TV=newx), interval = "confidence", level
= 0.90, type="response")
lines(newx, pred[,2], col="black")
lines(newx, pred[,3], col="black")
lines(newx, pred[,1], col="red")
```



What are the confidence intervals for the coefficients?

```
>confint(SalesTV)
              2.5 %     97.5 %
(Intercept) 6.12971927 7.93546783
TV          0.04223072 0.05284256
```

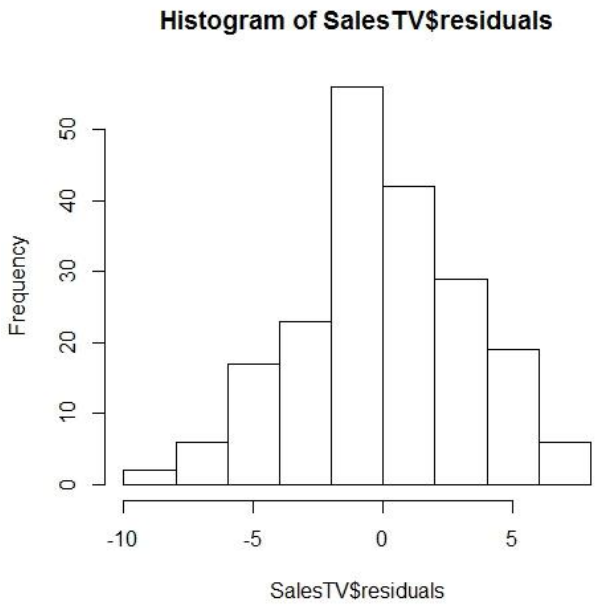
Hence, for example the 95% confidence interval for β_0 is [6.13,7.94].

Predicting three first values of "Sales" based on "TV":

```
>predict(SalesTV,data.frame(TV=(c(1,2,3))),interval="confidence")
      fit      lwr      upr
1 7.080130 6.181837 7.978423
2 7.127667 6.233947 8.021387
3 7.175203 6.286048 8.064359
```

How well is sales explained with this linear model? Plot of the residuals:

```
>hist(SalesTV$residuals)
```



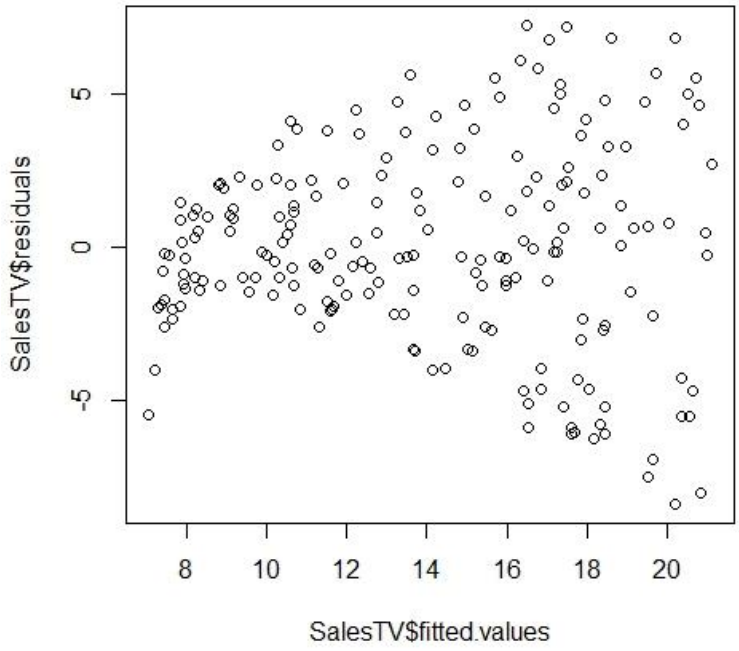
Shapiro-Wilk normality test:

```
>shapiro.test(SalesTV$residuals)
Shapiro-Wilk normality test

data:  SalesTV$residuals
W = 0.9905, p-value = 0.2133
```

The null hypothesis of this test is that the data is normally distributed. Recall that we reject the null hypothesis at 95% level if p-value is less than 0.05. Keep in mind that the residuals represent the variation in "Sales" that can't be explained with "TV". Based on this result "Sales" can be fairly well explained by "TV" in a linear setup.

How do residuals change depending on the predicted value of "Sales" based on "TV"?

```
>plot(SalesTV$fitted.values, SalesTV$residuals)
```



What can we conclude from the graph above?

5.2. Multiple linear regression

In the same way as a simple linear regression multiple regression is used to predict a variable based on other variables, but this time we want to measure the impact of multiple variables in the same regression. The multiple regression takes to the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon,$$

Where each vector X_j represents a different predictor. β_j is the average effect on Y of one unit increase in X_j, holding all other predictors fixed.

The coefficients $\beta_0, \beta_1, \ldots, \beta_p$ are chosen to minimize the sum of squared residuals

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip}\right)^2.$$

As in the case with the simple regression we are interested to know whether the estimated coefficients are statistically significantly different from zero, in other words whether our predictors explain well the behavior of the variable being explained. The null hypothesis takes the following form:

$$H_0: \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

And the alternative hypothesis

$$H_1: \text{at least one } \beta_j \text{ is non-zero.}$$

This hypothesis test is performed by computing the F-statistic:

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)},$$

Where TSS and RSS are defined as previously. When there is no relationship between the response and predictors, the F-statistic is close to 1 and if H_1 holds, then $E\{(TSS - RSS)/p\} > \sigma^2$. Nevertheless, the F-statistic doesn't necessarily have to be much larger than 1 to provide evidence against the null hypothesis. The rule of thumb is that the larger n the smaller F-statistic is enough to provide evidence against the null hypothesis. Luckily R like any other statistical software provides tools to calculate the p-value associated with any given F-statistic.

5.3. Logistic regression

The purpose of the logistic regression is similar to the linear one, but it's intended to be used with qualitative variables. The logistic regression belongs to the classifying models as it helps to predict the set the values of the model will fall to. Some examples of its applications:

- Predicting the probability of default in the banking industry based on the income, transaction history and other related variables
- Predicting the disease of a patient based on her symptoms
- Predicting the risk of a car accident based on the driver's personal factors
- Finding the most suitable marketing approach based on the customer's age, income, and sex

The outcome variables of the prediction models based on the logistic regression are discrete and categorical and hence, there's no uniform distance between them. This means that in most cases the linear regression can't be used to estimate them as it produces continuous values. As in the case of linear regression we want to predict a response variable with explanatory variables whose number can go from one upwards. For the sake of example let's say we want to predict a default risk of a customer based on her credit card balance. Hence, our response variable will have binary values, 0 or 1:

$$Y = \begin{cases} 0 & if\ no\ default \\ 1 & if\ default \end{cases}$$

One way to predict the default risk is estimate the probability of the response variable so that $\Pr(Y = 1|X)$, where X stands for the credit card balance. Using the linear model the probability could be predicted in the conventional way: $p(X) = \Pr(Y = 1|X) = \beta_0 + \beta_1 X$ producing the following result (graph X.X). As it becomes clear from the graph, using the linear regression approach can produce somewhat bizarre results. For instance the probability of default goes from negative to over 1, which is not possible. We could avoid the problem by using the logistic regression with the output values between 0 and 1. The logistic probability is estimated using the following setup:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

which can be easily modified

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0+\beta_1 X}$$

The quantity $p(X)/[1 - p(X)]$ is called the odds, and can take on any value between 0 and ∞. Values of the odds close to 0 and ∞ indicate very low and very high probabilities of default, respectively. For example, on average 1 in 5 people with an odds of 1/4 will default, since p(X) = 0.2 implies an odds of 1−0.2 = 1/4. Likewise on average nine out of every ten people with an odds of 9 will default, since p(X) = 0.9 implies an odds of 1−0.9 = 9.

By taking the logarithms from the previous equation we get the *log-odds* or *logit:*

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X.$$

The coefficients $\beta_0$and β_1 are usually estimated using the maximum likelihood approach:

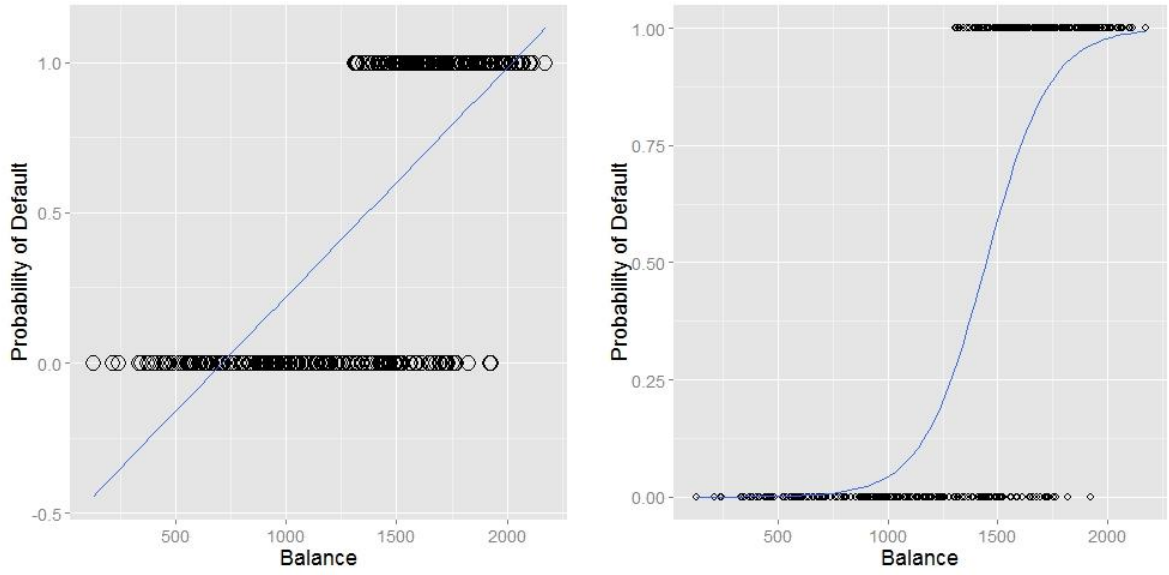
$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} \big(1 - p(x_{i'})\big).$$

Hence, $\hat{\beta}_0$and $\hat{\beta}_1$ are chosen so that the result in equation (x.x) yields a value close to 1 for the ones who defaulted and a value close to 0 for the ones who didn't.

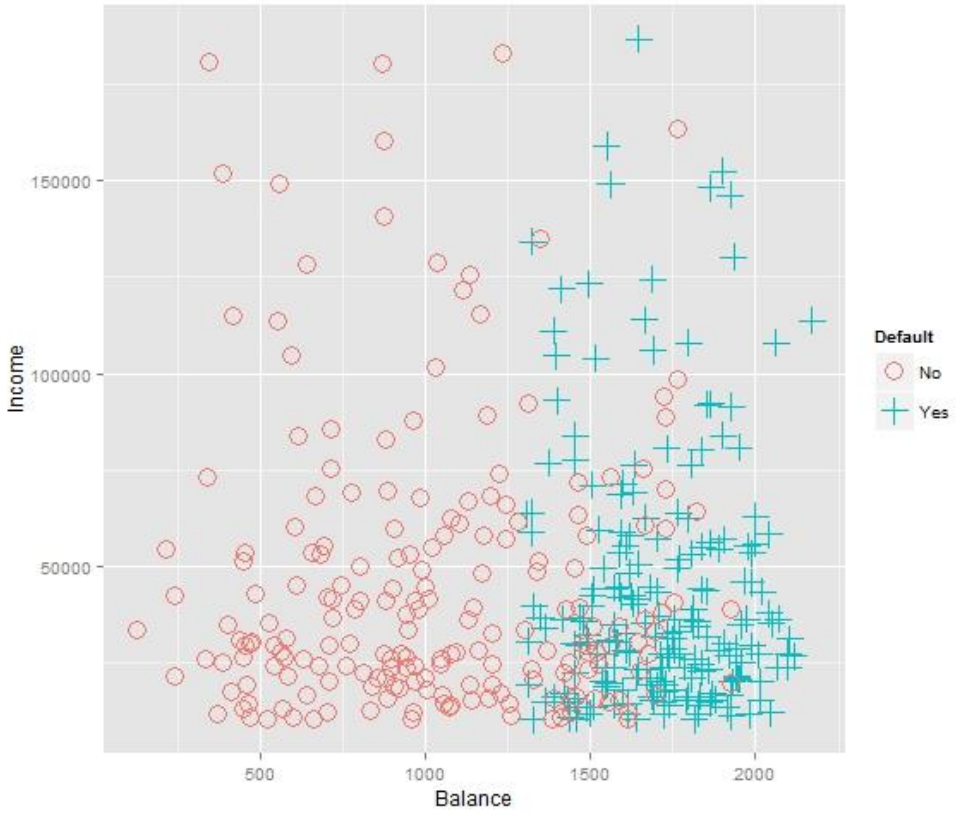
Example 56......

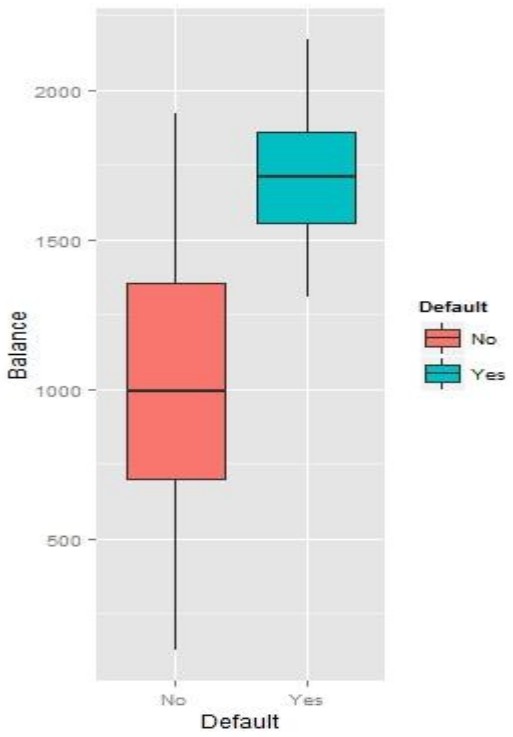
We use a partial data from a dataset "Default" that is included in the package "ISLR".

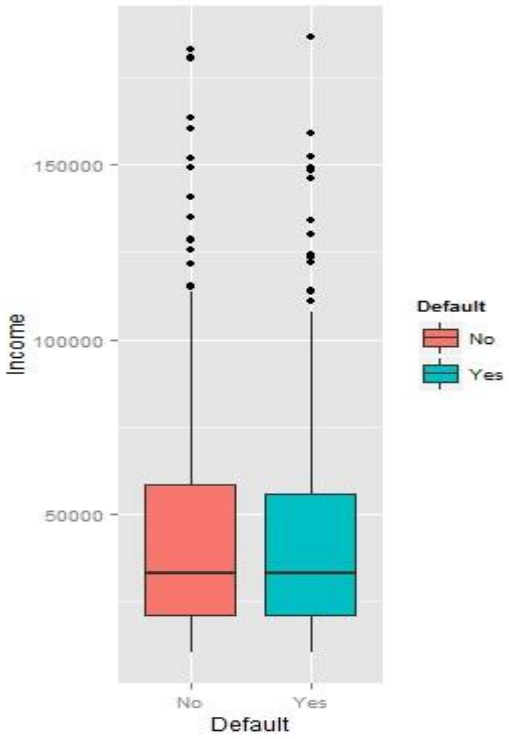
```
> head(Balance2)
   Balance Default Income
1 1861.624     Yes  14891
2 1688.673     Yes 106025
3 1395.560     Yes 104593
4 1560.478     Yes 148924
5 1997.980     Yes  55882
6 1834.381     Yes  80180

Balance3$Default <- ifelse(Balance3$Default=="Yes", 1, 0)

> head(Balance3)
   Balance Default Income
1 1861.624       1  14891
2 1688.673       1 106025
3 1395.560       1 104593
4 1560.478       1 148924
5 1997.980       1  55882
6 1834.381       1  80180


p <- ggplot(Balance3, aes(x=sex, y=Default))
#Logistic
p + geom_point(shape=1, size=3) +
  geom_smooth(method="glm", family="binomial", se=FALSE)+
  theme_grey(base_size = 18) +
  scale_x_continuous(name="Sex") +
  scale_y_continuous(name="Probability of Default")

#Linear
p + geom_point(shape=1, size=3) +
  geom_smooth(method=lm, se=FALSE)+
  theme_grey(base_size = 18) +
  scale_x_continuous(name="Sex") +
  scale_y_continuous(name="Probability of Default")
```



The right-hand side of the plot depicts the probability modeling using the logistic regression. As you can notice the standard linear regression assumes non-sense negative probabilities.







Example 57......

```
>reg=glm(Default~Balance, data=Balance3, family = "binomial")
>summary(reg)
```

The statistics of the regression:

```
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.003e+01  1.092e+00  -9.180   <2e-16 ***
Balance      6.937e-03  7.202e-04   9.631   <2e-16 ***


Coefficients:
(Intercept)      Balance
 -10.028759     0.006937

Degrees of Freedom: 399 Total (i.e. Null);  398 Residual
Null Deviance:        554.5
Residual Deviance: 263.1      AIC: 267.1
```

Using the values of the regression we can estimate for example the probability of default for an individual with the credit balance of 1500:

```
>eq <- exp(reg$coefficients[1] + reg$coefficients[2]*1500)
>eq/(1+eq)
```

$$\hat{p}(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} = \frac{e^{-10.028759+0.006937 \times 1500}}{1+e^{-10.028759+0.006937 \times 1500}} = 0.5930868$$

Thus, an individual with the credit balance of 1500 would default with a probability of around 60%.

Example 58......

How does sex of a client affect the default probability? In other words we would like to model:

$$\Pr(Default = Yes|Sex = Male) \text{ and } \Pr(Default = Yes|Sex = Female)$$

```
>sex <- rbinom(n=400, size=1, prob=0.5)
>Balance3<-cbind(Balance3,sex)
>reg2=glm(Default~sex, data=Balance3, family = "binomial")
>summary(reg2)
```

We get the following estimates from the regression:

```
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.27076  -1.07438   0.00623   1.08684   1.28405

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.2472     0.1474  -1.678   0.0934 .
sex           0.4640     0.2018   2.300   0.0215 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 554.52  on 399  degrees of freedom
Residual deviance: 549.19  on 398  degrees of freedom
AIC: 553.19

Number of Fisher Scoring iterations: 3
```

Inserting the values in the equation yields the following estimates (sex=1 if male, 0 otherwise):

```
eq2 <- exp(reg2$coefficients[1] + reg2$coefficients[2]*1)
eq2/(1+eq2)
eq3 <- exp(reg2$coefficients[1] + reg2$coefficients[2]*0)
eq3/(1+eq3)
```

$$\widehat{Pr}(Default = Yes|Sex = male) = \frac{e^{-0.2472+0.4640 \times 1}}{1 + e^{-0.2472+0.4640 \times 1}} = 0.5539887$$
$$\widehat{Pr}(Default = Yes|Sex = female) = \frac{e^{-0.2472+0.4640 \times 0}}{1 + e^{-0.2472+0.4640 \times 0}} = 0.4385128$$

The probability of a male individual to default is around 55% while for a female individual it is around 44% holding all other variables fixed.

Warning: the default probabilities of male and female individuals are independent and hence, their sum doesn't yield 1. This can be seen by assigning 0.5 for sex:

$$\widehat{Pr}(Default = Yes|Sex = 0.5) = \frac{e^{-0.2472+0.4640 \times 0.5}}{1 + e^{-0.2472+0.4640 \times 0.5}} = 0.4962001$$

Let's consider a case with several explanatory variables. This approach is referred to as multiple logistic regression. Say, we want to explain the probability of default with credit balance, income, and sex. In this case our predictor set will be $X = (X_1, X_2, X_3) = (balance, income, sex)$.

The multinomial model takes the form as follows:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \boldsymbol{X'\beta}$$

where $\boldsymbol{X'\beta} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$and in our case it becomes

$$\Pr(Y = 1|X) = p(X) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\beta_3 X_3}}{1 + e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\beta_3 X_3}} = \frac{exp\{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3\}}{1 + exp\{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3\}}$$

Example 59. .....

The estimation of the same data yields the following results:

```
>reg3=glm(Default~Balance+Income+sex, data=Balance3, family = "binomial")
>summary(reg3)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
```

```
-2.64500  -0.28106   0.05239   0.53101   1.65437

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.025e+01  1.139e+00  -8.999   <2e-16 ***
Balance      6.911e-03  7.220e-04   9.572   <2e-16 ***
Income       1.390e-06  4.612e-06   0.301    0.763
sex          3.749e-01  3.117e-01   1.203    0.229
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 554.52  on 399  degrees of freedom
Residual deviance: 261.51  on 396  degrees of freedom
AIC: 269.51

Number of Fisher Scoring iterations: 6
```

What can we conclude from these estimates above? When we include balance, income, and sex as the explanatory variables only balance remains statistically significant, while income and sex are both too weak to explain the default probability. Hence, even though sex is a strong explanatory variable by itself, it is not as strong together with balance and income variables. But it doesn't mean that it's needless!

Applying the coefficient values we could estimate the default probability for a male individual with income of 50000 and a credit balance of 1000:

```
eq <- exp(reg3$coefficients[1] + reg3$coefficients[2]*1000 +
reg3$coefficients[3]*50000 + reg3$coefficients[4]*1 )
eq/(1+eq)
```

$$\widehat{Pr}(Y = 1|X) = \hat{p}(X) = \frac{e^{-10.25 + 0.0069 \times 1000 + 0.000001390 \times 50000 + 0.3749 \times 1}}{1 + e^{-10.25 + 0.0069 \times 1000 + 0.000001390 \times 50000 + 0.3749 \times 1}} = 0.0519$$

and a female individual with the same balance and income:

```
eq <- exp(reg3$coefficients[1] + reg3$coefficients[2]*1000 +
reg3$coefficients[3]*50000 + reg3$coefficients[4]*0 )
eq/(1+eq)
```

$$\widehat{Pr}(Y = 1|X) = \hat{p}(X) = \frac{e^{-10.25 + 0.0069 \times 1000 + 0.000001390 \times 50000 + 0.3749 \times 0}}{1 + e^{-10.25 + 0.0069 \times 1000 + 0.000001390 \times 50000 + 0.3749 \times 0}} = 0.0362$$

Hence, a male individual with income of 50000 and a credit balance of 1000 will default with a probability of 5.19% and a female individual with the same balance and income with a probability of 3.62%.

5.4. Confounding variables (incomplete)

5.5. Multinomial logistic regression

Logistic regression is a nice tool to predict discrete outcomes that are binary. But what if we had a dependent or explanatory variable that would have more than just two possible discrete values? Say, we want to predict whether an individual would prefer to buy a BMW, Mercedes, Ford or Toyota based on her income. And how do choices change when income changes? How should we deal with this problem?

Let's look at an example of the heating system choice related to the number of rooms in the houses.

The households have five options of heating systems: gas central, gas room, electric central, electric room, and heat pump. The number of rooms has three classes: 2-3 rooms, 4-5 rooms and 6-7 rooms. This data is taken from the package "mlogit". The cross-table of options looks as follows:

Rooms	Heating choice					Total
	gc	gr	ec	er	hp	
2-3	208	48	19	32	19	326
4-5	182	39	25	20	15	281
6-7	183	42	20	32	16	293
Total	573	129	64	84	50	900

Here we can see how the number of rooms is related to the heating choice of the house. Obviously our response variable is the heating system choice and we want to know whether it can be explained with the number of rooms in the house. The original data includes observations for 900 households that can be indexed from 1 to 900, namely $i = 1, \dots, 900$. Next, we have 5 different heating system options, that can be indexed from 1 to 5 (gc=1, gr=2, ec=3, er=4, hp=5), namely $j = 1, \dots, 5$. The probability that a household i would choose the option j as a heating system can be denoted as $\pi_{ij} = Pr\{Y_i = j\}$. But since we have grouped the observations we only need to know which group a particular observation falls in, and instead we could index the groups from 1 to 3: $i = 1,2,3$ (1=2-3 rooms, 2=4-5 rooms, 3=6-7 rooms). Say, a household has a house with 4 rooms, what is the probability that it chooses electric central as a heating system (assuming that the number of rooms is the only factor affecting the choice of the heating system)? The answer is $25/281 \approx 0.089$, or about 8.9%. The probabilities of heating system choices for a certain household always add up to 1, hence $\sum_{j=1}^{J} \pi_{ij} = 1$ and consequently we need to have $J - 1$ parameters to have all the information needed. Next, we denote $\#_i$ the number of

observations in a particular group, here we have for example $\#_3 = 293$ and consequently $y_{i1}, \dots, y_{i5}$ represent the numbers of households per each heating system in the group i, which, of course, sum up as $\sum_j y_{ij} = \#_i$. And in general, y_{ij} represents the number of observations in a particular group for a particular heating system. Since a single household can obviously choose only one heating system, in this case $n_i = 1$ and Y_{ij} can be further interpreted as a dummy variable. Combining the information that we assigned to the variables we can represent it in the multinomial distribution form (probability mass function), namely

$$Pr\{Y_{i1} = y_{i1}, \dots, Y_{iJ} = y_{iJ}\} = \binom{n_i}{y_{i1}, \dots, y_{iJ}} \pi_{i1}^{y_{i1}} \cdots \pi_{iJ}^{y_{iJ}}.$$

In the multinomial logistic model we assume that the log-odds be represented in the linear model, as in the logistic regression model:

$$\log \frac{\pi_{ij}}{\pi_{iJ}} = \beta_0 + \boldsymbol{x}_i' \boldsymbol{\beta}_j,$$

where $j = 1, \dots, J-1$. By exponentiating the equation above, noting that $\log \frac{\pi_{iJ}}{\pi_{iJ}} = 0$ for all J and that $\sum_j \pi_{ij} = 1$ we obtain $\pi_{iJ} = 1/\sum_j \exp\{\log \pi_{ij}/\pi_{iJ}\}$ which leads in the equation below

$$\pi_{ij} = \frac{\exp\{\beta_0 + \boldsymbol{x}_i' \boldsymbol{\beta}_j\}}{\sum_{k=1}^{J} \exp\{\beta_0 + \boldsymbol{x}_i' \boldsymbol{\beta}_k\}}.$$

The log-odds of the multinomial logistic model can be also derived through the utility representation of the individuals' choices. Let U_{ij} denote the utility of the j-th choice for the i-th individual. Now let's split U_{ij} in the systematic component η_{ij} and a random component ϵ_{ij} (error terms) so that

$$U_{ij} = \eta_{ij} + \epsilon_{ij}.$$

If we further assume that individuals maximize their utility by choosing the largest of $U_{i1}, \dots, U_{iJ}$, then the probability that an individual i will choose alternative j to maximize her utility is

$$\pi_{ij} = \Pr\{Y_i = j\} = \Pr\{\max(U_{i1}, \dots, U_{iJ}) = U_{ij}\}.$$

Now, because error terms ϵ_{ij} have extreme value distributions with density

$$f(\epsilon) = \exp\{-\epsilon - \exp\{-\epsilon\}\}$$

then (Maddala, 1983, pp 60-61)

$$\pi_{ij} = \frac{\exp\{\eta_{ij}\}}{\sum_{k=1}^{J} \exp\{\eta_{ik}\}}.$$

Another useful representation of the model above is the conditional form (conditional logistic model) or in terms of alternatives rather than attributes of the individuals.
Let $\boldsymbol{w}_j$ represent a vector of j-th alternative, then the utility of the individual can be represented as

$$\eta_{ij} = \boldsymbol{w}_j'\boldsymbol{\gamma}$$

and by combining it with a multinomial logistic model we can obtain a general or mixed multinomial logistic model

$$\eta_{ij} = \boldsymbol{x}_i'\boldsymbol{\beta}_j + \boldsymbol{w}_{ij}'\boldsymbol{\gamma}$$

where the vector of independent variables $\boldsymbol{x}_i'$ is called alternative-invariant and $\boldsymbol{w}_{ij}'$ is called alternative-variant.

The theory of the multinomial logistic model can be rather tricky and it's not always clear what the conditional multinomial logistic model is meant for. Nevertheless, with the help of the following examples you should be able to understand the basic principles of the multinomial logistic model.

Example 60. .....

The data being used here is taken from the package "mlogit". It is called "Mode Choice for the Montreal-Toronto Corridor", where individuals choose the travelling mode between Montreal and Toronto in Canada. Most of them have four options: airplane, bus, car, and train, while some of them have fewer options. We will subset the data to include only the ones who have all four options available.

```
> library(mlogit) #this is the recommended package (nnet is another one, but
with different functions)
> data("ModeCanada") #the data set
```

```
> View(ModeCanada)
> table(ModeCanada$noalt)

    2     3     4
  462  3942 11116


> ModeCanada <- subset(ModeCanada, ModeCanada$noalt>3)
> table(ModeCanada$noalt)

    4
11116
>
> head(ModeCanada)
    case   alt choice dist   cost ivt ovt freq income urban noalt nchoice
304  109 train      0  377  58.25 215  74    4     45     0     4       4
305  109   air      1  377 142.80  56  85    9     45     0     4       4
306  109   bus      0  377  27.52 301  63    8     45     0     4       4
307  109   car      0  377  71.63 262   0    0     45     0     4       4
308  110 train      0  377  58.25 215  74    4     70     0     4       4
309  110   air      1  377 142.80  56  85    9     70     0     4       4
```

This form of data is called 'long'. Here the rows are organized according to the choices of the individuals. The actual chosen mode of transport is assigned 1 for each individual and the rest are 0. Hence, there are four rows of data for each individual. As you can see some of the rows repeat, such as income, because it doesn't change for an individual no matter which mode s/he chooses. The data could include other variables of choice as well and they would require the same type of dummy column of choice as above. Another form of data that could be used is called 'wide'. We will see later how one can transform from one to another and back, but it's really important to keep in mind that the form of data really affects the outcome. First, let's see how we can operate with the data in the form 'long'.

Warning: the coefficients of the pure, conditional or mixed multinomial logistic model are difficult to interpret. Neither the sign, nor the magnitude of the coefficients has an intuitive meaning. Nevertheless, careful analysis should always help.

Example 61......

Say, we are interested to know how distance (dist), income, and cost affect the choice of the transport mode.

```
> table(ModeCanada$choice,ModeCanada$alt)

    train  air  bus  car
  0  2316 1740 2769 1512
  1   463 1039   10 1267
```

As we see 'bus' was chosen really seldom while 'car' and 'air' are the most frequent choices.

```
> ModeCanada <- mlogit.data(ModeCanada, alt.var="alt", choice = "alt", shape
= "long")[,c(2,3,4,5,9)]
>
> ModeCanada[1:12,]
          alt choice dist   cost income
1.train train      0  377  58.25     45
1.air     air      1  377 142.80     45
1.bus     bus      0  377  27.52     45
1.car     car      0  377  71.63     45
2.train train      0  377  58.25     70
2.air     air      1  377 142.80     70
2.bus     bus      0  377  27.52     70
2.car     car      0  377  71.63     70
3.train train      0  377  58.25     35
3.air     air      1  377 142.80     35
3.bus     bus      0  377  27.52     35
3.car     car      0  377  71.63     35
>
> mlogit.model1 <- mlogit(choice ~ 1 | cost, data=ModeCanada, shape="long",
alt.var="alt", reflevel = "car")
>
> summary(mlogit.model1)

Call:
mlogit(formula = choice ~ 1 | cost, data = ModeCanada, reflevel = "car",
    shape = "long", alt.var = "alt", method = "nr", print.level = 0)

Frequencies of alternatives:
      car     train       air       bus
0.4559194 0.1666067 0.3738755 0.0035984

nr method
8 iterations, 0h:0m:0s
g'(-H)^-1g = 7.19E-07
gradient close to zero

Coefficients :
                    Estimate  Std. Error  t-value  Pr(>|t|)
train:(intercept)  -1.6019197   0.2067141  -7.7494 9.326e-15 ***
air:(intercept)   -10.5412593   0.4736322 -22.2562 < 2.2e-16 ***
bus:(intercept)    -4.1906113   1.1378782  -3.6828 0.0002307 ***
train:cost          0.0115781   0.0038360   3.0183 0.0025421 **
air:cost            0.0668507   0.0030441  21.9606 < 2.2e-16 ***
bus:cost           -0.0273963   0.0472261  -0.5801 0.5618412
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Log-Likelihood: -2533.4
McFadden R^2:  0.12743
Likelihood ratio test : chisq = 739.96 (p.value = < 2.22e-16)
>
```

This is the pure multinomial model. Our dependent variable is 'choice' and we fix at 1, since it is not varying with alternatives. We want to see how cost affects the choice of the transport mode. Next, we choose "car" as our mode of transport of reference. This means that we observe how other modes of transport change compared to "car" when the cost is changing. For example growing the cost of driving a car by one unit will grow the log-odds of taking a train by approximately 1.16% compared to driving a car. The coefficient of train is significant at 95% level. Growing the cost of driving a car by one unit results in increasing the log-odds of taking the airplane by approximately 6.69% and its coefficient is significant at 99% level. Nevertheless the coefficient of bus is not significant (the modulus of the standard error exceeds the size of the coefficient).

```
> exp(coef(mlogit.model1))
train:(intercept)   air:(intercept)   bus:(intercept)
     2.015093e-01      2.642343e-05      1.513703e-02
       train:cost          air:cost          bus:cost
         1.011645          1.069136         0.9729756
attr(,"fixed")
train:(intercept)   air:(intercept)   bus:(intercept)
            FALSE             FALSE             FALSE
       train:cost          air:cost          bus:cost
            FALSE             FALSE             FALSE
```

In the table above the same observation is obvious the value of x:cost tells the multiplier of the probability of taking the transport after increasing the cost of "car" by one unit.

Example 62......

Now, let's use another data set to demonstrate the conditional multinomial logistic model.

```
data("Car", package="mlogit") #the data set
View(Car)
Car <- mlogit.data(Car, alt.levels = 1:6, varying = 5:70, choice = "choice",
shape = "wide", sep = "")
#alt.levels: possible options, 6 for each person
#varying: nominal parameters
```

Remember, in the conditional model we estimate the coefficients that are alternative-invariant, so we don't have to choose a reference choice for the regression. For the sake of example let's regress the variable of choice on all available nominal variables in the data set.

```
# Conditional model
> mlogit.model2 <- mlogit(choice ~
price+range+acc+speed+pollution+size+space+cost+station, data=Car,
shape="long", alt.var="alt")
> summary(mlogit.model2)

Call:
```

```
mlogit(formula = choice ~ price + range + acc + speed + pollution +
    size + space + cost + station, data = Car, shape = "long",
    alt.var = "alt", method = "nr", print.level = 0)

Frequencies of alternatives:
       1        2        3        4        5        6
0.190589 0.057800 0.288999 0.074989 0.322089 0.065535

nr method
5 iterations, 0h:0m:1s
g'(-H)^-1g = 4.91E-06
successive function values within tolerance limits

Coefficients :
                 Estimate  Std. Error  t-value  Pr(>|t|)
2:(intercept) -1.19313360  0.06960508 -17.1415 < 2.2e-16 ***
3:(intercept) -0.01605586  0.06236401  -0.2575 0.7968283
4:(intercept) -1.36513323  0.07753132 -17.6075 < 2.2e-16 ***
5:(intercept) -0.32556327  0.09205583  -3.5366 0.0004053 ***
6:(intercept) -1.91780499  0.10528938 -18.2146 < 2.2e-16 ***
price         -0.18712971  0.02699695  -6.9315 4.164e-12 ***
range          0.00401531  0.00025783  15.5737 < 2.2e-16 ***
acc           -0.07281306  0.01095333  -6.6476 2.980e-11 ***
speed          0.00340031  0.00076759   4.4298 9.431e-06 ***
pollution     -0.18147772  0.09521123  -1.9061 0.0566432 .
size           0.07203239  0.02916950   2.4694 0.0135324 *
space          0.91481193  0.17538722   5.2160 1.829e-07 ***
cost          -0.07233509  0.00744226  -9.7195 < 2.2e-16 ***
station        0.25023356  0.07047056   3.5509 0.0003839 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Log-Likelihood: -7080.8
McFadden R^2:  0.035344
Likelihood ratio test : chisq = 518.87 (p.value = < 2.22e-16)
```

How to interpret the coefficients of the conditional model? If we added for example the “choice 3” as a reference level in the previous setup we’d notice that the coefficients of the independent variables are not changing. Why so? In fact the coefficients of the conditional model can be interpreted to explain whether these factors are affecting the choice of the car in the first hand. For example the coefficient of the price (price of a vehicle divided by the logarithm of income) is negative and significant at 99% level. This tells us that increasing the price should reduce the willingness to buy a car, which is quite a realistic result. The same logic applies to the coefficient of cost (cost per mile of travel (tens of cents)). But on the other hand since the coefficient of the space variable is positive and statistically significant, increasing the space of the car (fraction of luggage space in comparable new gas vehicle) is likely to increase its attractiveness.

Example 63......

Next, let's proceed with the mixed multinomial model.

```
# Mixed model
> mlogit.model3 <- mlogit(choice ~ cost+speed | price, data=Car,
shape="long", alt.var="alt", reflevel = "3")
> summary(mlogit.model3)

Call:
mlogit(formula = choice ~ cost + speed | price, data = Car, reflevel = "3",
    shape = "long", alt.var = "alt", method = "nr", print.level = 0)

Frequencies of alternatives:
       3        1        2        4        5        6
0.288999 0.190589 0.057800 0.074989 0.322089 0.065535

nr method
5 iterations, 0h:0m:1s
g'(-H)^-1g = 0.000266
successive function values within tolerance limits

Coefficients :
                Estimate Std. Error t-value  Pr(>|t|)
1:(intercept) -0.1865314  0.1051452 -1.7740  0.076057 .
2:(intercept) -0.8616399  0.1633765 -5.2740 1.335e-07 ***
4:(intercept) -0.9934246  0.1418352 -7.0041 2.486e-12 ***
5:(intercept)  0.0370236  0.0891179  0.4154  0.677816
6:(intercept) -1.1843830  0.1536278 -7.7094 1.266e-14 ***
cost          -0.0713879  0.0072506 -9.8458 < 2.2e-16 ***
speed          0.0035951  0.0007413  4.8497 1.237e-06 ***
1:price       -0.0488609  0.0221505 -2.2059  0.027394 *
2:price       -0.1783347  0.0382607 -4.6610 3.146e-06 ***
4:price       -0.0870310  0.0322393 -2.6995  0.006944 **
5:price        0.0171088  0.0189882  0.9010  0.367577
6:price       -0.0726519  0.0348173 -2.0867  0.036919 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -7260.9
McFadden R^2:  0.010807
Likelihood ratio test : chisq = 158.65 (p.value = < 2.22e-16)
```

The coefficients of the mixed multinomial model are alternative-variant and alternative-invariant. In this setup the coefficients of cost and speed are alternative-invariant and the coefficients of the price vector are with the reference to the choice 3. If you compare this regression to the conditional one, you should

notice that the coefficients of cost and speed are pretty much similar. The coefficients of price should be interpreted in the similar manner as with the 'ModeCanada' data set. For example increasing the price

Example 64......

In order to measure how much the change in the coefficients affects the probability of making a specific choice we need calculate the marginal effects. The marginal effects for the conditional model can be calculated as follows. As you recall we created the conditional model with the following function:

```
mlogit.model2 <- mlogit(choice ~
price+range+acc+speed+pollution+size+space+cost+station, data=Car,
shape="long", alt.var="alt", reflevel = "4")
```

First, we need to calculate the means of the independent variables for each choice. This can be done:

```
z <-with(Car, data.frame(price=tapply(price, index(mlogit.model2)$alt, mean),
                 range=tapply(range, index(mlogit.model2)$alt, mean),
                 acc=tapply(acc, index(mlogit.model2)$alt, mean),
                 speed=tapply(speed, index(mlogit.model2)$alt, mean),
                 pollution=tapply(pollution, index(mlogit.model2)$alt, mean),
                 size=tapply(size, index(mlogit.model2)$alt, mean),
                 space=tapply(space, index(mlogit.model2)$alt, mean),
                 cost=tapply(cost, index(mlogit.model2)$alt, mean),
                 station=tapply(station, index(mlogit.model2)$alt, mean)))
```

```
> effects(mlogit.model2, covariate = "cost", data = z)
             4            1             2
4 -0.004972598  0.001006346  0.0003051940
1  0.001006346 -0.011013015  0.0007699902
2  0.000305194  0.000769990 -0.0038763863
3  0.001537217  0.003878325  0.0011761776
5  0.001764767  0.004452424  0.0013502842
6  0.000359075  0.000905930  0.0002747409
             3            5             6
4  0.001537216  0.001764767  0.0003590753
1  0.003878325  0.004452424  0.0009059301
2  0.001176177  0.001350284  0.0002747409
3 -0.014776727  0.006801179  0.0013838289
5  0.006801179 -0.015957328  0.0015886736
6  0.001383829  0.001588673 -0.0045122481
```

The values in the table above are the covariance values of the choices. The first important observation is that the diagonal values are negative and the rest of the values are positive. Let's take the cell 1,1 as an example. Remember, our independent variable is cost. Hence, the negative coefficient in the cell 1,1 means that if the cost of the choice 4 increases then the choice 4 becomes less wanted. The values of

the covariates can be interpreted simply as percentage changes, since the marginal effects are calculated as

$$\frac{\partial p_{ij}}{\partial \boldsymbol{w}_i} = p_{ij}\left(\gamma_j - \bar{\gamma}_i\right)$$

We can also calculate the marginal effects of the mixed model with respect to the variable that has a reference choice. Recall the mixed multinomial model:

```
mlogit.model3 <- mlogit(choice ~ cost+speed | price, data=Car, shape="long",
alt.var="alt", reflevel = "1")
```

Now marginal effects w.r.t. price:

```
> effects(mlogit.model3, covariate = "price", data = z)
           1            2            3            4
 0.008701082 -0.008388572 -0.006544084 -0.004722780
           5            6
 0.014035903 -0.003081549
```

Surprisingly, increasing the price of the choice 1 is also increasing its attractiveness (irrationality or a Giffen good?), as well as for the choice 5. For the choices 2, 3, 4 and 6 increasing the price decreases their attractiveness.

Finally, about an assumption called “Independence of irrelevant alternatives” (IIA). By the definition of the multinomial logistic regression it is assumed that introducing new choices should not have any effect on the willingness to select older choices. This is rather a restrictive assumption and therefore it is not believed to hold in practice. Nevertheless, it is possible to control for it to some degree by running the Hausman-McFadden test. The basic idea is to subset a collection of choices by dismissing a choice on purpose.

Example 65......

Say, we want test whether IIA is holding for the mixed logistic model. We first subset a new set of choices by dropping the choice 1 and choice 2 and then perform a test.

```
m.1 <- mlogit(choice ~ cost+speed | price, data=Car, shape="long",
alt.var="alt", reflevel = "4")
m.2 <- mlogit(choice ~ cost+speed | price, data=Car, alt.subset =
c("3","4","5","6"), reflevel = "4")
# Hausman-McFadden test
hmftest(m.1,m.2)

Hausman-McFadden test
```

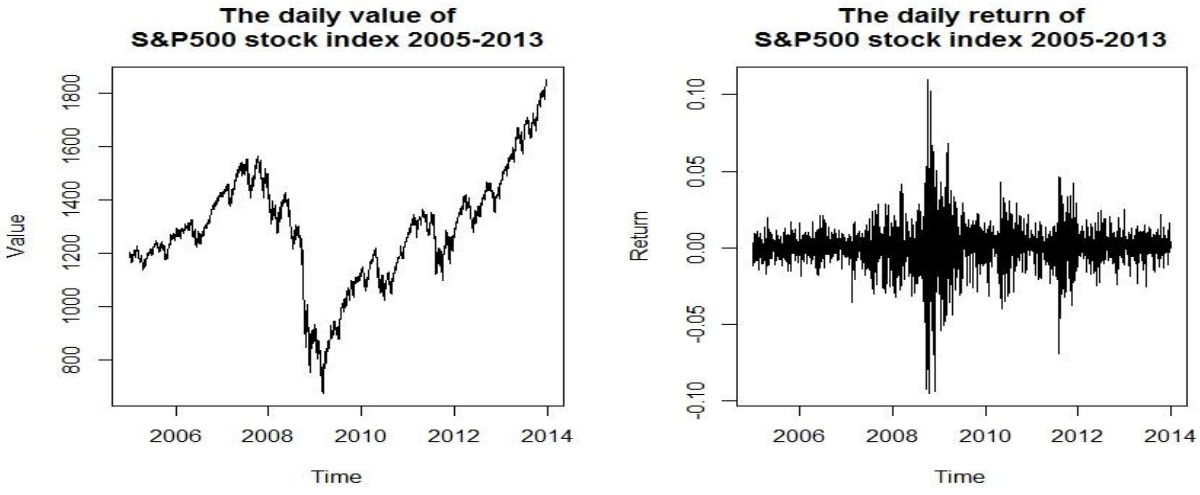
```
data:  Car
chisq = 13.085, df = 8, p-value = 0.109
alternative hypothesis: IIA is rejected
```

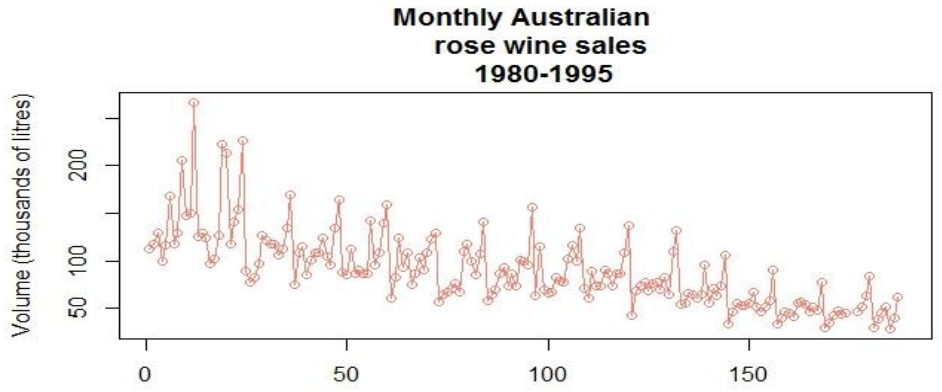
Recall, small p-value is an argument against the null hypothesis; hence we can't reject the null hypothesis, which is "IIA holds". Changing the composition of choices may change the result of the test significantly. Try it!

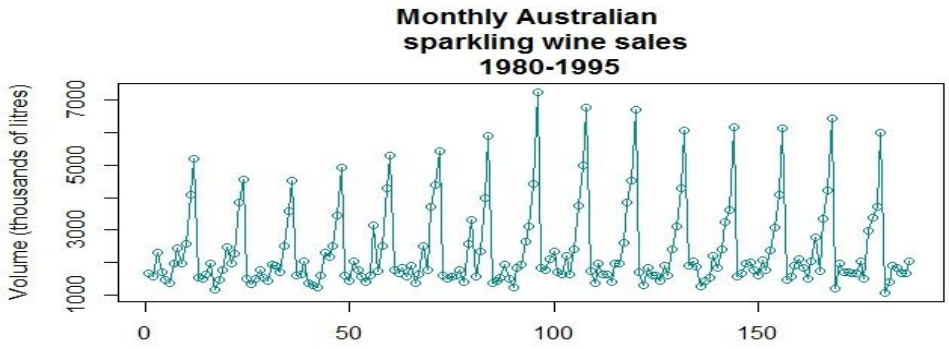
5.6. Time series

Time series technics are widely applied in macro and microeconomics. It represents a collection of data in which the observations correspond to consecutive time periods that can vary in length and frequency. Time series can be forecasted and related to other data using their own or other time series historical data. Despite several modern branches and technics, the most applied technique in time series analysis nowadays is still the linear regression. Time series analysis has been developing rapidly since the late 70's due to applications in macroeconomics and finance; later on it was adapted in other business industries as well. Time series analysis is famous for models trying to catch the dynamics of the error terms and volatility (linearly and non-linearly) of the variables. Nevertheless, it still relies on the basic concepts of statistics, such as hypothesis testing, regression analysis, maximum likelihood estimators, and correlation and variance technics. Other than business or economics fields such as meteorology and signal processing have also largely benefited from the development of the time series analysis. In this paragraph you will be introduced to the most important and profound time series technics that should provide you the basic grip in this field of statistics.

As in the case of data analytics in general, inspecting the graphical representation of a time series is the best way to gain an intuition of what could be the starting point for the modeling. Some popular time series graphed:







One of the most important observations in the time series analysis could be to check whether time series values are fluctuating around a certain mean, like zero. For example the daily returns of the S&P500 index have a clear mean around zero, while some of the time series presented have a trend or a seasonal variation. Most of the times it is necessary to untrend the time series or remove a cyclical part from it in order to model it or to analyze it in comparison to other time series.

Perhaps the most common techniques for untrending is taking a difference or a log-difference as follows:

$$\Delta y_t = y_t - y_{t-1}$$

OR

$$\Delta \log y_t = \log(y_t/y_{t-1}) = \log y_t - \log y_{t-1} \approx (y_t - y_{t-1})/y_{t-1}$$

where y_t is a common notation for time series data at the time period $t = 1, \dots, T$, where T is the total number of observations. y_{t-1} is usually called the first lag of the time series, y_{t-2} is the second lag and so on. y_{t+1} is the first lead, y_{t+2} is the second lead and so on.

In case of a linear trend it's possible to fit a trend line using least squares regression and to consider modeling the resulting residual series. Then, the observed time series y_t would be replaced by the time series

$$(\hat{\varepsilon}_t = y_t - \hat{\beta}_0 - \hat{\beta}_1 t, \qquad t = 1, \dots, T),$$

where the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are obtained by regressing y_t on a constant and t.

If the time series seem to have a cyclical behavior, then most of the times it's not a bad idea to identify a seasonal (repetitive) pattern to produce a relevant analysis. A possible solution could be to use trigonometric functions of time, but unfortunately especially in the case of economic and business time series the seasonality is not uniform across the data and therefore trigonometric functions demand a lot of parametric adjustment. Hence, an often used alternative is to use seasonal indicators. For example in case of quarter-annual data one could use the residual series:

$$\hat{\varepsilon}_t = y_t - \hat{\beta}_1 d_{1t} - \cdots - \hat{\beta}_4 d_{4t}, \qquad t = 1, \dots, T,$$

where ith seasonal indicator d_{it} takes value one when the observation in question corresponds to quarter i and zero otherwise ($i = 1, \dots, 4$).

Very often, the aim of time series analysis is to build a statistical model that represents the observed time series and its fluctuations and the dependence structure of consecutive observations. This dependence is often quite apparent, either by inspecting a graph of the series, or because of an underlying theory explaining how the phenomenon generating the time series behaves. After a statistical model has been chosen, one can estimate the unknown parameters of the model, check whether the estimated model and the observations are compatible, test hypotheses concerning the model parameters, and finally use the model for the intended purpose.

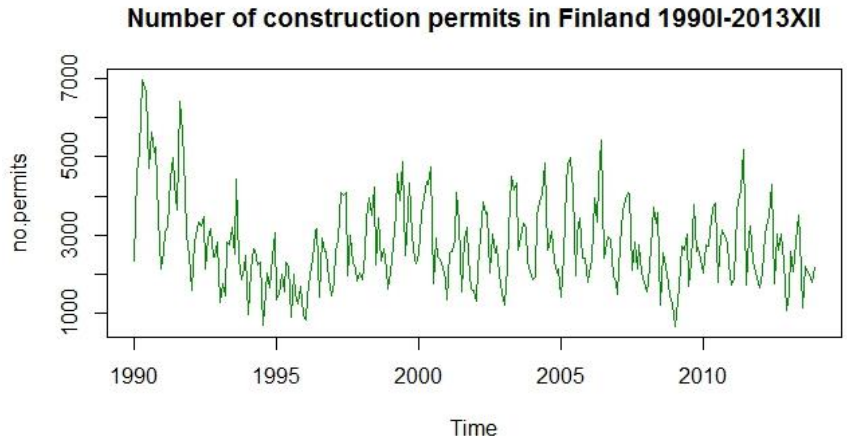
One purpose could be to simply provide a summarized description of the observed data set, which may be useful in understanding the underlying data generation process producing the data. Another central use of a time series model is to forecast the future values of the time series (or several time series). On the other hand, a series of interest may be forecasted making use of information contained in other explanatory time series. In case of multiple time series, often exploring potential temporal and contemporaneous dependencies between the time series is another issue of interest.

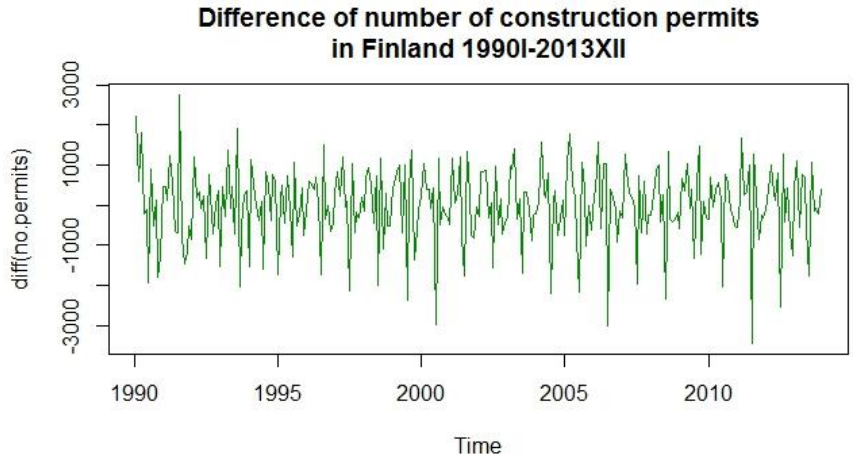
5.6.1. General techniques

Example 66. .....

We've got a construction permits monthly data from Finland spanning 1990-2013:

```
# Data: construction permits in Finland 1990M01-2013M12
no.permits <- ts(constr$number, start = c(1990,1), frequency=12)
# Plotting the original series
plot(no.permits, col="green4", main="Number of construction permits in
Finland 1990I-2013XII")
# Plotting the differences of the original series
plot(diff(no.permits), col="green4", main="Difference of number of
construction permits
     in Finland 1990I-2013XII")
> mean(no.permits)
[1] 2784.997
> mean(diff(no.permits))
[1] -0.630662
```





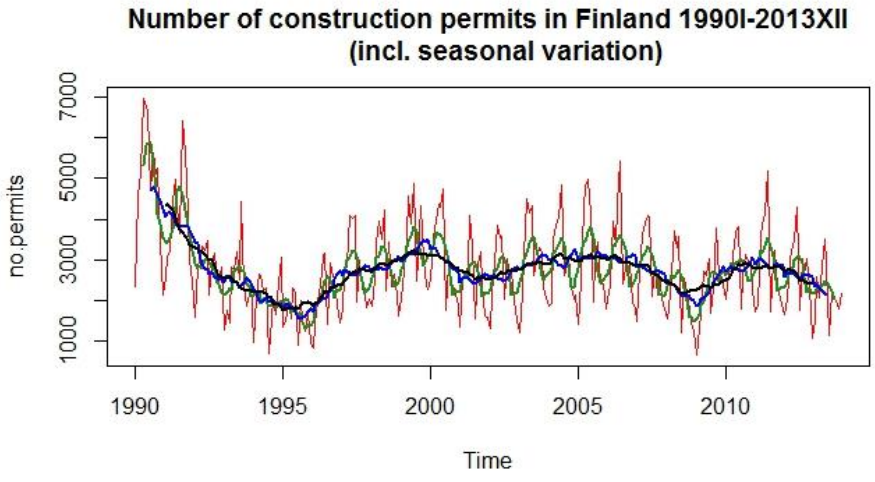
Example 67......

The original series as well the differences of it seems to have a seasonal behavior. Let's check the means of each month first to find if there are differences between them:

```
> colMeans(permits)
     Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1629.750 2397.042 3042.750 3656.375 3796.458 3988.917 2048.042 3088.000
     Sep      Oct      Nov      Dec
2980.292 2621.208 2215.625 1955.500
```

June seems to have the most permits on average and then there's another spike in August. Otherwise the series gradually declines to wards January which has the lowest number of permits. Another bottom is in July. In fact we can identify to cycles here, since the series has its lowest values every half a year on average. We could fit the cyclical part repeating every half a year and every month by using a hands-on algorithm:

```
plot(no.permits, col="red", main="Number of construction permits in Finland
1990I-2013XII
     (incl. seasonal variation)")

# Quartely a=3
permits.cycle.3 <- filter(no.permits,filter=rep(1/7,7))
lines(permits.cycle.3,col="forestgreen", lwd=2)

# Every half a year a=6
permits.cycle.6 <- filter(no.permits,filter=rep(1/13,13))
lines(permits.cycle.6,col="blue", lwd=2)

# Every year a=12
permits.cycle.12 <- filter(no.permits,filter=rep(1/25,25))
lines(permits.cycle.12,col="black", lwd=2)
```



How to interpret the coefficients of *filter()*?

In fact this algorithm represents a decomposition with a simple class of linear filters which are moving averages with equal weights:

$$T_t - \sum_{i=-\infty}^{\infty} \lambda_i y_{t+i} = T_t - \frac{1}{2a+1} \sum_{i=-a}^{a} y_{t+i}$$

In this case, the filtered value of a time series at a given period t is represented by the average of the values $\{y_{t-a}, \dots, y_t, \dots, y_{t+a}\}$ and the coefficients of the filtering are $\left\{\frac{1}{2a+1}, \dots, \frac{1}{2a+1}\right\}$.

For instance,

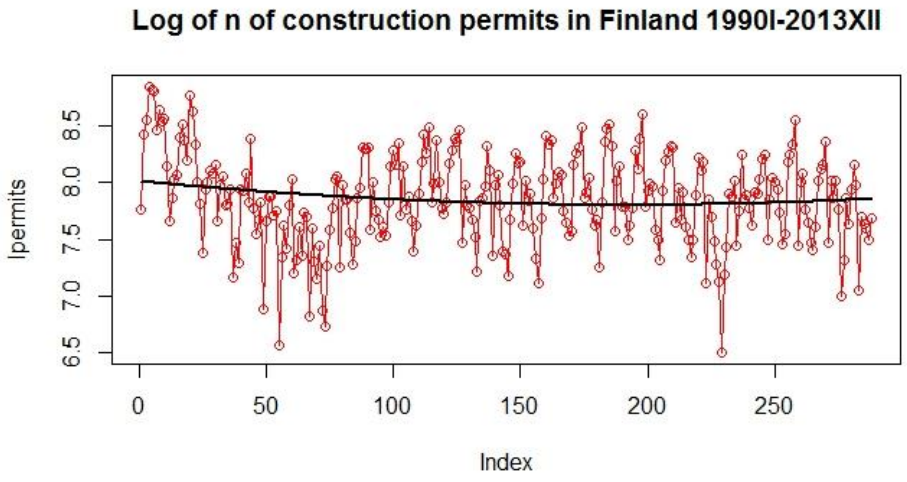
if $a = 3$, then $\lambda_7 = \left\{\underbrace{\frac{1}{2a+1}, \dots, \frac{1}{2a+1}}_{7\ times}\right\} = \left\{\frac{1}{7}, \dots, \frac{1}{7}\right\}$

if $a = 12$, then $\lambda_{12} = \left\{\underbrace{\frac{1}{2a+1}, \dots, \frac{1}{2a+1}}_{25\ times}\right\} = \left\{\frac{1}{25}, \dots, \frac{1}{25}\right\}$

Example 68. .....

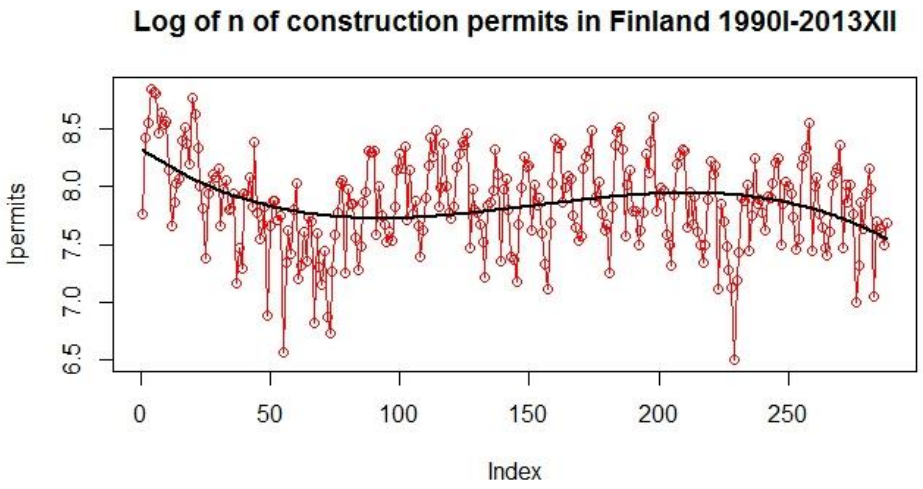
Another beautiful way to fit a trend line is to use a quadratic function of time:

```
# Quadratic function of time fitted to the series
lpermits<-log(constr$number)
t<-seq(from=1,to=length(constr$number),by=1)
t2<-t^2
plot(lpermits,type="o", pch=1, lty=1, col=2, main="Log of n of construction
permits in Finland 1990I-2013XII")
lines(lm(lpermits~t+t2)$fit,col="black",lwd=2)
```



If it seems that a quadratic equation $\log y_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \varepsilon_t$ doesn't describe the data well enough, then you can always try higher degree polynomials, such as a cubic one $\log y_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \varepsilon_t$:

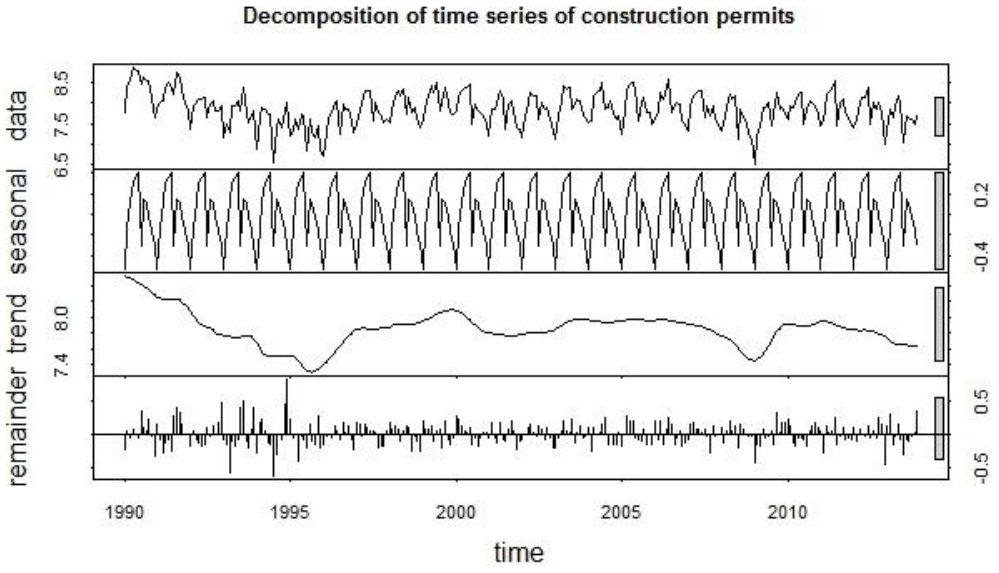
```
# Cubic function of time fitted to the series
lpermits<-log(constr$number)
t<-seq(from=1,to=length(constr$number),by=1)
t2<-t^2
t3<-t^3
plot(lpermits,type="o", pch=1, lty=1, col=2, main="Log of n of construction
permits in Finland 1990I-2013XII")
lines(lm(lpermits~t+t2+t3)$fit,col="black",lwd=2)
```



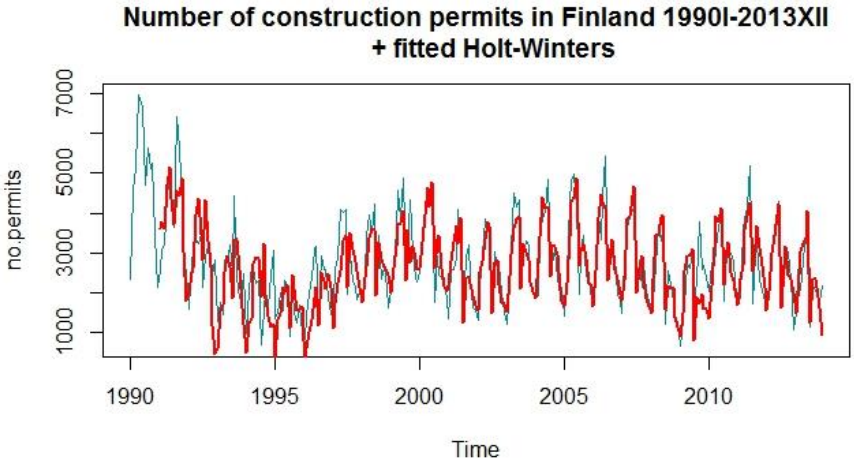
Example 69......

The decomposition of the time series into cyclical, trending and main data components can be easily done by using for example a function *stl()*:

```
permits.stl <- stl(log(no.permits),s.window="periodic")
plot(permits.stl, main="Decomposition of time series of construction
permits")
```

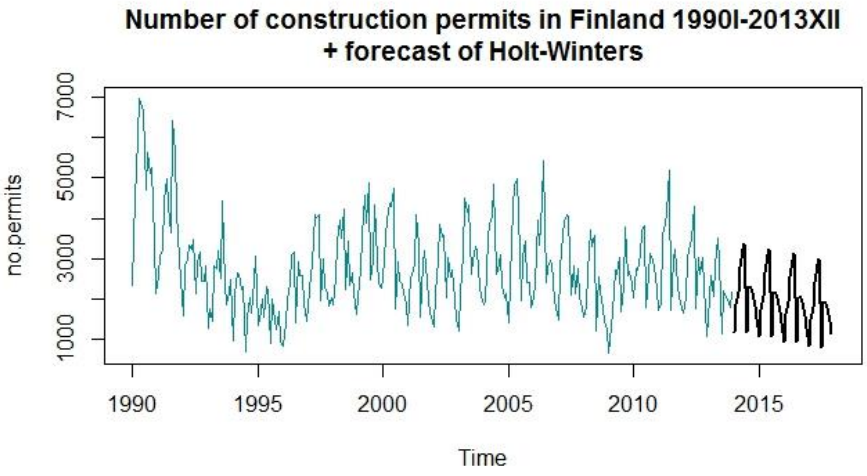



```
plot(no.permits, col="cyan4", main="Number of construction permits in Finland
1990I-2013XII
     + fitted Holt-Winters")
lines(HoltWinters(no.permits)$fit[,1],col="red", lwd=2)
```



Predicting the future values using Holt-Winters values can be done using a generic function predict().It only requires that we rescale the x-axis so that the new values can be fitted in the graph.

```
# Predicting using Holt-Winters
plot(no.permits, col="cyan4", main="Number of construction permits in Finland
1990I-2013XII
     + forecast of Holt-Winters", xlim=c(1990,2018))
lines(predict(HoltWinters(no.permits),n.ahead=48),col="black", lwd=2)
```



5.6.2. ARIMA models

ARIMA models are used for describing the evolution of a single (univariate) time series. ARIMA are constructed using two kinds of components: autoregressive (AR) and moving average (MA). They are applied for forecasting.

In ARIMA modeling often one prefer to use Box-Jenkins approach consisting of three stages (in the application order):

1) Model identification
2) Parameter estimation
3) Diagnostic checking

These stages can be repeated until the model suits the data considerably well. The goal is to find the most parsimonious model, i.e. the model with the smallest amount of parameters p and q.

Autocorrelation (AC) and partial autocorrelation (PAC) functions play a key role, because they help to identify the number of parameters (p and q in $ARMA(p,q)$) or the order that may be needed. In partial autocorrelation function the impact of values $y_{t-1}, \dots, y_{t-s+1}$ is not included.

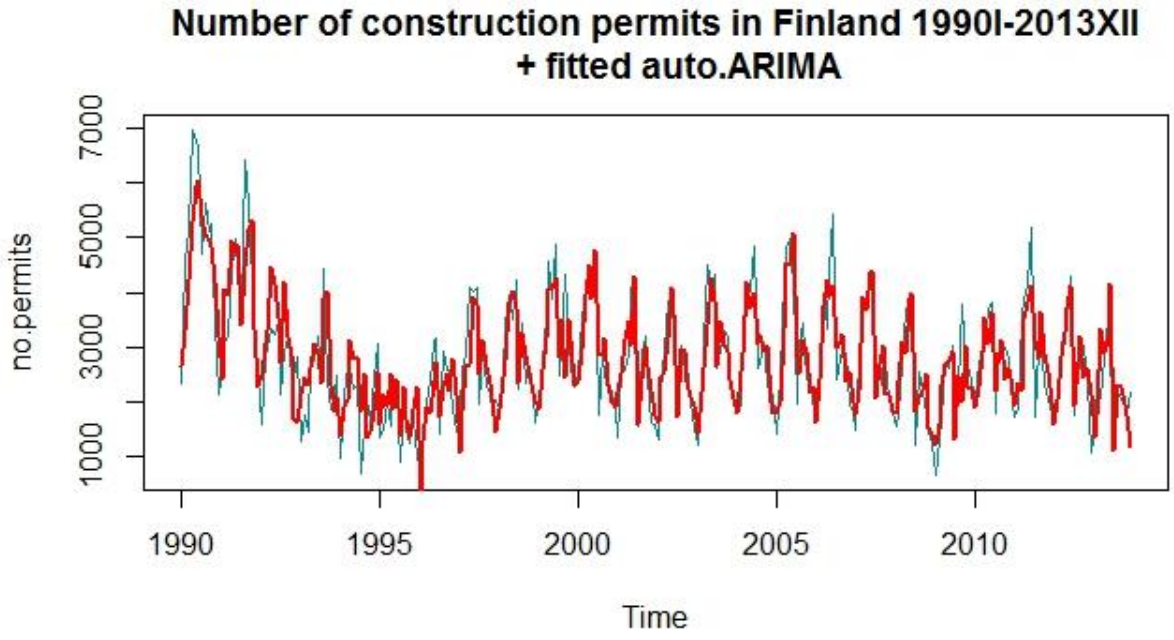
A rule of thumb is that for $ARMA(p,q)$

- AC function's values decay after lag q (but may alternate in sign)
- PAC function's values decay after lag p (but may alternate in sign)
- For $AR(p)$ PAC function's values are zeroes for $s > p$

Example 70. .....

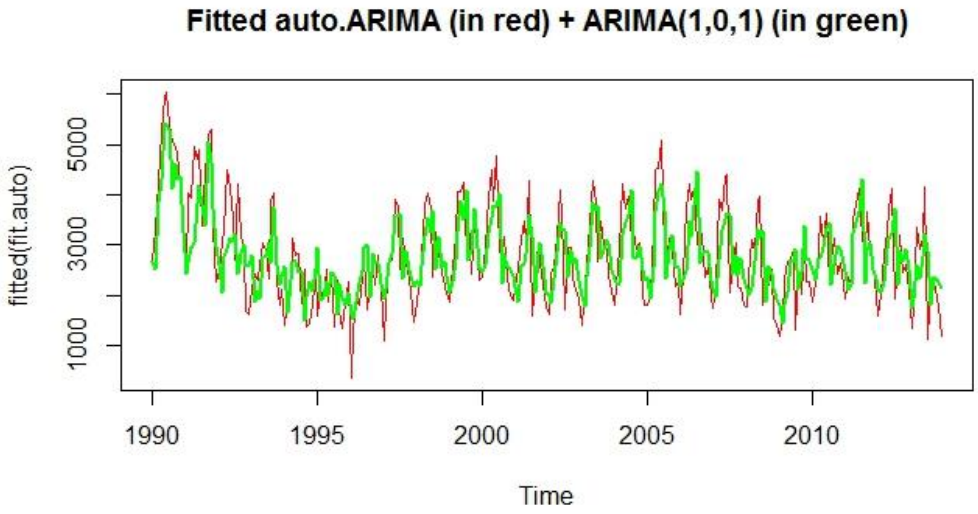
This example demonstrates how the estimated model by *auto.arima()* compares with the original series:

```
plot(no.permits, col="cyan4", main="Number of construction permits in Finland
1990I-2013XII
     + fitted auto.ARIMA (in red)")
lines(fitted(fit.auto), col="red", lwd=2)
```



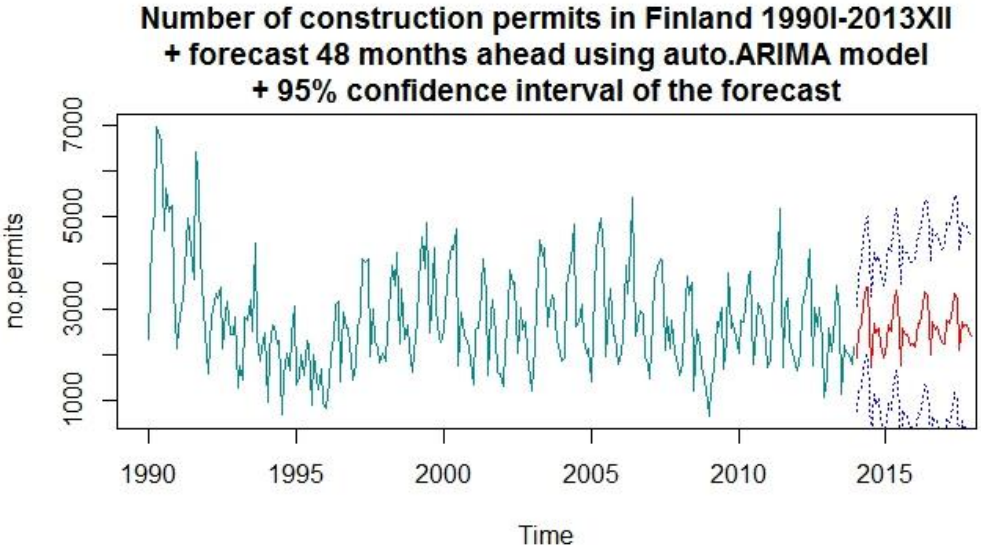
And fitted auto.ARIMA against $ARIMA(1,0,1)$:

```
nobs=length(no.permits)
manual.fit <- arima(no.permits,order=c(1,0,1) , xreg=1:nobs)
summary(manual.fit)
plot(fitted(fit.auto), col="red", main="Fitted auto.ARIMA (in red) +
ARIMA(1,0,1) (in green)")
lines(fitted(manual.fit), col="green", lwd=2)
```



How to forecast using ARIMA? The simplest way is to use the generic function predict():

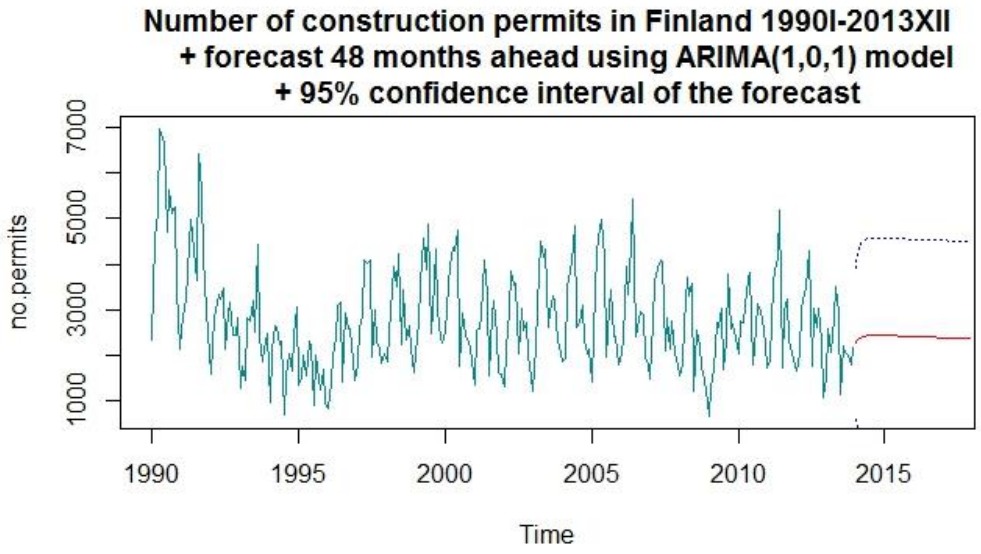
```
no.permits.pred <- predict(fit.auto, n.ahead=48)
plot(no.permits, col="cyan4", main="Number of construction permits in Finland
1990I-2013XII
     + forecast 48 months ahead using auto.ARIMA model
     + 95% confidence interval of the forecast", xlim=c(1990,2017))
lines(no.permits.pred$pred,col="red")
lines(no.permits.pred$pred+1.96*no.permits.pred$se,col="blue",lty=3)
lines(no.permits.pred$pred-1.96*no.permits.pred$se,col="blue",lty=3)
```



And here comes the forecast by $ARIMA(1,0,1)$ without seasonal parameters:

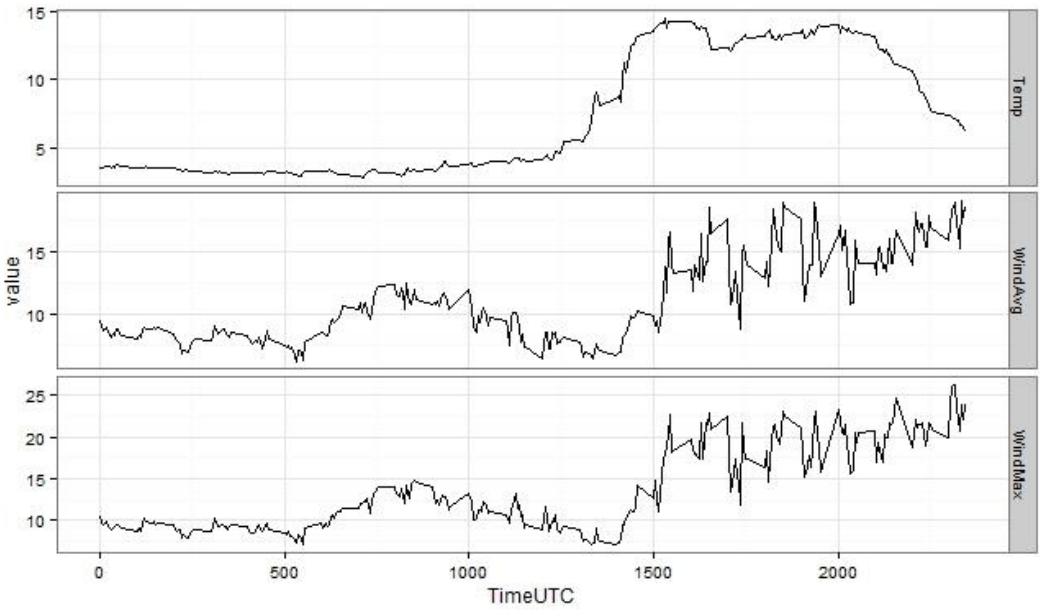
```
no.permits.pred2 <- predict(manual.fit, n.ahead=48,
newxreg=(nobs+1):(nobs+48))
plot(no.permits, col="cyan4", main="Number of construction permits in Finland
1990I-2013XII
```

```
     + forecast 48 months ahead using ARIMA(1,0,1) model
     + 95% confidence interval of the forecast", xlim=c(1990,2017))
lines(no.permits.pred2$pred,col="red")
lines(no.permits.pred2$pred+1.96*no.permits.pred2$se,col="blue",lty=3)
lines(no.permits.pred2$pred-1.96*no.permits.pred2$se,col="blue",lty=3)
```



5.6.3. VAR models

How can we test for the impact of a variable on other variables over time?



$VAR(p)$ is a system of k equations with $k \times p$ explanatory variables (p lags of each of the k variables included in y), plus k constants (if means are not zeros). Say, we have two variables $y_{1,t}$ and $y_{2,t}$, then VAR(1) model is

$$\begin{cases} y_{1,t} = \varphi_{11} y_{1,t-1} + \varphi_{12} y_{2,t-1} + \varepsilon_{1,t} \\ y_{2,t} = \varphi_{21} y_{1,t-1} + \varphi_{22} y_{2,t-1} + \varepsilon_{2,t} \end{cases}$$

which can be expressed as

$$\boldsymbol{y}_t = \boldsymbol{\Phi} \boldsymbol{y}_{t-1} + \boldsymbol{\varepsilon}_t,$$

where

$$\boldsymbol{y}_t = \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix}, \quad \boldsymbol{\varepsilon}_t = \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix}, \quad \boldsymbol{\Phi} = \begin{bmatrix} \varphi_{11} & \varphi_{12} \\ \varphi_{21} & \varphi_{22} \end{bmatrix}.$$

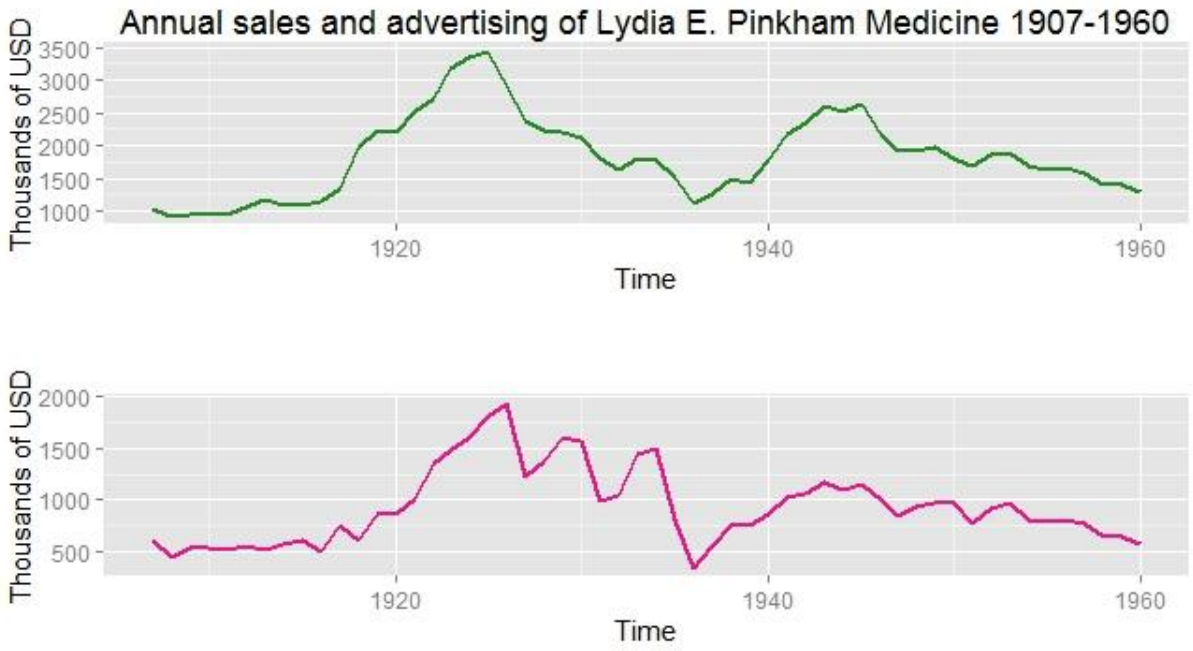
In general $VAR(p)$ model can be expressed as

$$\boldsymbol{y}_t = \boldsymbol{\Phi}_1 \boldsymbol{y}_{t-1} + \boldsymbol{\Phi}_2 \boldsymbol{y}_{t-2} + \cdots + \boldsymbol{\Phi}_p \boldsymbol{y}_{t-p} + \boldsymbol{\varepsilon}_t.$$

The equation above is called a reduced from VAR. Error terms of the linear equations are allowed to be correlated with each other, but since the explanatory variables are the same this setup can be estimated using OLS.

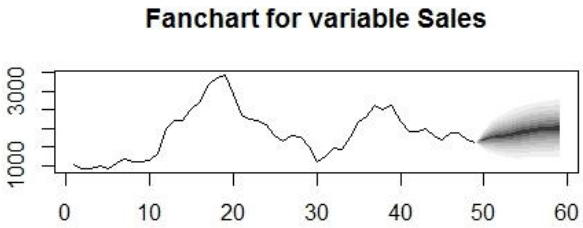
VAR model can be expressed using lag polynomials as $\Phi(L)y_t = \varepsilon_t$, or equivalently $y_t = \Phi(L)^{-1}\varepsilon_t = \sum_{j=0}^{\infty}\psi_j\varepsilon_{t-j} = \boldsymbol{\Psi\varepsilon}_t$, where $\psi_0 = I$ is an identity matrix. This is a particularly practical form, since many times one is interested to know how a shock in one variable is reflected in another one. $\boldsymbol{\Psi}$ is thus a matrix of impulse responses. An impact of one unit (standard deviation) shock in one error term on each of the variables in the model produces *impulse response functions.* Sometimes the shocks $\varepsilon_{1,t}, \varepsilon_{2,t}$ … can be correlated making it difficult distinguishing shocks from each other. Hence, the model should be transformed (orthogonized) to make error terms uncorrelated.

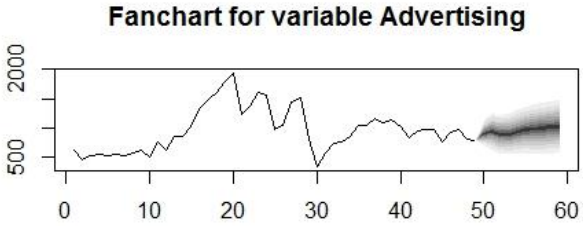
Example 71. .....



Does advertising create more sales or other way around? Let's investigate that.

```
library(vars)
library(tseries)

lydia=ts(lydia[,],start=c(1907,1),deltat=1)
lydia
colnames(lydia)=c("Sales","Advertising")
lydia1955<-window(lydia,end=1955)
dim(lydia1955)
var3_1955=VAR(lydia1955, p = 3, type = c("const"),season = NULL, exogen =
NULL, lag.max = NULL)
summary(var3_1955)
fcast=predict(var3_1955,n.ahead=10,ci=0.95)
fcast$fcst
fanchart(fcast)
```





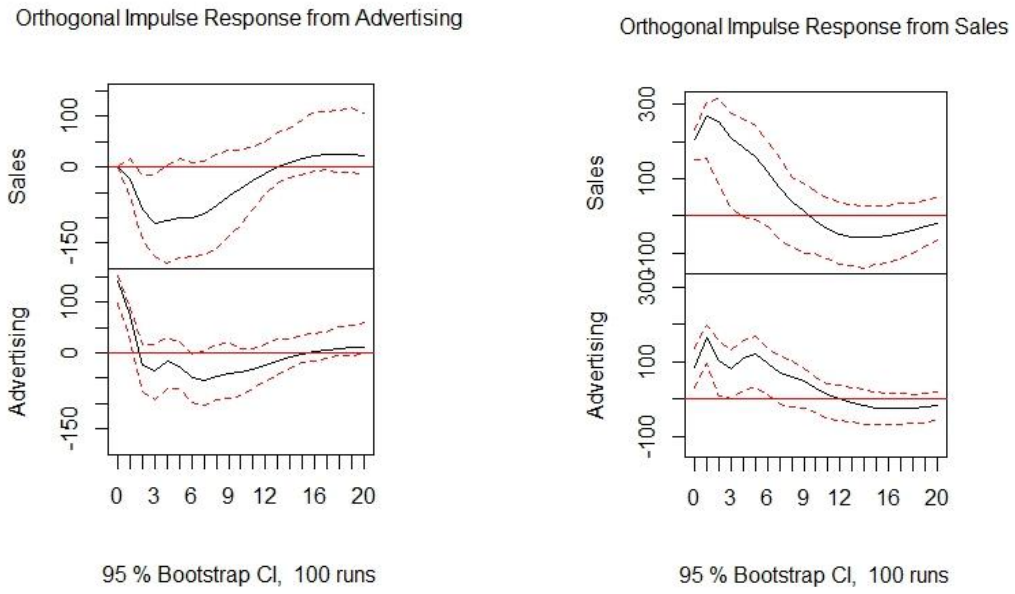

```
> cause_sales=causality(var_lag3, cause = "Sales")
> cause_adv=causality(var_lag3, cause = "Advertising")
> cause_sales
$Granger

      Granger causality H0: Sales do not Granger-cause Advertising

data:  VAR object var_lag3
F-Test = 9.6761, df1 = 3, df2 = 88, p-value = 1.385e-05


$Instant

      H0: No instantaneous causality between: Sales and Advertising

data:  VAR object var_lag3
Chi-squared = 10.2754, df = 1, p-value = 0.001348


> cause_adv
$Granger

      Granger causality H0: Advertising do not Granger-cause Sales

data:  VAR object var_lag3
F-Test = 1.8284, df1 = 3, df2 = 88, p-value = 0.1478


$Instant

      H0: No instantaneous causality between: Advertising and Sales

data:  VAR object var_lag3
Chi-squared = 10.2754, df = 1, p-value = 0.001348
```

Based on our model it seems that advertising is not causing more sales!

Example 72. .....

Impulse response functions is a nice way to see how changes in one variable get reflected in another variable.

```
impulse=irf(var_lag3, n.ahead=20, ortho=TRUE, boot=TRUE,ci=0.95,runs=100,
cumulative=FALSE)
plot(impulse)
```



5.7. Bayesian inference (incomplete)

5.8. Tree-based methods (incomplete)

Tree-based methods are powerful yet simple to interpret classification tools.

5.9. Support vector classifier and support vector machine

The support vector machine (SVM) is an extension of the support vector classifier which is an extension of the maximal margin classifier. All definitions aside, SVM is a hands-on technique with a rather engineering flavor to classify the variables that are correlated in a more complex way than the one that

would be possible to model with a logistic regression. SVM has gained its reputation because of its functionality and flexibility, and that is the reason why it is being introduced here. SVM can be used to classify the data into many classes, but in this chapter we will only go through the binary classification setting.

The concept of a hyperplane is a foundation of the technique being employed in the classification methods applied in SVM. Hyperplane a $p-1$ –dimensional subspace of the p –dimensional space. Hence, in the case of the ordinary 3-dimensional space its hyperplane is 2-dimensional. This means that the hyperplane of the 3-dimensional space is a plane. From that, the reader can easily infer that the hyperplane of a 2-dimensional space is a line, which is a 1-dimensional object. Mathematically a p -dimensional hyperplane in a $p+1$ –dimensional space is defined as

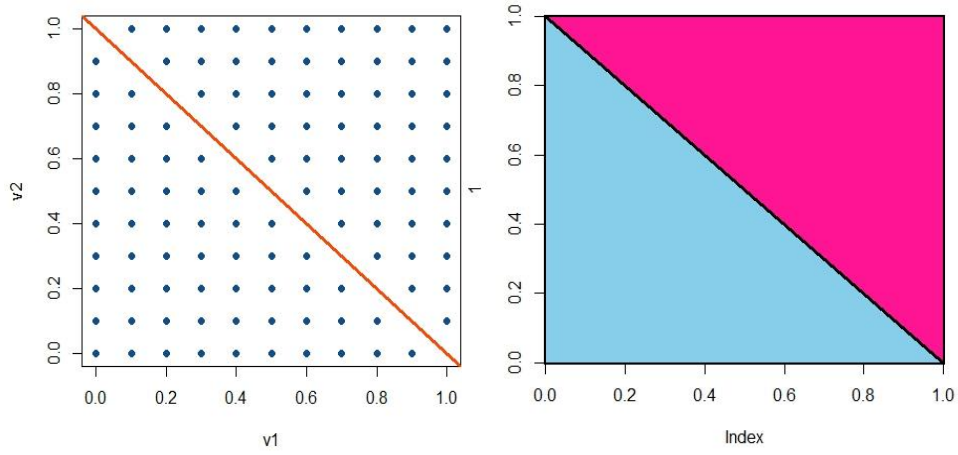
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

and a 2-dimensional hyperplane stands for a linear equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

where $X = (X_1, X_2)^T$ stands for a vector that assigning the points the hyperplane. This case can be showcased graphically as follows.

```
v1<- rep(seq(0,1,0.1), times=10, each=1)
v2<- rep(seq(0,1,0.1), times=1, each=10)
m.svm<-cbind(v1,v2)
plot(m.svm, xlim=c(0,1), ylim=c(0,1), pch=19, col="dodgerblue4")
abline(coef = c(1,-1), lwd=3, col='orangered')
plot(1,xlim=c(0.035,0.965), ylim=c(0.035,0.965))
polygon(x=c(0,0,1), y=c(0,1,0), col = 'skyblue')
polygon(x=c(1,0,1), y=c(1,1,0), col = 'deeppink')
abline(coef = c(1,-1), lwd=3)
```



The line in the graph represents a hyperplane

$$X_1 + X_2 - 1 = 0,$$

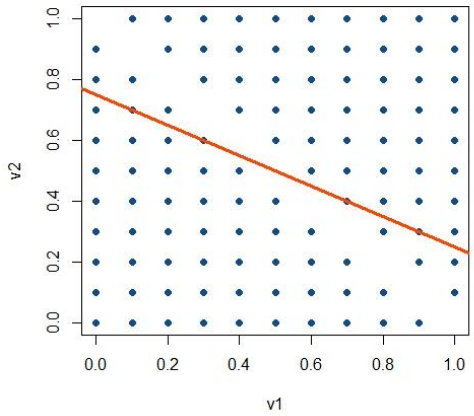
where X_1 is represented by a vector v_1 and X_2 is represented by v_2. Thus, $\beta_0 = -1$, $\beta_1 = 1$, and $\beta_2 = 1$. All points which lie above or below the line in the left-hand side of the graph satisfy an equation

$$X_1 + X_2 - 1 > 0$$

or

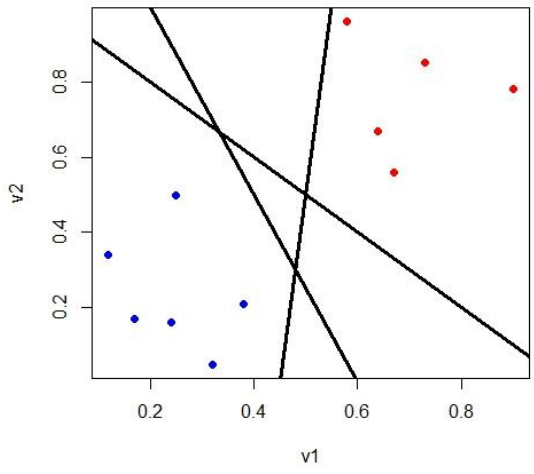
$$X_1 + X_2 - 1 < 0$$

respectively, and are represented by a purple and a blue area in the right-hand side of the graph. For instance, in case of $3 - 2X_1 - 4X_2 = 0$, the graph would look as follows.



Now, say, that we'd like to classify two sets of points in a 2-dimensional setting, then the hyperplane can be used for that purpose (i.e. decision making):

```
v1=c(0.12,0.32,0.38,0.24,0.17,0.25,.64,.58,.73,.67,.90)
v2=c(0.34,0.05,0.21,0.16,0.17,0.50,.67,.96,.85,.56,.78)
y=c(rep(-1,6) ,rep (1,5))
m.svm2<-cbind(v1,v2)
plot(m.svm2, col =(3-y), pch=19)
abline(coef = c(1,-1), lwd=3)
abline(coef = c(-4.5,10), lwd=3)
abline(coef = c(1.5,-2.5), lwd=3)
```



Recall from the linear algebra course or high school mathematics, the formula for line equation is

$$y_0 - y = \frac{y - y_0}{x - x_0}(x - x_0),$$

where $(y - y_0)/(x - x_0)$ is the slope coefficient / first derivative of the equation.

Based on the example above we may be able construct many different separating planes (not always the case). From the result above we may as well conclude that for a line to be a separating plane for two classes it has to satisfy the following property:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) > 0,$$

where $y_i = \{-1,1\}$. And from this result we may generalize further that

$$y_i\big(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\big) > 0,$$

for a p –dimensional hyperplane, for all $i = 1, \dots, n$ observations.

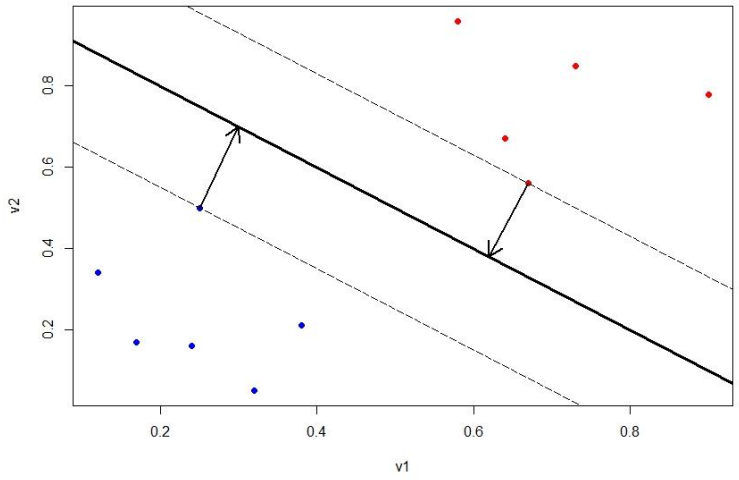
After being able to find several separating hyperplanes the natural question arises, what would be the best one to classify the data? One approach is to search for a separating hyperplane with the largest possible M, that represents the minimum distance from each point to that particular plane. Formally this becomes a maximization problem with the following setup

$$\underset{\beta_0,\beta_1,\dots,\beta_p}{\text{maximize}}\, M$$

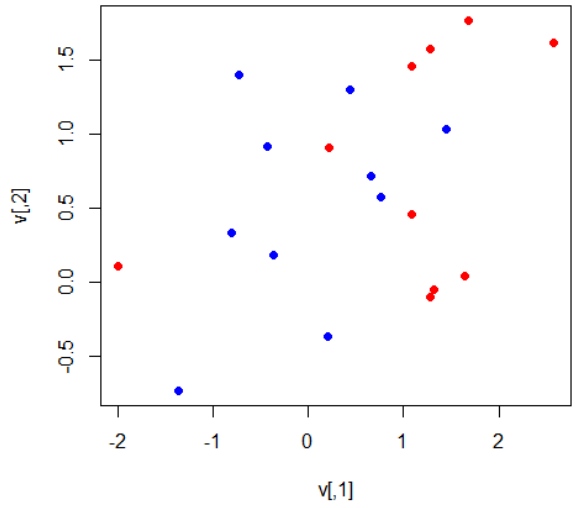
$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i\big(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\big) \geq M \;\forall i = 1, \dots, n.$$

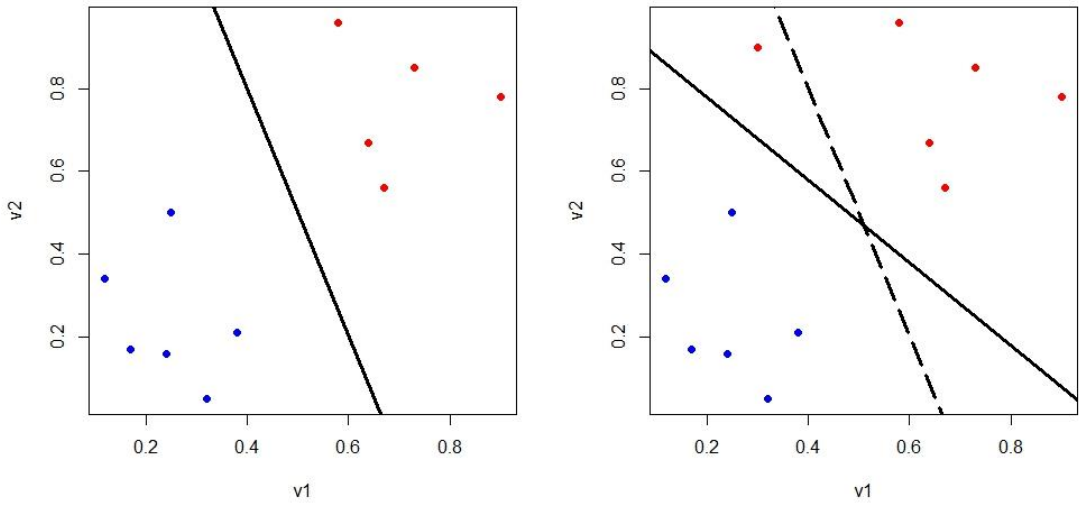
In case one is able to find such M we call it the maximal margin hyperplane. Say, we have a two class set of observations, then the distance of observations from the hyperplane looks graphically as follows:



The two points connected with arrows are the ones with minimal distance from the hyperplane in case. Sometimes it is not possible to separate the set of points linearly, i.e. $M > 0$ does not exist for that case as shown in the graph below. Nevertheless, it is possible to apply a support vector classifier technique in that case.



On the other hand, even when there's a possibility to separate observations into classes, introducing new observations may lead to contradictions with a defined hyperplane, which is showcased in the graph below.



In case a new observation doesn't fit a predefined hyperplane, it can be a sign of an inappropriate hyperplane. Since a separating hyperplane is intended to perfectly classify the observations, there's a chance for it to be sensitive to individual observations. Adjusting the hyperplane every time after new observations are introduced may work as well, but this operation has a tendency to reduce the margins of the classifier, while the margins can be seen as a measure of confidence. On top of that, a real life data is rarely perfectly classifiable, therefore instead of searching for a perfect classifier one could try to search for techniques bringing more robustness to the single observations and instead focus on finding a classifier that is classifying most observations at a satisfactory level that could be just enough to make decisions and accept a certain level of uncertainty.

Accepting a possible non-perfectness of a classifier brings us to a concept of the support vector classifier. Its maximization problem is defined as follows:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots\epsilon_n}{\text{maximize}} M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

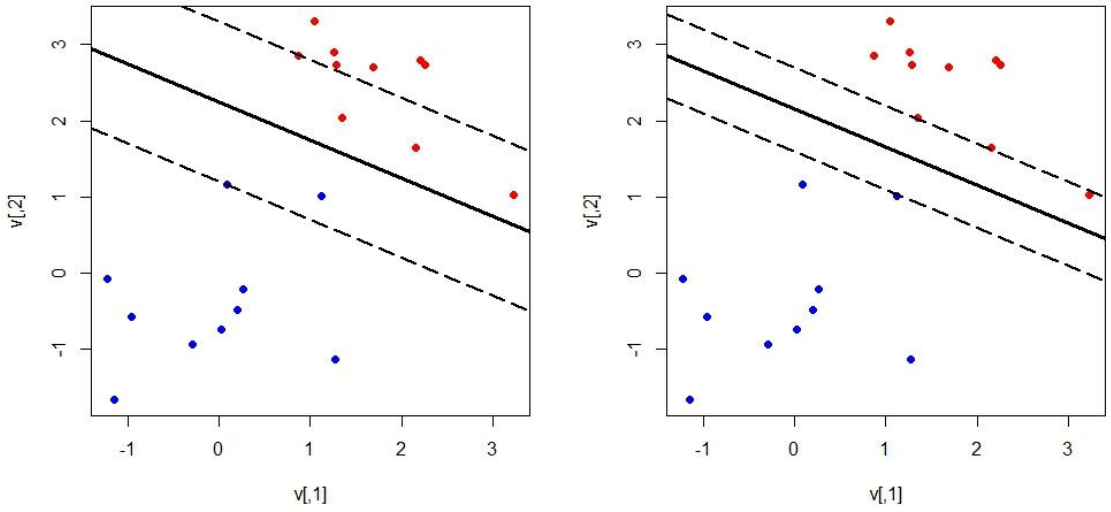
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)\ \forall i = 1, \ldots, n,$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

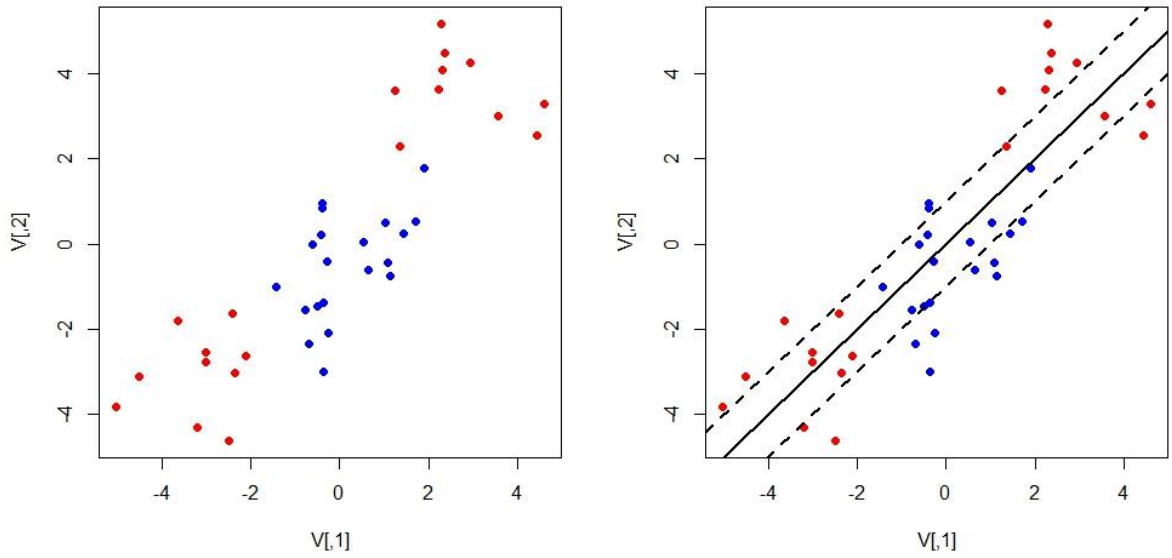
where C is a nonnegative tuning parameter, M is the width of the margin, and $\epsilon_1, \dots, \epsilon_n$ are slack variables to allow single observations to end up on the wrong side of the margin. Slack variable ϵ_i tells us the location of the ith variable with respect to the classifier and it provides the following information:

$$\begin{aligned} \epsilon_i = 0 &\quad \text{if observation } i \text{ is on the correct side of the margin} \\ 0 < \epsilon_i \leq 1 &\quad \text{if observation } i \text{ violates the margin} \\ \epsilon_i > 1 &\quad \text{if observation } i \text{ is on the wrong side of the hyperplane.} \end{aligned}$$

In the light of the slack variable's definition the parameter C can be seen as a total level of violations caused by the observations with respect to the hyperplane in case. On the other hand C is positively correlated with the choice of M, as increasing the margins inevitably increases the violations as the number of observations grows, which is demonstrated in the following graph.



So far we have focused on the classifiers that are linear. Unfortunately this set of classifiers has big limitations when the data is not clustered linearly. In these cases even accepting a non-perfectness of the classifier can be not sufficient enough in terms of its performance. Have a look at the following graph.



In the graph above one can clearly identify three clusters of data points that belong to either class. Trying to fit a linear classifier in this case can be rather disappointing and this becomes obvious in the right-hand side of the picture.

One solution to this problem could be to use non-linear restrictions with more features, i.e. instead of using only linear observations $X_1, X_2, \dots, X_p$ we could use their second power $X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2$ and even higher power m in general:

$$X_1, X_1^2, \dots, X_1^m, X_2, X_2^2, \dots X_2^m, \dots, X_p, X_p^2, \dots, X_p^m$$

while doubling or multiplying the number of parameters even further. In that case the familiar maximization problem becomes

$$\underset{\beta_0, \beta_{11}, \beta_{1m}, \dots, \beta_{p1}, \beta_{pm}, \epsilon_1, \dots \epsilon_n}{\text{maximize}} M$$

$$\text{subject to} \sum_{j=1}^{p} \sum_{k=1}^{m} \beta_{jk}^2 = 1,$$

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 + \cdots + \sum_{j=1}^{p} \beta_{jm} x_{ij}^m \right) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

Even though the decision boundary in this cases is linear the solutions of this maximization problem are in general non-linear. Unfortunately increasing the parameters brings another drawback to this problem: computations. Luckily, there is another efficient way to deal with nonlinearities while controlling for the computation capacity that the problem involves. The support vector machine (SVM) is one more extension to deal with this set of challenges. The idea of the support vector machine is to model non-linear boundaries between the classes, but instead of using linear parameters its foundation lies in kernels. The theory of the kernels is complex, therefore we will omit it, but some of the most commonly used kernels will be introduced in a general manners and then some of the examples of their use will be provided after that.

Let $K(x_i, x_{i^*})$ be the kernel, where observations $\{x_n\}$ represent training vectors. The most basic kernel used in SVM is a **linear** kernel that is in fact the inner product of the vectors x_i and x_{i^*}, $\langle x_i, x_{i^*} \rangle = \sum_{j=1}^{p} x_{ij} x_{i^* j}$. Its extension is a **polynomial** kernel $\left(1 + \sum_{j=1}^{p} x_{ij} x_{i^* j}\right)^d$. Another commonly used kernel is called **radial**, and it takes a form: $\exp\left(-\gamma \sum_{j=1}^{p} \left(x_{ij} - x_{i^* j}\right)^2\right)$. Finally, the last kernel that deserves our attention is called **hyperbolic tangent** or **sigmoid**, and it takes a form as follows: $\tanh\left(\kappa \sum_{j=1}^{p} x_{ij} x_{i^* j} + c\right)$. Thus, the four most commonly applied kernels are

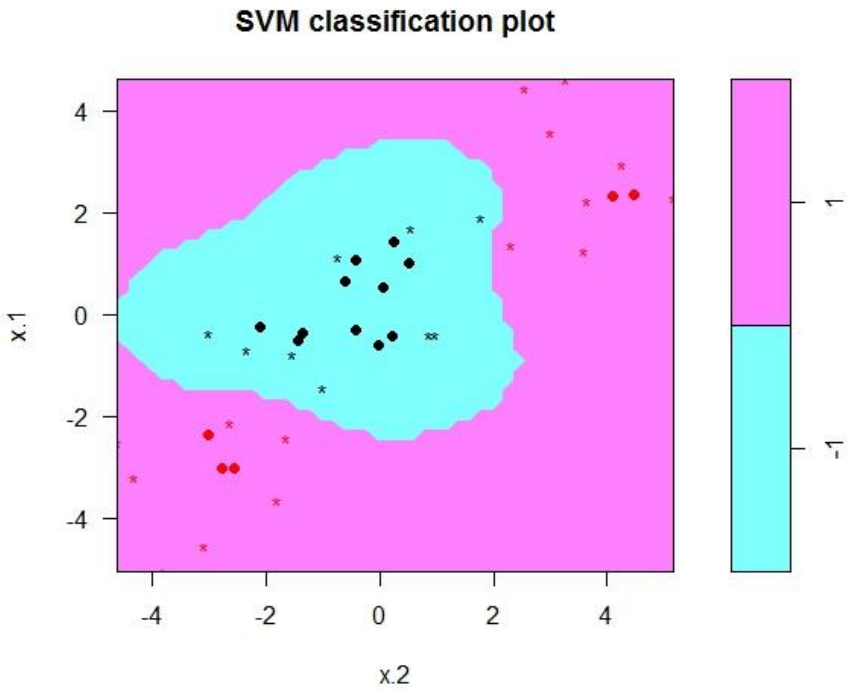
$$K(x_i, x_{i^*}) = \begin{cases} \sum_{j=1}^{p} x_{ij} x_{i^* j} \\ \left(1 + \sum_{j=1}^{p} x_{ij} x_{i^* j}\right)^d, \text{for } d > 1 \\ \exp\left(-\gamma \sum_{j=1}^{p} \left(x_{ij} - x_{i^* j}\right)^2\right), \text{for } \gamma > 1 \\ \tanh\left(\kappa \sum_{j=1}^{p} x_{ij} x_{i^* j} + c\right), \text{for } \kappa > 0 \text{ and } c < 0. \end{cases}$$

You have to have a package "e1071" installed in order to run the following examples. It's an easy use package to elaborate SVM's.

Example 73. .....

```
V<-matrix(rnorm(80,0,1), ncol = 2)
y=c(rep(-1,20), rep(1,20))
V[21:30,]=V[21:30,] + 3
V[31:40,]=V[31:40,] - 3
dat=data.frame(x=V, y=as.factor(y))
library(e1071)
```

```
tune.out=tune(svm ,y~.,data=dat ,kernel ="radial",
              ranges =list(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
tune.out$best.model

Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost =
c(0.001, 0.01, 0.1,
    1, 5, 10, 100)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  14

svmfit=svm(y~., data=dat, kernel="radial", cost=1, gamma=0.5, scale=FALSE)
plot(svmfit, dat, fill = TRUE, svSymbol = 42, dataSymbol = 19)
```

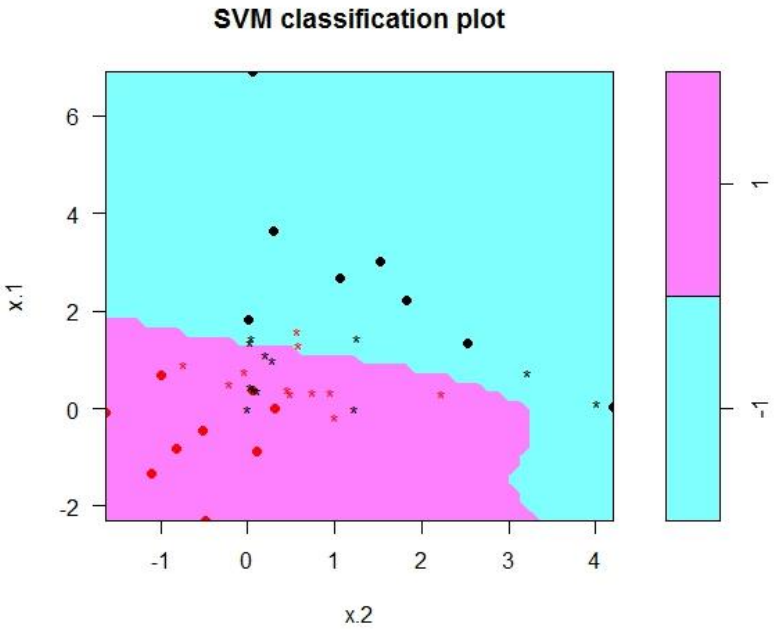


This is a radial classification, where the star signs stand for the support vectors and dots are data symbols.

Example 74......

```
V.poly<-matrix(rnorm(80,0,1), ncol = 2)
y=c(rep(-1,20), rep(1,20))
V.poly[1:20,]=(V.poly[1:20,])^2
dat=data.frame(x=V.poly, y=as.factor(y))
tune.out=tune(svm ,y~.,data=dat ,kernel ="polynomial",
```

```
              ranges =list(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
summary(tune.out)
bestmod=tune.out$best.model
bestmod


Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost =
c(0.001, 0.01, 0.1,
    1, 5, 10, 100)), kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  0.5
     coef.0:  0

Number of Support Vectors:  28

svmfit=svm(y~., data=dat, kernel="polynomial", cost=1, degree=3, gamma=0.5,
coef.0=0, scale=FALSE)
plot(svmfit, dat, fill = TRUE, svSymbol = 42, dataSymbol = 19)
```

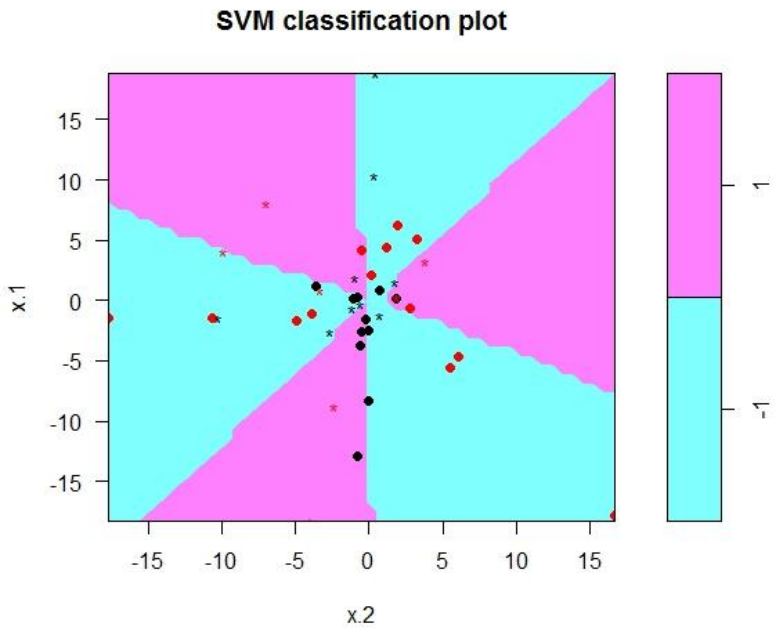


This is a polynomial classification method.

Example 75......

```
V.tan<-matrix(rnorm(80,0,8), ncol = 2)
y=c(rep(-1,20), rep(1,20))
V.tan[1:20,]=tan(V.tan[1:20,])
plot(V.tan, col=(3-y),pch=19)
```

```
dat=data.frame(x=V.tan, y=as.factor(y))
tune.out=tune(svm ,y~.,data=dat ,kernel ="sigmoid",
              ranges =list(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
tune.out$best.model

Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost =
c(0.001, 0.01, 0.1,
    1, 5, 10, 100)), kernel = "sigmoid")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  sigmoid
       cost:  10
      gamma:  0.5
     coef.0:  0

Number of Support Vectors:  27

svmfit=svm(y~., data=dat, kernel="sigmoid", cost=10, gamma=0.5, coef.0=0,
scale=FALSE)
tmp<-dat[,1]
dat[,1]<-dat[,2]
dat[,2]<-tmp
plot(svmfit, dat, fill = TRUE, svSymbol = 42, dataSymbol = 19)
```



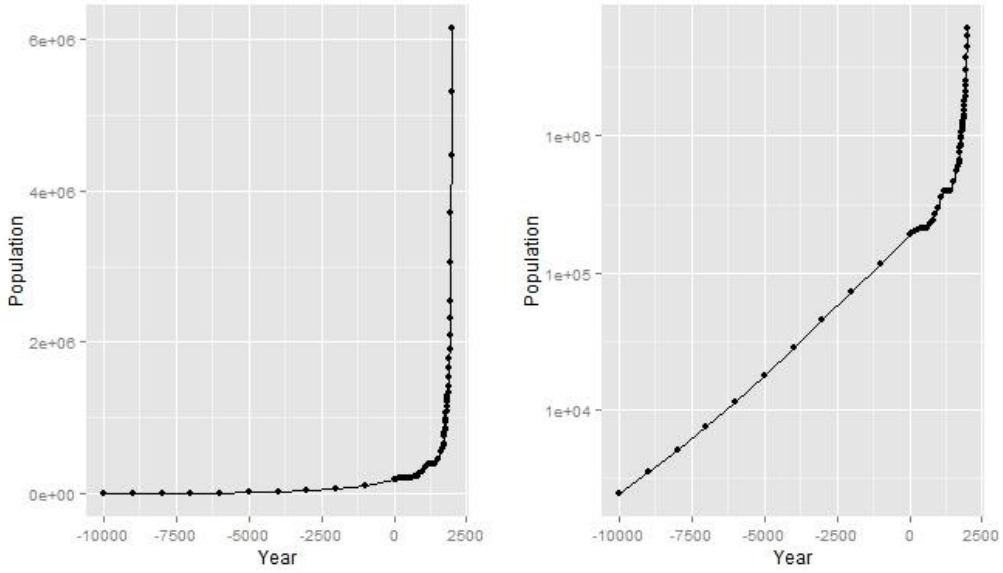
This is a hyperbolic tangent attempt to classify the (complicated) data.

5.10. Depicting the data

This section is meant to raise the issues related to displaying the data.

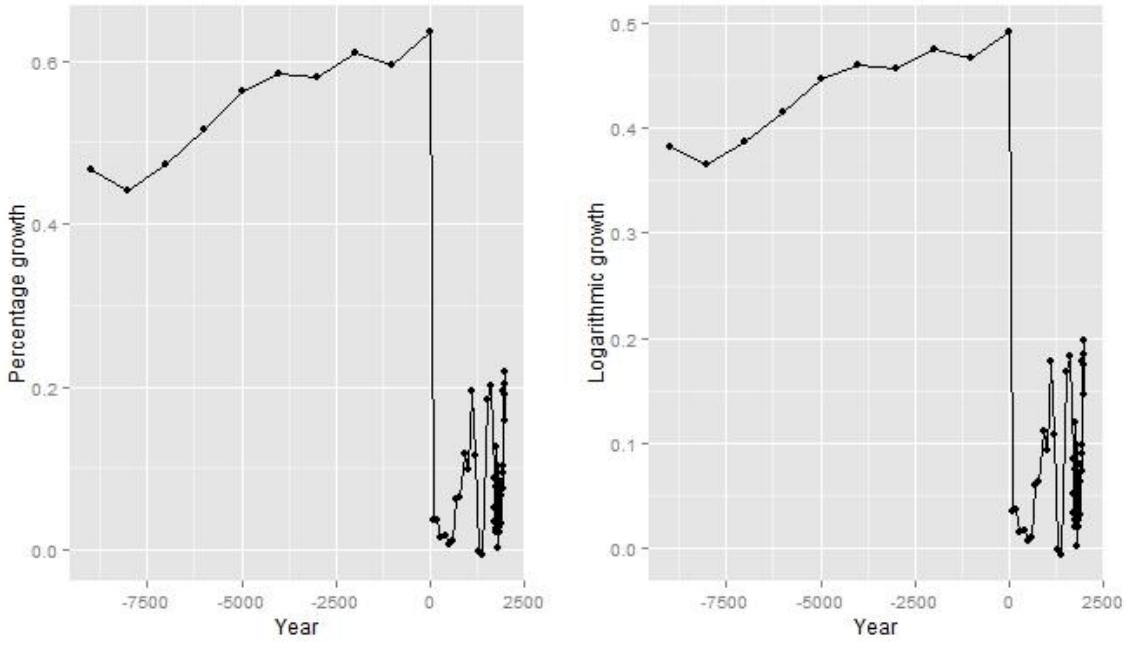
Linear scale vs. log scale

```
# World population
library(gcookbook)
library(gridExtra)
data("worldpop")
# Linear scale
linS <- ggplot(worldpop, aes(x=Year, y=Population)) + geom_line() +
geom_point()
# Log-scale
logS <- ggplot(worldpop, aes(x=Year, y=Population)) + geom_line() +
geom_point() +
  scale_y_log10()
grid.arrange(linS, logS, nrow=1, ncol=2)
```



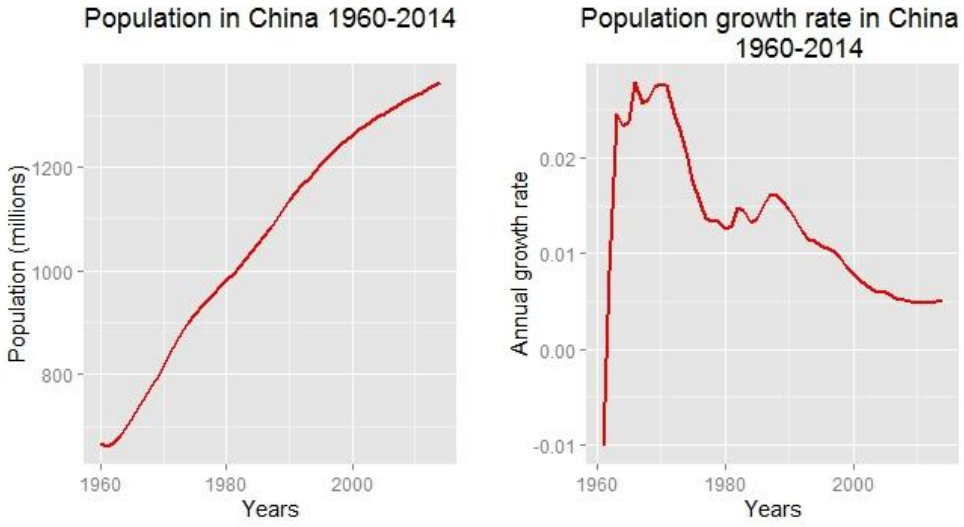
Linear growth rates vs. log growth rate

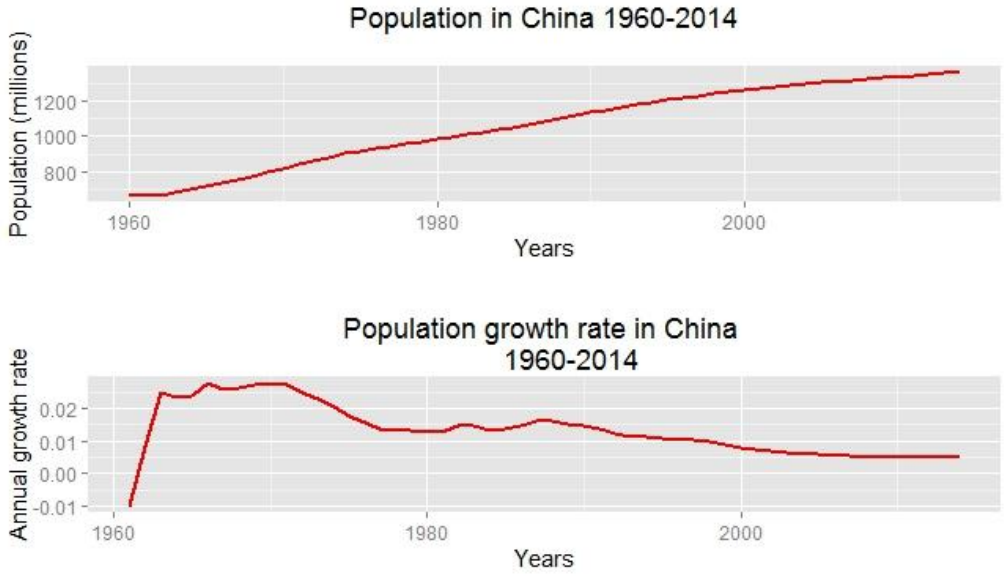
```
#World population growth
diff.p<-numeric(length(worldpop$Population)-1)
for (i in 1:length(diff.p)){
diff.p[i]<-((worldpop$Population[i+1]-
worldpop$Population[i])/worldpop$Population[i])
}
diff.log.p<-diff(log(worldpop$Population))
worldpop2<-cbind(worldpop[2:length(worldpop$Population),],diff.p,diff.log.p)
# Linear scale
diff.linS <- ggplot(worldpop2, aes(x=Year, y=diff.p)) + geom_line() +
geom_point() + ylab("Percentage growth")
# Log-scale
diff.logS <- ggplot(worldpop2, aes(x=Year, y=diff.log.p)) + geom_line() +
geom_point() + ylab("Logarithmic growth")
grid.arrange(diff.linS, diff.logS, nrow=1, ncol=2)
```



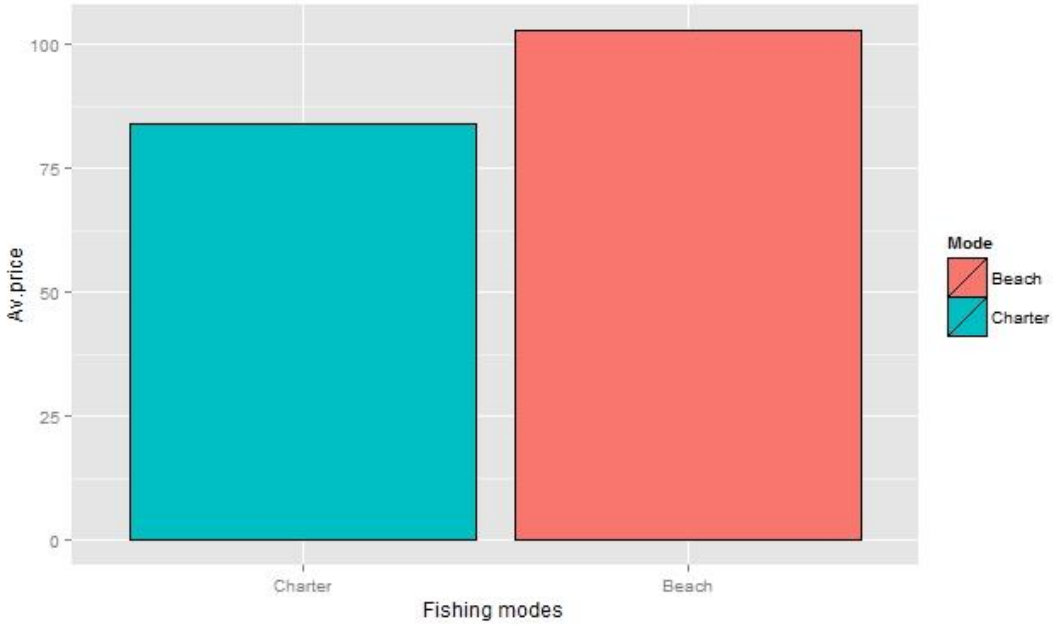
Cutting and scaling the axes

```
pop <- ts(china$Pop, start = c(1960), frequency = 1)
library(ggplot2)
library(ggfortify)
p1 <- autoplot(pop/10^6, ts.colour = 'red2', size = 1)
p1 <- p1 + theme_grey(base_size = 15) +
  scale_x_continuous(name="Years") +
  scale_y_continuous(name="Population (millions)") +
  labs(title = "Population in China 1960-2014
       ", size=3)
p2 <- autoplot(diff(log(pop/10^6)), ts.colour = 'red2', size = 1)
p2 <- p2 + theme_grey(base_size = 15) +
  scale_x_continuous(name="Years") +
  scale_y_continuous(name="Annual growth rate") +
  labs(title = "Population growth rate in China
       1960-2014", size=3)
grid.arrange(p1,p2,nrow=1,ncol=2)
grid.arrange(p1,p2,nrow=2,ncol=1)
```

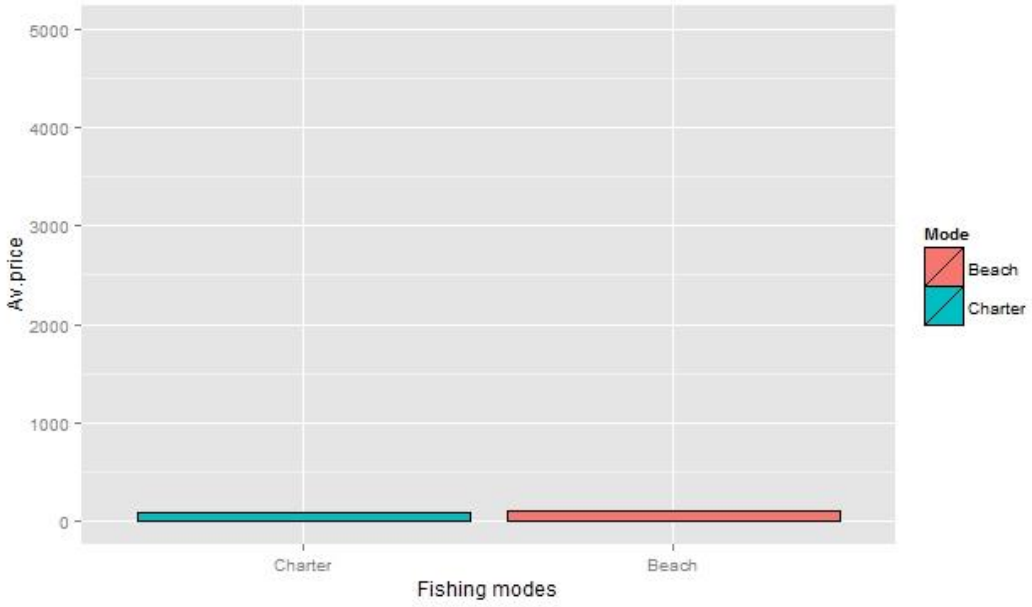




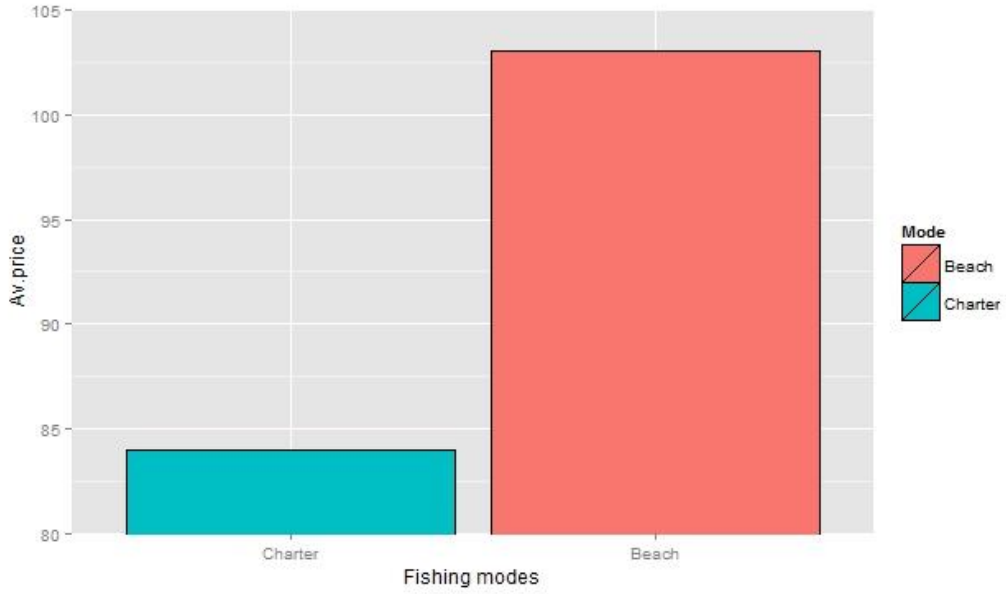

```
#Average prices of fishing using 'charter' and 'beach'
library(mlogit)
data(Fishing)
prices<-cbind(c("Beach","Charter"), c(round(mean(Fishing$price.beach)),
round(mean(Fishing$price.charter))))
colnames(prices)<-c("Mode", "Av.price")
prices<- as.data.frame(prices)
prices$Av.price<- as.numeric(levels(prices$Av.price))
gg <- ggplot(prices, aes(x=reorder(Mode, Av.price), y=Av.price, fill=Mode)) +
geom_bar(stat="identity", colour="black")
gg
```




```
#Average prices of fishing using 'charter' and 'beach' with a rescaled
gg + scale_y_continuous(limits=c(0, 5000))
```



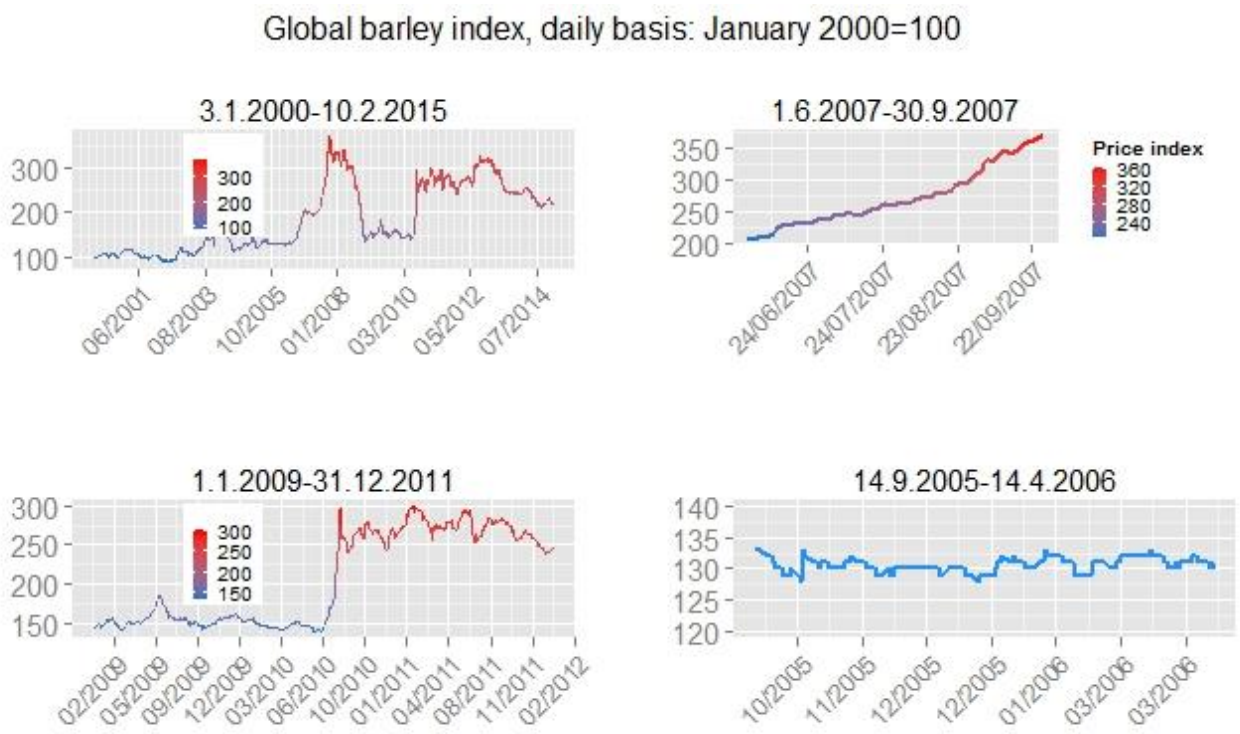

```
#Average prices of fishing using 'charter' and 'beach' with a limited y axis
gg +  coord_cartesian(ylim = c(80, 105))
```



Spans of time series

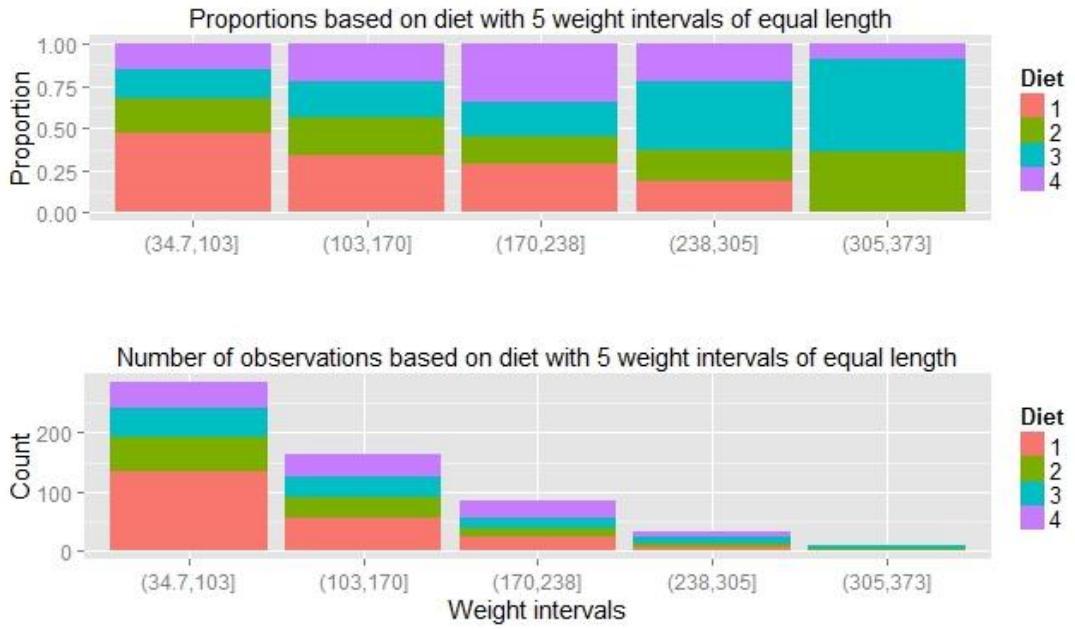
```
library(scales)
library(grid)
library(ggplot2)
library(gridExtra)
library(ggfortify)
library (plyr)
#Global barley index 3.1.2000-10.2.2015
gg1 <- ggplot(barley, aes(x=Date2, y=P)) +
  geom_line(aes(colour = P)) +
  scale_colour_gradient(high="red", low="dodgerblue3", name="") +
  scale_x_datetime(breaks = "800 days", minor_breaks = "100 days",
                   labels = date_format("%m/%Y")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
        axis.text.y = element_text(size=15),
        axis.title.x = element_text(size = 12),
        axis.title.y = element_text(size = 12),
        plot.title = element_text(size = 15),
        legend.text = element_text(size=10),
        legend.title = element_text(size=1),
        legend.key.size = unit(0.25, "cm"),
        legend.position = c(.3,.6)) +
  labs(title="3.1.2000-10.2.2015", x="", y="")
gg1

#Global barley index 1.6.2007-30.9.2007
gg2 <- ggplot(barley[1934:2019,], aes(x=Date2, y=P)) +
  geom_line(aes(colour = P), size=1) +
  scale_colour_gradient(high="red", low="dodgerblue3", name="Price index") +
  scale_x_datetime(breaks = date_breaks("30 days"),
                   labels = date_format("%d/%m/%Y")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
        axis.text.y = element_text(size=15),
        axis.title.x = element_text(size = 12),
```

```
        axis.title.y = element_text(size = 12),
        plot.title = element_text(size = 15),
        legend.text = element_text(size=10),
        legend.title = element_text(size=10),
        legend.key.size = unit(0.25, "cm")) +
  labs(title="1.6.2007-30.9.2007", x="", y="")
gg2

#Global barley index 1.1.2009-31.12.2011
gg3 <- ggplot(barley[2348:3129,], aes(x=Date2, y=P)) +
  geom_line(aes_string(col = "P", fill = NULL)) +
  scale_colour_gradient(high="red", low="dodgerblue3", name="") +
  scale_x_datetime(breaks = date_breaks("100 days"),
                   labels = date_format("%m/%Y")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
        axis.text.y = element_text(size=15),
        axis.title.x = element_text(size = 12),
        axis.title.y = element_text(size = 12),
        plot.title = element_text(size = 15),
        legend.text = element_text(size=10),
        legend.title = element_text(size=1),
        legend.key.size = unit(0.25, "cm"),
        legend.position = c(.3,.6)) +
  labs(title="1.1.2009-31.12.2011", x="", y="")
gg3

#Global barley index 14.9.2005-14.4.2006
gg4 <- ggplot(barley[1487:1639,], aes(x=Date2, y=P)) +
  geom_path(col="dodgerblue", size=1) +
  scale_x_datetime(breaks = date_breaks("30 days"),
                   labels = date_format("%m/%Y")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
        axis.text.y = element_text(size=15),
        axis.title.x = element_text(size = 12),
        axis.title.y = element_text(size = 12),
        plot.title = element_text(size = 15),
        legend.text = element_text(size=15),
        legend.title = element_text(size=12)) +
  labs(title="14.9.2005-14.4.2006", x="", y="") +
  ylim(c(120,140))
gg4

#Combining the plots
gg.l=list(gg1,gg2, gg3, gg4)
aG <- arrangeGrob(grobs=gg.l, nrow=2,ncol=2, top="Global barley index, daily
basis: January 2000=100")
grid.arrange(aG)
```



Proportions vs. counts

```
library(ggplot2)
library(grid)
library(gridExtra)
weight.int <- cut(ChickWeight$weight,5)
ChickWeight <- data.frame(ChickWeight, weight.int)
t <- theme(plot.title = element_text(size = 15),
           axis.text.x = element_text(size = 12),
           axis.text.y = element_text(size=12),
           axis.title.x = element_text(size = 15),
           axis.title.y = element_text(size = 15),
           legend.text = element_text(size=13),
           legend.title = element_text(size=13),
           legend.key.size = unit(0.5, "cm"),
           legend.position="right")
gg1 <- ggplot(ChickWeight, aes(weight.int, fill=Diet)) +
geom_bar(position="fill") + labs(title="Proportions based on diet with 5
weight intervals of equal length", x="", y="Proportion") + t
gg2 <- ggplot(ChickWeight, aes(weight.int, fill=Diet)) + geom_bar() +
labs(title="Number of observations based on diet with 5 weight intervals of
equal length", x="Weight intervals", y="Count") + t
gg.l=list(gg1,gg2)
aG <- arrangeGrob(grobs=gg.l, nrow=2,ncol=1)
grid.arrange(aG)
```

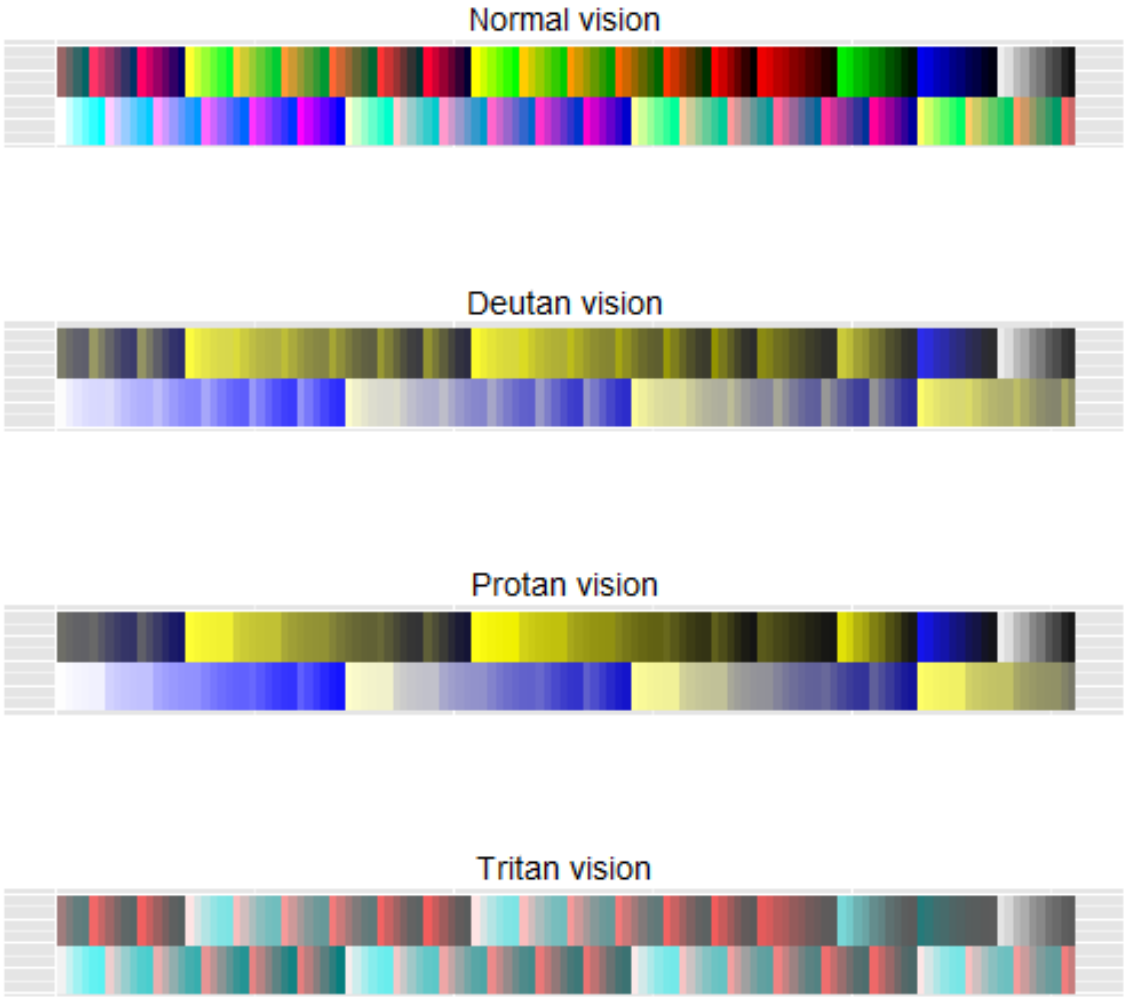


Palettes for everyone

It is estimated that 8% of all men and 0.5% of all women are affected by this problem. It is making their lives difficult and handicapped. This problem is not curable (as of Aug 4, 2015), but it is possible to help these people. The issue is color blindness. Even though the problem is called color blindness, in fact most of the people affected are not completely color blind. Instead, they are not able to distinguish some of the hues that people with the standard vision are able to see. Dichromacy is the most common type of color blindness and it is possible to split it into three separate generalized deficiencies: protanopia, deuteranopia, and tritanopia. People suffering from the first deficiency are less sensitive to red light, while the ones suffering from deuteranopia have issues with green light. Nevertheless, both are having problems with red and green hues. People affected by tritanopia confuse blue with green and yellow with violet. Let's see how these issues compare to the normal vision.

```
# Color blindness ####
library(grid)
library(gridExtra)
library(dichromat)
data("dalton")
df <- data.frame(v1=rep(1:2, each=128),v2=1:128, v3=1:256)
gg <- ggplot(df, aes(v2,v1, fill=factor(v3))) +
  geom_raster(hjust = 0, vjust = 0) +
  theme(legend.position ="none",
        axis.text.x = element_blank(),
        axis.text.y = element_blank(),
```

```
        axis.ticks = element_blank()) +
  xlab("")+
  ylab("")
gg.1 <- gg +
  scale_fill_manual(values = dalton.colors$normal)+
  ggtitle("Normal vision")
gg.2 <- gg +
  scale_fill_manual(values = dalton.colors$deutan)+
  ggtitle("Deutan vision")
gg.3 <- gg +
  scale_fill_manual(values = dalton.colors$protan)+
  ggtitle("Protan vision")
gg.4 <- gg +
  scale_fill_manual(values = dalton.colors$tritan)+
  ggtitle("Tritan vision")
grid.arrange(arrangeGrob(grobs=list(gg.1, gg.2,gg.3,gg.4), nrow=4, ncol=1,
top=""))
```



6. References

6.1. Function reference

abline {graphics}..... adds one or more straight lines through the current plot

acf {stats}.....computes estimates of the autocovariance or autocorrelation function of a given time series

Acf {forecast}..... computes estimates of the (partial) autocorrelation function of a given time series

aes {ggplot2} .....creates a list of unevaluated expressions in ggplot

apply {base}..... returns a vector or an array or a list of values obtained by applying a function

arima {stats}.....fits an ARIMA model to a univariate time series

arima.sim {stats} .....simulates an ARIMA model

arrangeGrob {gridExtra}..... produces plots with combined graphs

array {base}.....creates or tests for arrays

as.data.frame {base}.....coerces its argument to an object of type “data frame”

as.factor {base} .....coerces its argument to a factor

as.numeric {base}..... coerces its argument to a an object of type “numeric”

attach {base} ..... attaches the databases to the R environment

attr {base}.....gets or sets specific attributes of an object

auto.arima {forecast}..... returns the best ARIMA model according to a lag selection criterion

autoplot {ggplot2}.....draws a plot based on the class of an object

boxplot {graphics}..... produces box-and-whisker plots of the given (grouped) values

c {base}..... combines values into a vector or a list

cat {base}..... outputs the objects and concatenates the representations

cbind {base}..... combines R Objects by columns

chisq.test {stats}.....performs chi-squared contingency table tests and goodness-of-fit tests

class {base}..... prints the vector of names of classes an object inherits from

coef {stats} ..... extracts model coefficients from objects returned by modeling functions

colMeans {base}..... forms column means for numeric arrays

colnames {base}..... retrieves or sets the column names of a matrix-like object

confint {stats}.....computes confidence intervals for one or more parameters in a fitted model

cos {base}.....computes the cosine

curve {graphics}.....draws a curve corresponding to a function over an interval

cut {base} .....divides and codes the values in x according to which interval they fall

cut2 {Hmisc}.....extended version of cut()

data {utils}..... loads specified data sets, or lists the available data sets

data.frame {base}..... creates data frames

ddply {plyr}.....applies a function and combines results into a data frame for each subset

diff {base}..... returns suitably lagged and iterated differences

dimnames {base}.....retrieves or sets the dimnames of an object

dnorm {stats} ..... gives the density for the normal distribution

duplicated {base} .....determines duplicated elements and returns a logical vector

effects {stats} ..... returns (orthogonal) effects from a fitted model, usually a linear model

exp {base}..... computes the exponential function

facet_grid {ggplot2} ..... factorizes the plots by a given variable

factor {base}.....returns an object of class "factor"

file.choose {base}.....choose a file interactively

filter {stats} .....applies linear filtering to a time series or to a multivariate time series searately

fitted {stats} .....extracts fitted values from objects returned by modeling functions

getwd {base} ..... returns a path of the current working directory

geom_bar {ggplot2} ..... produces bar charts for categorical x, and histograms for continuous y of a ggplot

geom_errorbar {ggplot2}..... draws error bars in ggplot2

geom_hline {ggplot2}.....draws horizontal lines in ggplot2

geom_line {ggplot2}.....connects observations in ggplot

geom_point {ggplot2}..... creates scatterplots in ggplot

geom_raster {ggplot2}.....draws rectangles in ggplot2

geom_smooth {ggplot2}.....adds a smooth mean to the ggplot2

ggplot {ggplot2}..... initializes a ggplot object

ggtitle {ggplot2}..... changes labels of axes and titles in ggplot2

grid.arrange {gridExtra}..... sets up a gtable layout to place multiple ggplot graphs on a page

head {utils}..... returns the first parts of a vector, matrix, table, data frame or a function

hist {graphics}.....computes a histogram of the given data values

hmftest {mlogit}..... tests the IIA hypothesis for a multinomial logit model

HoltWinters {stats}.....computes Holt-Winters filtering of a given time series

is.list {base}..... checks if its argument is a list

labs {ggplot2}.....changes the axis labels and the legend title of a ggplot object

lapply {base}.....applies and a function for each corresponding element of a list or a vector

layout {graphics}..... divides the device up into as many rows and columns as there are in matrix

length {base}.....gets or sets the length of an R object for which a method has been defined

levels {base}..... provides access to the levels attribute of a variable

library {base}..... loads add-on packages

lines {graphics}.....joins the corresponding points with line segments

list {base}.....constructs a list

lm {stats}..... fits linear models

log {base}.....computes logarithms (by default natural logarithms)

ls {base}.....gives the names of the objects in the specified environment

matrix {base}..... constructs a matrix

max {base}..... returns the maxima of the input values

mean {base} ..... returns the arithmetic mean of the input values

median {stats}..... computes the sample median

min {base} .....returns the minima of the input values

mlogit {mlogit} ..... estimates the multinomial logit model

mlogit.data {mlogit}..... shapes a data.frame in a suitable form for the use of the mlogit function

nrow {base}..... return the number of rows present in a vector, array or data frame

numeric {base} .....creates objects of type “numeric”

options {base} .....sets a variety of global options to compute and display the results

pacf {stats} ..... computes estimates of the partial autocorrelation function

Pacf {forecast}.....computes an estimate of the partial autocorrelation function of a univariate time series

par {graphics}..... sets or queries graphical parameters

paste {base}..... concatenates vectors after converting to character

pchisq {stats}.....gives the distribution function of Chi-Squared distribution

plot {graphics}.....plots R objects

pnorm {stats} ..... gives the distribution function for the normal distribution

polygon {graphics}.....draw polygons manually with predefined coordinates

position_dodge {ggplot2}..... adjusts position by dodging overlaps to the side in ggplot2

predict {stats}.....predicts from the results of various model fitting functions

print {base}..... prints its argument and returns it

prop.table {base}..... prints out the proportions of each cell of an array

qnorm {stats} .....gives the quantile function for the normal distribution

qplot {ggplot2} ..... allows to use the syntax of plot() in the ggplot2 environment

reorder {stats}.....reorders its levels based on the values of the second variable

rbind {base}.....combines R Objects by rows

rbinom {stats}..... generates random deviates for the binomial distribution

read.csv {utils}..... reads a file in CSV format and creates a data frame from it

read.octave {foreign} ..... reads a MATLAB/Octave file into a data frame

read.spss {foreign} ..... reads an SPSS file into a data frame

read.stata {foreign}..... reads a STATA file into a data frame

read.xls {gdata} ..... reads an excel file into a data frame

read.xlsx {xlsx}..... reads an excel file into a data frame

read.table {utils}..... reads a file in table format and creates a data frame from it

remove *OR* rm {base}..... removes a given object from working environment in R

reorder {stats}..... reorder factors and other objects

rep {base}..... replicates the values

replicate {base} .....*see sapply {base}*

return {base} ..... returns a given value

rnorm {stats} .....gives random deviates for the normal distribution

row.names {base} .....changes the names of rows in a given object

round {base}..... rounds the values in its first argument to the specified number of decimal places

sample {base}..... gives a random sample of the specified size

sapply {base}..... a user-friendly version and wrapper of lapply()

scale {base} .....scales the columns of a given matrix

scale_y_log10 {ggplot2}.....logarithmizes the y-scale of ggplot

sd {stats}.....computes the standard deviation of a given array

setRepositories() {utils}..... interacts with the user to choose the package repositories to be used

seq {base}.....generates regular sequences

setwd() {base} ..... sets a given path as the new working directory

shapiro.test {stats}..... performs the Shapiro-Wilk test of normality

sin {base} ..... computes the sine

scale_colour_gradient {ggplot2} ..... creates gradient transition for a given color

scale_colour_hue {ggplot2} ..... creates gradient colours to different variables in ggplot2

scale_fill_gradient {ggplot2} ..... creates gradient color transitions for given colors

scale_x_continuous {ggplot2} ..... sets the continuous position *x* scale

scale_x_datetime {ggplot} ..... scales the time scale

scale_y_continuous {ggplot2} ..... sets the continuous position *y* scale

solve {base} ..... solves the equation $ax = b$ for x; calculates the inverse of a matrix

split {base} ..... split the data into subgroups

sqrt {base} ..... computes the square root

stat.desc {pastecs} ..... compute a table giving various descriptive statistics of the given data

stat_smooth {ggplot2} ..... helps to distinguish the objects in case of heavy plotting

stl {stats} ..... decomposes a time series into seasonal, trend and irregular components

str {utils} ..... compactly display the internal structure of an R object

subset {base} ..... returns subsets of vectors, matrices or data frames

sum {base} ..... returns the sum of all the values present in its arguments

summary {base} ..... produces result summaries of the results of various model fitting functions

svm {e1071} ..... trains support vector machine

t {base} ..... transposes a matrix or a data frame

t.test {stats} ..... performs one and two sample t-tests on vectors of data

table {base} ..... builds a contingency table of the counts at each combination of factor levels

tan {base} ..... calculates the tangent

tapply {base} ..... applies a function to each cell of a ragged array

theme {ggplot2} ..... works with generic themes of ggplot2

theme_grey {ggplot2} ..... sets the general aspect of a ggplot, check *ggtheme* for more themes

ts {stats} ..... creates time-series objects

tune {e1071} ..... tunes the parameters of a function, for exampe svm()

unit {grid} ..... creates a unit vector

unique {base} ..... removes duplicate elements in a vector

unlist {base}..... converts a list into a vector

View {utils} ..... invokes a spreadsheet-style data viewer on a matrix-like R object

which {base}..... gives the `TRUE` indices of a logical object

with {base} ..... evaluates an R expression in an environment constructed from data

xlim {ggplot2} ..... limits the range of observation to the given range on x axis of a ggplot

ylim {ggplot2} ..... adds and controls the label of y-axis in ggplot2

xlab {ggplot2} ..... adds and controls the label of x-axis in ggplot2

ylab {ggplot2} ..... changes the label of y-axis in a ggplot object

6.2. Literature reference

Chang, W. 2012. R Graphics Cookbook. O'Reilly Media.

Hastie, T., James, G., Tibshirani, R., Witten, D. 2015. An Introduction to Statistical Learning with Applications in R. Springer Texts in Statistics.

Provost, F. and Fawcett, T. 2013. Data Science for Business: What you need to know about data mining and data-analytic thinking. O'Reilly Media.