Quantitative Methods

PART 2: An introduction to statistics with R

Prof. Dr. Philippe J.S. De Brouwer

2016–2017,
Warsaw, Poland

last compiled: June 25, 2017

executive master of business administration
emba@uw

Contents

PART I

An Introduction to statistics with R

PART I .:. CHAPTER 1 .:.

The Basics of R

R is . . .

- a programming language build for statistical analysis, graphics representation and reporting.
- an interpreted computer language which allows branching, looping, modular programming using functions as well as object oriented programming features.

The main features of R

- allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.
- is freely (under the GNU General Public License), and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.
- simple and effective
- free and open
- has an effective data handling and storage facility,
- provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- provides a large, coherent and integrated collection of tools for data analysis.
- provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.
- allows you to stand on the shoulders of giants
- has a supportive on-line community

R is the most widely used statistics programming language and is used from universities to business applications

- You need a working installation of R on your computer.

- R is available for Mac, Linux and Windows from `https://cran.r-project.org/`
- To start R, open the command line and type `R` (followed by enter)
- You should then get the command line prompt of R. It is of course also possible to use a graphical interface such as RStudio (see `https://www.rstudio.com`)

alternative

Use R online:

- `https://www.tutorialspoint.com/execute_r_online.php`
- `http://www.r-fiddle.org`

Basic arithmetic

```
#addition
2+3
#product
2*3
#power
2**3
2^3
#logic
2< 3
x <- c(1,3,4,3)
x.mean <- mean(x)
x.mean
y <- c(2,3,5,1)
x+y
```

the `scan()` function

```
x <- scan()
```

invites you to type all values of the vector one by one.
In order to end: type enter without typing a number (ie. leave one empty to end).

Batch mode

1. create a file `test.R`
2. add the content `print("Hello World")`

3. run the command line `Rscript test.R`
4. now, open R and run the command `source("test.R")`
5. add in the file

```
my_function <- function(a,b)
{
  a +  b
}
```

6. now repeat step 4 and run `my_function(4,5)`

1.1 Variables

Valid Variables

Variables

- can contain letters as well as "_" (underscore) and "." (dot)
- variables must start with a letter (that can be preceded with a dot

eg. my_var.1 and my.Cvar. are valid, but _myVar, my%var and 1.var are not

Variable assignment

Assignment can be made left or right:

```
x.1 <- 5
x.1 + 3 -> .x
print(.x)

## [1] 8
```

Useful functions for variables

```
# List all variables
ls()                    # variables starting with dot are hidden
ls(all.names = TRUE)    # shows all

# Remove a variable
rm(x.1)                 # removes the variable x.1
ls()                    # x.1 is not there any more
rm(list = ls())         # removes all variables
ls()
```

1.2 Data Types

1.2.1 Basic Data Types

Basic Data Types

Variables are not declared as a data type, rather they are assigned a class object

```
x <- TRUE
class(x)

## [1] "logical"
```

```
x <- 5L
class(x)

## [1] "integer"
```

```
x <- 5.135
class(x)

## [1] "numeric"
```

```
x <- 2.2 + 3.2i
class(x)

## [1] "complex"
```

```
x <- "test"
class(x)

## [1] "character"
```

1.2.2 Vectors

Composed Data Types

Vectors vs Lists

Vectors are lists of objects of the same type. vectors are declared with the function `c()`

```
x <- c(2,2.5,4,6)
y <- c("apple", "pear")
class(x)

## [1] "numeric"

class(y)

## [1] "character"
```

a list of objects of different types is called a "list".

```
# Create a list.
list1 <- list(c(1,2,3),3.1415,sin)

# Print the list.
print(list1)

## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 3.1415
##
## [[3]]
## function (x)  .Primitive("sin")
```

Accessing Elements of a vector

```
v <- c(1:5)
# access elements via indexing
v[2]

## [1] 2

v[c(1,5)]

## [1] 1 5

#via TRUE/FALSE:
v[c(TRUE,TRUE,FALSE,FALSE,TRUE)]

## [1] 1 2 5

#leave out certain elements:
v[c(-2,-3)]

## [1] 1 4 5
```

Vector arithmetic

All standard behaviour is element per element

```
v1 <- c(1,2,3)
v2 <- c(4,5,6)
# Standard arithmetic
v1 + v2

## [1] 5 7 9

v1 - v2

## [1] -3 -3 -3

v1 * v2

## [1]  4 10 18
```

```
v1 / v2

## [1] 0.25 0.40 0.50

# Vector Recycling
v1 <- c(1,2,3,4)
v2 <- c(1,2)
v1+v2

## [1] 2 4 4 6

# because v2 became (1,2,1,2)
```

Vector sorting

```
v1 <- c(1,4,2,5,6,3,2,99,0,pi)
sort(v1)

##  [1]  0.000000  1.000000  2.000000  2.000000  3.000000  3.141593  4.00
##  [8]  5.000000  6.000000 99.000000

v2 <- c("January", "February", "March", "April")
sort(v2)

## [1] "April"    "February" "January"  "March"

sort(v2, decreasing=TRUE)

## [1] "March"    "January"  "February" "April"
```

Exercise: S&P500

Question

The time series nottem (from the package "datasets" that is usually loaded when R starts) contains the temperatures in Notthingham from 1920 to 1939 in Farenheit. Create a new object that contains a list of all temperatures in Celcius.

Note that nottem is a time-series object (see: Chapter 2.8 on page 102) and not a matrix. All its elements are addressed with nottam[n] where n is between 1 and length(nottam). Remember that $T(C) = \frac{5}{9}(T(F) - 32)$.

1.2.3 Lists

Lists: definition

Definition 1 .:. List .:.

In R lists are objects which contain elements of different types (they can mix numbers, strings, vectors, matrices, functions, boolean variables and even lists).

```
# List is created using list() function.
myList <- list("Approximation", pi, 3.14, c)
print(myList)

## [[1]]
## [1] "Approximation"
##
## [[2]]
## [1] 3.141593
##
## [[3]]
## [1] 3.14
##
## [[4]]
## function (...)  .Primitive("c")
```

Naming elements of lists

```
# create the list
L <- list("Approximation", pi, 3.14, c)

# assign names to elements
names(L) <- c("description", "exact", "approx","function")
print(L)

#addressing elements of the named list
print(paste("The difference is", L$exact - L$approx))
print(L[3])
print(L$approx)

# however "function" was a reserved word: bad idea, but possible:
a <- L$`function`(2,3,pi,5)
print(a)
```

Merged lists are also lists

```
V1 <- c(1,2,3)
L2 <- list(V1, c(2:7))
L3 <- list(L2,V1)
print(L3)

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

print(L3[[1]][[2]][3])

## [1] 4
```

Add and delete list elements

```
L <- list("mystring", matrix(c(1,2,3,4),nrow=2))
```

```
#add an element
L <- list(L, c(1:10))

#delete an element
L[1] <- NULL
print(L[1])

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

print(L[2])

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

Convert list to vectors

```
L <- list(c(1:5),c(6:10))
v1 <- unlist(L[1])
v2 <- unlist(L[2])
v2-v1

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

1.2.4 Matrics

Matrices

A matrix is a two-dimensional data set. The `matrix()` function offers a convenient way to define it:

```
# Create a matrix.
M = matrix( c(1:6), nrow = 2, ncol = 3, byrow = TRUE)
print(M)

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

```
M = matrix( c(1:6), nrow = 2, ncol = 3, byrow = FALSE)
print(M)

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

Naming rows and columns

```
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
M <- matrix(c(10:21), nrow = 4, byrow = TRUE, dimnames = list(rownames, c
print(M)

##      col1 col2 col3
## row1   10   11   12
## row2   13   14   15
## row3   16   17   18
## row4   19   20   21
```

Accessing data in a Matrix

```
M <- matrix(c(10:21), nrow = 4, byrow = TRUE)

#access one element
M[1,2]

## [1] 11

# second row
M[2,]

## [1] 13 14 15

# second column
M[,2]

## [1] 11 14 17 20
```

Matrix Arithmetic

works element by element

```
M1 <- matrix(c(10:21), nrow = 4, byrow = TRUE)
M2 <- matrix(c(0:11), nrow = 4, byrow = TRUE)
M1+M2

##      [,1] [,2] [,3]
## [1,]   10   12   14
## [2,]   16   18   20
## [3,]   22   24   26
## [4,]   28   30   32

M1*M2

##      [,1] [,2] [,3]
## [1,]    0   11   24
## [2,]   39   56   75
## [3,]   96  119  144
## [4,]  171  200  231

M1/M2

##          [,1]      [,2]     [,3]
## [1,]      Inf 11.000000 6.000000
## [2,] 4.333333  3.500000 3.000000
## [3,] 2.666667  2.428571 2.250000
## [4,] 2.111111  2.000000 1.909091
```

Exercise: crossproduct

Question

Write a function for the dot-product for matrices. Add also some security checks. Finally compare your results with the "%*%-operator" (or the function `crossproduct()`

```
#Example of the results that you should find:
a <- c(1:3)
a %*% a

##      [,1]
## [1,]   14

a %*% t(a)

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    4    6
## [3,]    3    6    9
```

```
t(a) %*% a

##      [,1]
## [1,]   14

A <- matrix(1:9,nrow=3)
A %*% a

##      [,1]
## [1,]   30
## [2,]   36
## [3,]   42

A %*% t(a) # this is bound to fail!

## Error in A %*% t(a):  non-conformable arguments

A %*% A

##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

.....

.....

.....

.....

1.2.5 Arrays

Creating and accessing arrays

Arrays can be of any number of dimensions (matrices are always 2-dimensional), but need to be of one data type. They can be created with the `array()` function; this function takes a "dim" attribute which defines the number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('A','B'),dim = c(3,3,2))
print(a)

#access one element
```

```
a[2,2,2]

#access one layer
a[,,2]
```

Naming array elements

```
# Create two vectors
v1 <- c(1,1)
v2 <- c(10:13)
row.names <- c("col1","col2")
col.names <- c("R1","R2","R3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
a <- array(c(v1,v2),dim = c(2,3,2),
     dimnames = list(row.names,col.names,
     matrix.names))
print(a)
```

Manipulating arrays

```
M1 <- a[,,1]
M2 <- a[,,2]
M2

##      R1 R2 R3
## col1  1 10 12
## col2  1 11 13
```

Applying functions over arrays

An efficient way to apply the same function over an array is the function `apply`

Definition 2

`apply(X, MARGIN, FUN, ...)` with:

1. X: an array, including a matrix.
2. MARGIN: a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.
3. FUN: the function to be applied: see Details. In the case of functions like +, backquoted or quoted

An example for apply()

```
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
dimnames(x)[[1]] <- letters[1:8]
apply(x, 2, mean, trim = .2)

## x1 x2
##  3  3

col.sums <- apply(x, 2, sum)
row.sums <- apply(x, 1, sum)
rbind(cbind(x, Rtot = row.sums), Ctot = c(col.sums, sum(col.sums)))

##      x1 x2 Rtot
## a     3  4    7
## b     3  3    6
## c     3  2    5
## d     3  1    4
## e     3  2    5
## f     3  3    6
## g     3  4    7
## h     3  5    8
## Ctot 24 24   48
```

1.2.6 Factors

Factors

Factors are the r-objects which hold a series of labels. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of data type of the elements in the input vector. They are useful in statistical modelling.

Factors are created using the `factor()` function.

```
# Create a vector containing all your observations.
feedback <- c('good','good','bad','average','bad','good','good')

# Create a factor object.
factor_feedback <- factor(feedback)

# Print the factor object.
print(factor_feedback)

## [1] good    good    bad     average bad     good    good
## Levels: average bad good
```

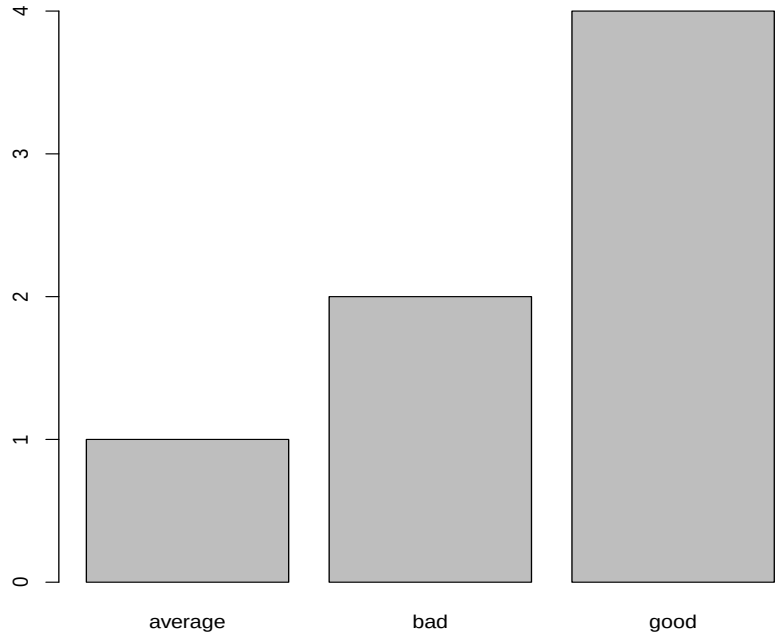


Figure 1.1: *Factors*

```
# Plot the histogram -- note the default order is alphabetic
plot(factor_feedback)
```

```
# The {\tt nlevels} functions gives the number of levels.
print(nlevels(factor_feedback))

## [1] 3
```

Ordering the factors

```
feedback <- c('good','good','bad','average','bad','good','good')
factor_feedback <- factor(feedback,levels=c("bad","average","good"))
plot(factor_feedback)
```

The function gl() can generate factors

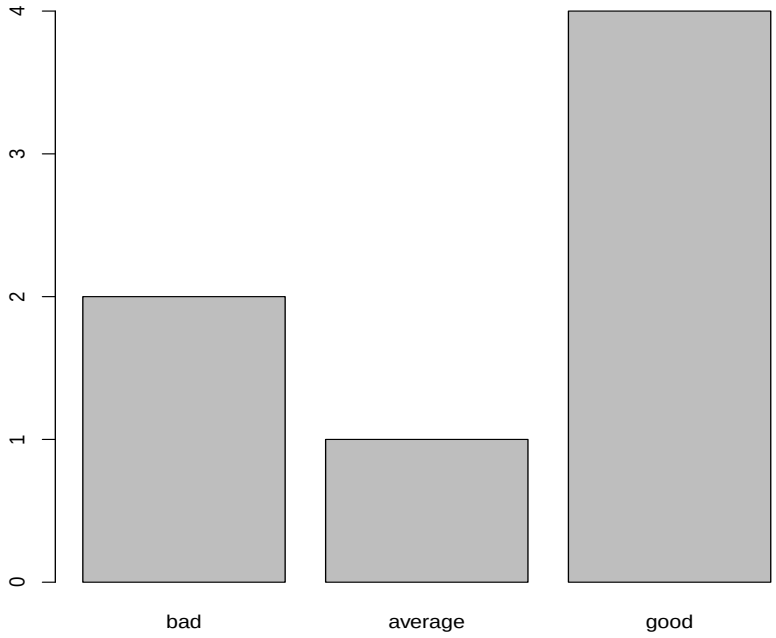


Figure 1.2: *Ordered factors*

Definition 3 .:. gl() .:.

`gl(n, k, length = n*k, labels = seq_len(n), ordered = FALSE)`
with

- n: the number of levels
- k: the number of replications (for each level)
- length (optional): an integer giving the length of the result
- labels (optional): a vector with the labels
- ordered: a boolean variable indicating whether the results should be ordered

```
gl(3,2,,c("bad","average","good"),TRUE)

## [1] bad     bad     average average good    good
## Levels: bad < average < good
```

Exercise: cars

Question

Use the dataset mtcars (from the library MASS) and explore the distribution of number of gears. Then focus on the transmission and create a factor-object with the words "automatic" and "manual" in stead of the number 0 and 1.

Use the `?mtcars` to find out the exact definition of the data.

.....

.....

.....

.....

Exercise: cars-horsepower

Question

Use the dataset mtcars (from the library MASS) and explore the distribution of the horsepower (hp). How would you proceed to make a factoring (eg. Low, Medium, High) for this attribute?

.....

.....

.....

.....

1.2.7 Data Frames

Data Frames

Data frames are very useful for statistical modelling; they are objects that contain data in a tabular way. Unlike a matrix in data frame each column can contain different modes of data. For example, the first column can be factorial, the second logical and the third numerical.

It is a composed data type consisting of a list of vectors of equal length.

Data Frames are created using the `data.frame()` function.

```
# Create the data frame.
test.data <- data.frame(
   name = c("Piotr", "Pawel","Paula","Lisa","Laura"),
   gender = c("Male", "Male","Female", "Female","Female"),
```

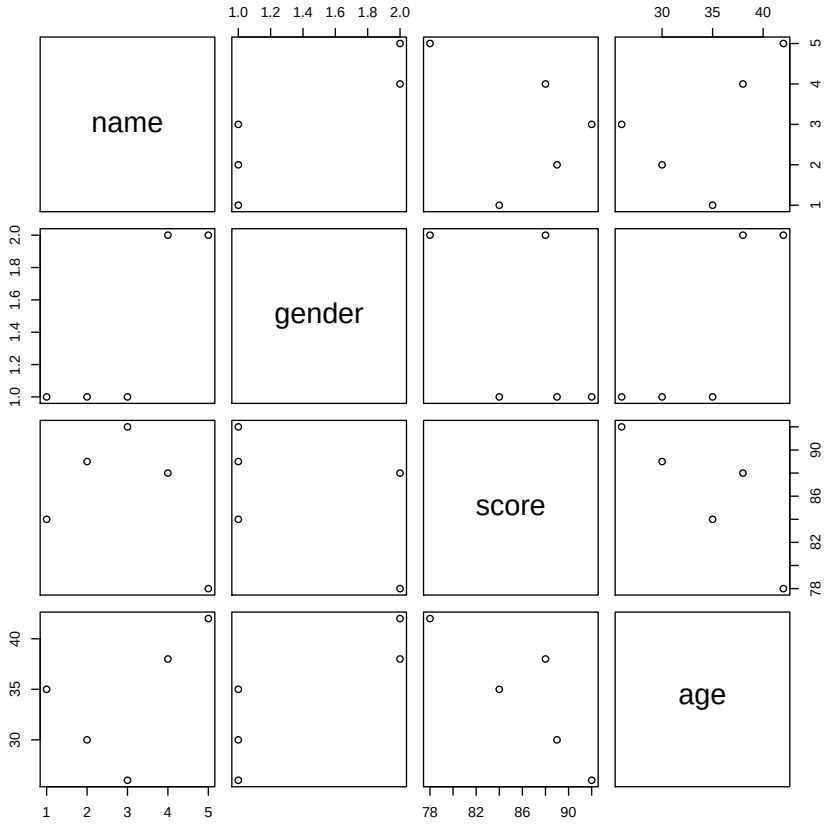


Figure 1.3: *The standard plot for a data frame in R.*

```
   score = c(78,88,92,89,84),
   age = c(42,38,26,30,35)
)
print(test.data)

##    name gender score age
## 1 Piotr   Male    78  42
## 2 Pawel   Male    88  38
## 3 Paula Female    92  26
## 4  Lisa Female    89  30
## 5 Laura Female    84  35

# the standard plot function on a data-frame ...
plot(test.data)
```

```
# ... coincides with the pairs() function
# pairs(test.data)
```

Edit data in data-frames

```
de(x)                    # fails if x is not defined
de(x <- c(NA))           # works
x <- de(x <- c(NA))      # will also save the changes
data.entry(x)            # de is short for data.entry
x <- edit(x)             # use the standard editor (vi in *nix)
```

Get information about data frames

```
# Get the structure of the data frame:
str(test.data)

## 'data.frame': 5 obs. of  4 variables:
##  $ name  : Factor w/ 5 levels "Laura","Lisa",..: 5 4 3 2 1
##  $ gender: Factor w/ 2 levels "Female","Male": 2 2 1 1 1
##  $ score : num  78 88 92 89 84
##  $ age   : num  42 38 26 30 35

# Get the summary of the data frame:
summary(test.data)

##     name      gender      score           age
##  Laura:1   Female:3   Min.   :78.0   Min.   :26.0
##  Lisa :1   Male  :2   1st Qu.:84.0   1st Qu.:30.0
##  Paula:1              Median :88.0   Median :35.0
##  Pawel:1              Mean   :86.2   Mean   :34.2
##  Piotr:1              3rd Qu.:89.0   3rd Qu.:38.0
##                       Max.   :92.0   Max.   :42.0

# Get the first rows:
head(test.data)

##    name gender score age
## 1 Piotr   Male    78  42
## 2 Pawel   Male    88  38
## 3 Paula Female    92  26
## 4  Lisa Female    89  30
## 5 Laura Female    84  35

# Get the last rows:
tail(test.data)

##    name gender score age
## 1 Piotr   Male    78  42
## 2 Pawel   Male    88  38
## 3 Paula Female    92  26
## 4  Lisa Female    89  30
## 5 Laura Female    84  35
```

```
# Extract the column 2 and 4 and keep all rows
test.data.1 <- test.data[,c(2,4)]
print(test.data.1)

##   gender age
## 1   Male  42
## 2   Male  38
## 3 Female  26
## 4 Female  30
## 5 Female  35
```

Add columns to a data-frame

```
# Expand the data frame, simply define the additional column
test.data$end_date <-  as.Date(c("2014-03-01", "2017-02-13",
                       "2014-10-10", "2015-05-10","2010-08-25"))
print(test.data)

##    name gender score age   end_date
## 1 Piotr   Male    78  42 2014-03-01
## 2 Pawel   Male    88  38 2017-02-13
## 3 Paula Female    92  26 2014-10-10
## 4  Lisa Female    89  30 2015-05-10
## 5 Laura Female    84  35 2010-08-25

# Or use the function cbind()
a <- c (1:5)
b <- c (11:15)
c <- c (111:115)
df <- cbind(a,b,c)
print(df)

##      a  b   c
## [1,] 1 11 111
## [2,] 2 12 112
## [3,] 3 13 113
## [4,] 4 14 114
## [5,] 5 15 115
```

Add new data (rows) to a data frame

```
# To add a row, we need the rbind() function
test.data.to.add <- data.frame(
   name = c("Ricardo", "Anna"),
```

```
   gender = c("Male", "Female"),
   score = c(66,80),
   age = c(70,36),
   end_date = as.Date(c("2016-05-05","2016-07-07"))
   )
test.data.new <- rbind(test.data,test.data.to.add)
print(test.data.new)

##      name gender score age   end_date
## 1   Piotr   Male    78  42 2014-03-01
## 2   Pawel   Male    88  38 2017-02-13
## 3   Paula Female    92  26 2014-10-10
## 4    Lisa Female    89  30 2015-05-10
## 5   Laura Female    84  35 2010-08-25
## 6 Ricardo   Male    66  70 2016-05-05
## 7    Anna Female    80  36 2016-07-07
```

Merging data frames

Merging allows to extract the subset of 2 data-frames where a given set of columns match

```
test.data.1 <- data.frame(
   name = c("Piotr", "Pawel","Paula","Lisa","Laura"),
   gender = c("Male", "Male","Female", "Female","Female"),
   score = c(78,88,92,89,84),
   age = c(42,38,26,30,35)
)
test.data.2 <- data.frame(
   name = c("Piotr", "Pawel","notPaula","notLisa","Laura"),
   gender = c("Male", "Male","Female", "Female","Female"),
   score = c(78,88,92,89,84),
   age = c(42,38,26,30,135)
)
test.data.merged <- merge(x=test.data.1,y=test.data.2,by.x=c("name","age"
print(test.data.merged)

##    name age gender.x score.x gender.y score.y
## 1 Pawel  38     Male      88     Male      88
## 2 Piotr  42     Male      78     Male      78
```

Shortcuts

R will allow shortcuts provided they're unique.

```
test.data$n

## [1] Piotr Pawel Paula Lisa  Laura
## Levels: Laura Lisa Paula Pawel Piotr
```

Exercise: data-frames

Question

1. create a matrix a 3 by 3 matrix with the numbers 1 to 9,
2. convert it to a data-frame,
3. add names for the columns and rows,
4. add a column with the column-totals
5. drop the second column
6. plot the first row in function of the second row

1.3 Operators

1.3.1 Arithmetic Operators

Arithmetic operators act on each element of an object

```
v1 <- c(2,4,6,8)
v2 <- c(1,2,3,5)
v1 + v2      # addition

## [1]  3  6  9 13

v1 - v2      # subtraction

## [1] 1 2 3 3

v1 * v2      # multiplication

## [1]  2  8 18 40

v1 / v2      # division

## [1] 2.0 2.0 2.0 1.6

v1 %% v2     # remainder of division

## [1] 0 0 0 3

v1 %/% v2    # round(v1/v2 -0.5)

## [1] 2 2 2 1

v1 ^ v2      # v1 to the power of v2

## [1]     2    16   216 32768
```

1.3.2 Relational Operators

Logical Operators compare vectors element by element

```
v1 <- c(8,6,3,2)
v2 <- c(1,2,3,5)
v1 > v2     # bigger than

## [1]  TRUE  TRUE FALSE FALSE

v1 < v2     # smaller than

## [1] FALSE FALSE FALSE  TRUE

v1 <= v2    # smaller or equal

## [1] FALSE FALSE  TRUE  TRUE

v1 >= v2    # bigger or equal

## [1]  TRUE  TRUE  TRUE FALSE

v1 == v2    # equal

## [1] FALSE FALSE  TRUE FALSE

v1 != v2    # not equal

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

1.3.3 Logical Operators

Logical Operators combine vectors element by element

They only are possible on numeric, logical or complex types

```
v1 <- c(TRUE,FALSE,TRUE,FALSE,8,6+3i)
v2 <- c(FALSE,TRUE,TRUE,FALSE,8,6)
v1 & v2     # and

## [1] FALSE FALSE  TRUE FALSE  TRUE  TRUE

v1 | v2     # or

## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE

!v1         # not

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

```
v1 && v2    # and applied to the first element

## [1] FALSE

v1 || v2    # or applied to the first element

## [1] TRUE
```

1.3.4 Assignment Operators

Assignment operators are left or right

```
# left assignment
x <- 3
x = 3
x <<- 3

# right assignment
3 -> x
3 ->> x

#chained assignment
x <- y <- 4
```

1.3.5 Other Operators

Various operators

```
# : creates a list
2:6

## [1] 2 3 4 5 6

x <- c(1:10)
x

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

# %in% can find an element in a vector
2  %in% x

## [1] TRUE

11 %in% x

## [1] FALSE
```

```
# %*% the matrix multiplication (or crossproduct)
M = matrix(c(1,2,3,7,8,9,4,5,6), nrow = 3,ncol = 3,
           byrow = TRUE)
M %*% t(M)

##      [,1] [,2] [,3]
## [1,]   14   50   32
## [2,]   50  194  122
## [3,]   32  122   77

M %*% M

##      [,1] [,2] [,3]
## [1,]   27   33   39
## [2,]   99  123  147
## [3,]   63   78   93

exp(M)

##              [,1]        [,2]       [,3]
## [1,]    2.718282    7.389056   20.08554
## [2,] 1096.633158 2980.957987 8103.08393
## [3,]   54.598150  148.413159  403.42879
```

1.3.6 Loops

For

```
for (value in vector) {
   statements
}
```

will execute the statements for each value in the given vector. eg.

```
x <- LETTERS[1:5]
for ( j in x) {
   print(j)
}

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

Repeat

```
repeat {
   commands
   if(condition) {break}
}
```

example:

```
x <- c(1,2)
c <- 2
repeat {
   print(x+c)
   c <- c+1
   if(c > 4) {break}
}

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

While

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

The statements are executed as long the test_expression is true.

```
x <- c(1,2); c <- 2
while (c < 4) {
   print(x+c)
   c <- c + 1
}

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

Loop control statements

Break

The `break` statement in R programming language has the following two usages:

- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

- It can be used to terminate a case in the switch statement (covered in the next chapter).

```
v <- c(1:5)
for ( j in v) {
   if (j == 3) {
      print("--break--")
      break
   }
   print(j)
}

## [1] 1
## [1] 2
## [1] "--break--"
```

Loop control statements

Break

The `next` statement will skip the remainder of the current iteration of a loop and starts next iteration of the loop.

```
v <- c(1:5)
for ( j in v) {
   if (j == 3) {
      print("--skip--")
      next
   }
   print(j)
}

## [1] 1
## [1] 2
## [1] "--skip--"
## [1] 4
## [1] 5
```

1.3.7 Functions

Built-in function

Standard Functionality

Examples:

- `q()`: quits R

- `data()`: shows the data-sets available
- `help()`: shows help
- `ls()`: shows variables
- `c()`: creates a vector
- `seq()`: creates a sequence
- `mean()`: calculates the mean
- `max()`: returns the maximum
- `sum()`: returns the sum
- `paste()`: concatenates vector elements

Help with functions

```
help(c)         #help with functions

apropos("cov") #fuzzy search for functions
```

Creating a function

User defined functions

```
function_name <- function(arg_1, arg_2, ...) {
   function_body
   return_value
}
```

Example 1

```
c.surface <- function(radius) {
  x <- radius ^ 2 * pi
  return (x)
  }
c.surface(2) + 2

## [1] 14.56637
```

Note that it is not necessary to explicitly "return" something. A function will automatically return the last value that is send to the standard output. So, the following fragment would do exactly the same:

```
c.surface <- function(radius) {
  radius ^ 2 * pi
  }
c.surface(2) + 2

## [1] 14.56637
```

Editing functions in R

```
# edit the function with vi
 fix(c.surface)

# or
 c.surface <- edit()
```

Function with a default argument

Example 2

what is the default separator used in `paste()`?

Creating functions with a default value

```
c.surface <- function(radius = 2) {
  radius ^ 2 * pi
  }
c.surface(1)

## [1] 3.141593

c.surface()

## [1] 12.56637
```

Exercise: use functions

Question

calculate the mean of all numbers from 1000 to 1500

.....

.....

.....

.....

Exercise: create functions

Question

Create a function that takes a matrix as input, and calculates the sum of the rows and returns this as a vector.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

1.3.8 Packages

The package system

Additional functions come in "packages". To use them one needs

```
#download the package (only once):
install.packages('DiagrammeR')

#Then load it each session it is used
library(DiagrammeR)
```

Examples of packages

- **To load data**
  - `RODBC, RMySQL, RPostgresSQL, RSQLite`: read data from a database
  - `XLConnect, xlsx`: read and write Micorsoft Excel files (of course You can also just export your spreadsheets from Excel as csv-files)
  - `foreign`: to use eg SAS data
  - `Note`: among R's standard functionality is handling text files. Use the functions read.csv, read.table, and read.fwf.
- **To manipulate data**
  - `dplyr`: creating subsets, summarizing, rearranging, and joining data sets
  - `tidyr`: changing the layout of your data sets
  - `stringr`: tools for regular expressions and character strings
  - `lubridate`: tools to facilitate working with dates and times
  - `reshape`: tools to present data differently (melt and cast)

- **To visualize data**
  - `ggplot2`: allows professional graphics (and has many extensions)
  - `ggvis`: to build interactive, web based graphics
  - `rgl`: Interactive 3D visualizations with R
  - `htmlwidgets`: build interactive (javascript based) visualizations. Packages that implement htmlwidgets include: leaflet (maps), dygraphs (time series), DT (tables), diagrammeR (diagrams), network3D (network graphs), threeJS (3D scatterplots and globes).
  - `googleVis`: use Google Chart tools to visualize data in R.
- **To model data**
  - `car`: car's Anova function is popular for making type II and type III Anova tables
  - `mgcv`: Generalized Additive Models
  - `lme4/nlme`: Linear and Non-linear mixed effects models
  - `randomForest`: random forest methods from machine learning
  - `multcomp`: tools for multiple comparison testing
  - `vcd`: visualization tools and tests for categorical data
  - `glmnet`:Lasso and elastic-net regression methods with cross validation
  - `survival`: tools for survival analysis
  - `caret`: tools for training regression and classification models
- **To report results**
  - `shiny`: make interactive web-apps (eg. explore data and share findings with non-programmers)
  - `R Markdown`: write R code in markdown report (when run render is run, R Markdown will replace the code with its results and then export your report as an HTML, pdf, or MS Word document, or a HTML or pdf slideshow. Hence allows automated reporting. R Markdown is integrated into RStudio.
  - `knitr`: the same tool but for use in LaTeX (and can be used for other markup languages)
  - `xtable`: coverts R objects (such as data frames) and returns the latex or HTML code
- **For Spatial data**
  - `sp, maptools`:tools for loading and using spatial data including shapefiles.
  - `maps`: use map polygons for plots.

  - `ggmap`: use street maps from Google maps as a background in ggplots

- **For Time Series and Financial data**

  - `zoo`: provides a format for saving time series objects
  - `xts`: tools for manipulating time series data sets
  - `quantmod`: tools for downloading financial data, plotting common charts, and doing technical analysis

- **To write high performance R code**

  - `Rcpp`: use C++ code from within R functions for fast speed
  - `data.table`: an alternative way to organize data sets for faster operations. Useful for big data.
  - `parallel`: parallel processing in R

- **To work with the web**

  - `XML`: read and create XML documents with R
  - `jsonlite`: read and create JSON data tables with R
  - `httr`: tools for working with http connections

- **To write your own R packages**

  - `devtools`: tools for turning your code into an R package
  - `testthat`: provides an easy way to write tests for your code
  - `roxygen2`: (like oxygen for C++) turns inline code comments into documentation pages and builds a package namespace.

Useful functions for libraries

```
# See the path where libraries are stored
.libPaths()

# See the list of installed packages
library()

# See the list of currently loaded packages
search()
```

1.3.9 Strings

Simple rules

- strings must start and end with single or double quotes
- a string must ends when the same quotes are encountered the next time
- until then it can contain the other type of quotes

```
a <- "Hello"
b <- "world"
paste(a,b,sep=", ")

## [1] "Hello, world"

c <- "A 'valid' string"
```

Formatting with `format()`

```
format(x, trim = FALSE, digits = NULL, nsmall = 0L, justify =
c("left", "right", "centre", "none"), width = NULL, na.encode =
TRUE, scientific = NA, big.mark = "", big.interval = 3L, small.mark
= "", small.interval = 5L, decimal.mark = getOption("OutDec"), zero.print
= NULL, drop0trailing = FALSE, ...)
```

- `x` is the vector input.
- `digits` is the total number of digits displayed.
- `nsmall` is the minimum number of digits to the right of the decimal point.
- `scientific` is set to TRUE to display scientific notation.
- `width` is the minimum width to be displayed by padding blanks in the beginning.
- `justify` is the display of the string to left, right or center.

Formatting examples

```
a<-format(100000000,big.mark=" ",
                  nsmall=3,
                  width=20,
                  scientific=FALSE,
                  justify="r")
print(a)

## [1] "       100 000 000.000"
```

More information? `?format` or `help(format)`

Other string functions

- `nchar()`: returns the number of characters in a string
- `toupper()`: puts the string in uppercase
- `tolower()`: puts the string in lowercase
- `substring(x,first,last)`: returns a substring from x starting with the "first" and ending with the "last"

1.4 Selected Data Interfaces

1.4.1 CSV Files

Import a CSV file

download the CSV file with currency exchange rates vs the euro from `http://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/index.en.html` or directly `http://www.ecb.europa.eu/stats/eurofxref/eurofxref-hist.zip?c6f8f9a0a5f970e31538be5271051b3c`

```
# To read a CSV-file it needs to be in the current directory
getwd()                              # show actual working directory
setwd("/home/philippe/Downloads") # change working directory
data <- read.csv("eurofxref-hist.csv")
is.data.frame(data)
ncol(data)
nrow(data)
head(data)
hist(data$CAD)
```

```
plot(data$USD,data$CAD)
```

Finding data

```
# get the maximum exchange rate
maxCAD <- max(data$CAD)
# use SQL-like selection
d0 <- subset(data, CAD == maxCAD)
d1 <- subset(data, CAD > maxCAD - 0.1)
d1[,1]

##  [1] 2008-12-30 2008-12-29 2008-12-18 1999-02-03 1999-01-29 1999-01-28
##  [7] 1999-01-27 1999-01-26 1999-01-25 1999-01-22 1999-01-21 1999-01-20
## [13] 1999-01-19 1999-01-18 1999-01-15 1999-01-14 1999-01-13 1999-01-12
## [19] 1999-01-11 1999-01-08 1999-01-07 1999-01-06 1999-01-05 1999-01-04
## 4718 Levels: 1999-01-04 1999-01-05 1999-01-06 1999-01-07 ... 2017-06-0

d2<- data.frame(d1$Date,d1$CAD)
d2
```

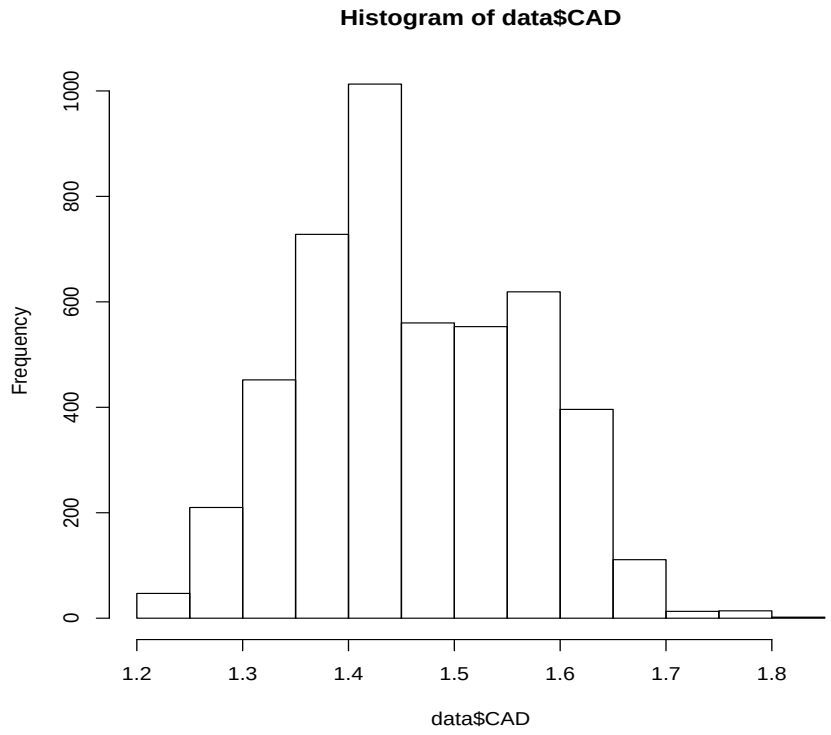


Figure 1.4: *The histogram of the CAD*

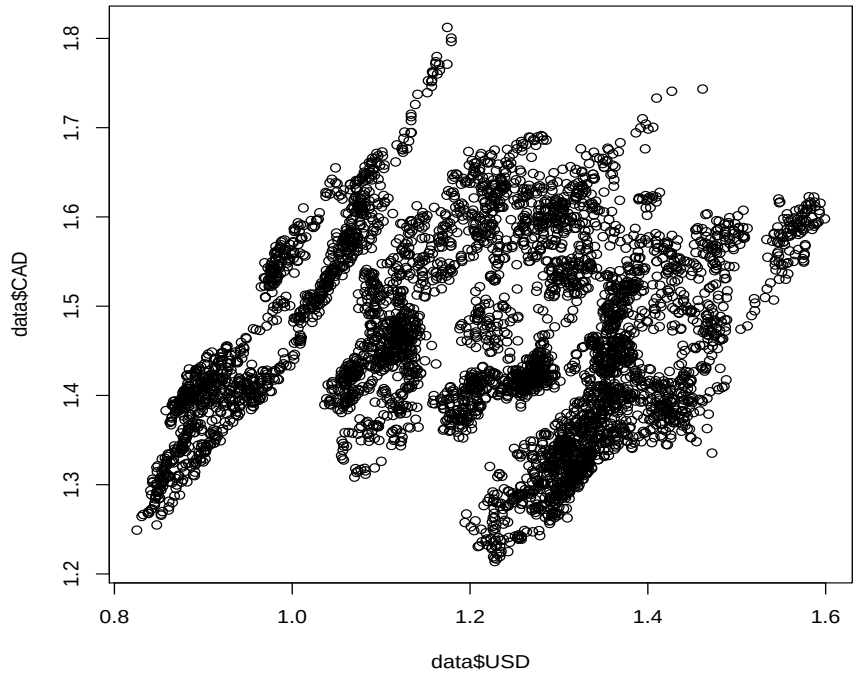


Figure 1.5: *A scatter-plot of one variable with another.*

```
##       d1.Date d1.CAD
## 1  2008-12-30 1.7331
## 2  2008-12-29 1.7408
## 3  2008-12-18 1.7433
## 4  1999-02-03 1.7151
## 5  1999-01-29 1.7260
## 6  1999-01-28 1.7374
## 7  1999-01-27 1.7526
## 8  1999-01-26 1.7609
## 9  1999-01-25 1.7620
## 10 1999-01-22 1.7515
## 11 1999-01-21 1.7529
## 12 1999-01-20 1.7626
## 13 1999-01-19 1.7739
## 14 1999-01-18 1.7717
## 15 1999-01-15 1.7797
## 16 1999-01-14 1.7707
## 17 1999-01-13 1.8123
## 18 1999-01-12 1.7392
## 19 1999-01-11 1.7463
## 20 1999-01-08 1.7643
## 21 1999-01-07 1.7602
## 22 1999-01-06 1.7711
## 23 1999-01-05 1.7965
## 24 1999-01-04 1.8004

hist(d2$d1.CAD)
```

Working with dates

```
# Get the recent peaks
recent <- subset(d2,
   as.Date(d1.Date) > as.Date("2005-01-01"))
print(recent)

##      d1.Date d1.CAD
## 1 2008-12-30 1.7331
## 2 2008-12-29 1.7408
## 3 2008-12-18 1.7433
```

Writing to a CSV file

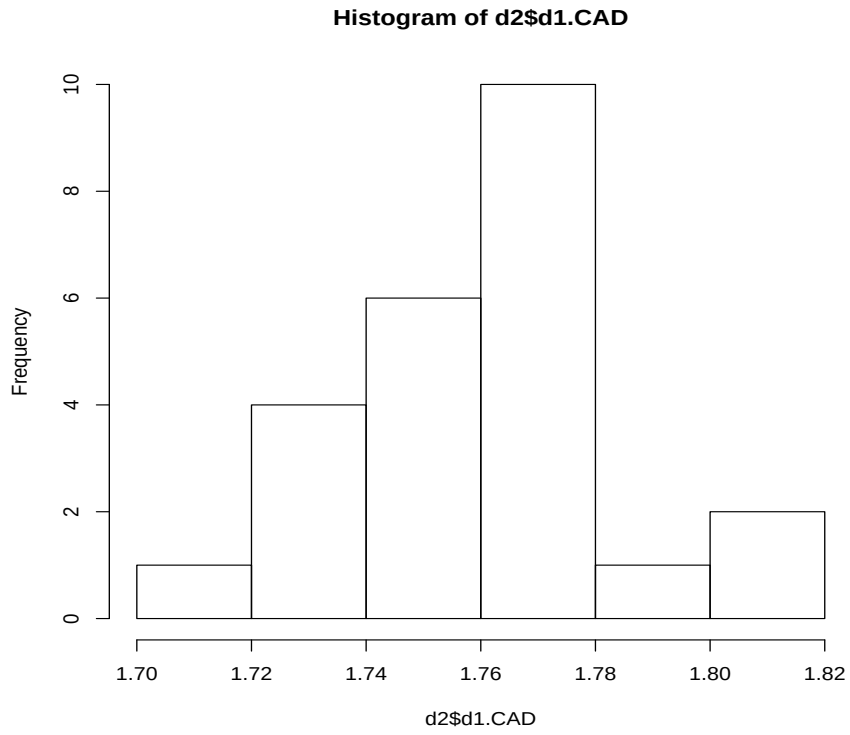


Figure 1.6: *The histogram of the most recent values of the CAD only.*

```
write.csv(d2,"output.csv", row.names = FALSE)
new.d2 <- read.csv("output.csv")
print(new.d2)

##       d1.Date d1.CAD
## 1  2008-12-30 1.7331
## 2  2008-12-29 1.7408
## 3  2008-12-18 1.7433
## 4  1999-02-03 1.7151
## 5  1999-01-29 1.7260
## 6  1999-01-28 1.7374
## 7  1999-01-27 1.7526
## 8  1999-01-26 1.7609
## 9  1999-01-25 1.7620
## 10 1999-01-22 1.7515
## 11 1999-01-21 1.7529
## 12 1999-01-20 1.7626
## 13 1999-01-19 1.7739
## 14 1999-01-18 1.7717
## 15 1999-01-15 1.7797
## 16 1999-01-14 1.7707
## 17 1999-01-13 1.8123
```

```
## 18 1999-01-12 1.7392
## 19 1999-01-11 1.7463
## 20 1999-01-08 1.7643
## 21 1999-01-07 1.7602
## 22 1999-01-06 1.7711
## 23 1999-01-05 1.7965
## 24 1999-01-04 1.8004
```

note: without the "row.names = FALSE" statement this procedure would add a row "X"

1.4.2 Excel Files

Excel Files are similar to CSV files

```
# install the package xlsx if not yet done
if (!any(grepl("xlsx",installed.packages()))){
  install.packages("xlsx")}
library(xlsx)
data <- read.xlsx("input.xlsx", sheetIndex = 1)
```

1.4.3 Databases

Databases

R can connect to many popular database systems. For example MySQL: as usual there is a package that will provide this functionality

```
if(!any(grepl("xls", installed.packages()))){
  install.packages("RMySQL")}
library(RMySQL)
```

Connecting to the database

```
# the connection is stored in an R object (myConnection)
# the connection needs the database name (db_name), username and password
myConnection = dbConnect(MySQL(),
   user = 'root',
   password = 'xxx',
   dbname = 'db_name',
   host = 'localhost')
```

```
# eg. list the tables available in this database.
dbListTables(myConnection)
```

Fetching data from a database

```
# Prepare the query for the database
result <- dbSendQuery(myConnection,
  "SELECT * from tbl_students WHERE age > 33")

# Fetch all the records(with n = -1) and store it in a data frame.
data <- fetch(result, n = -1)
```

Update Queries

The `dbSendQuery()` function can be used to send any query, including UPDATE, INSERT, CREATE TABLE and DROP TABLE queries so we can push results back to the database.

```
sSQL = ""
sSQL[1] <- "UPDATE tbl_students
            SET score = 'A' WHERE raw_score > 90;"
sSQL[2] <- "INSERT INTO tbl_students
            (name, class, score, raw_score)
    VALUES ('Robert', 'Grade 0', 88,NULL);"
sSQL[3] <- "DROP TABLE IF EXISTS tbl_students;"
for (k in c(1:3)){
  dbSendQuery(myConnection, sSQL[k])
  }
```

Create tables from R data frames

R can write the value of a data frame into a table

```
dbWriteTable(myConnection, "tbl_name",
             data_frame_name[, ], overwrite = TRUE)
```

1.5 Charts & Graphs

1.5.1 Pie Charts

Pie charts

Definition 4 .:. pie() .:.

`pie(x, labels = names(x), edges = 200, radius = 0.8, clockwise = FALSE, init.angle = if(clockwise) 90 else 0, density = NULL, angle = 45, col = NULL, border = NULL, lty = NULL, main = NULL, ...)`
where the most important parameters are

- `x`: a vector of non-negative numerical quantities. The values in x are displayed as the areas of pie slices
- `labels`: strings with names for the slices
- `radius`: the radius of the circle of the chart (value between 1 and +1)
- `main`: indicates the title of the chart
- `col`: the colour palette
- `clockwise`: a logical value indicating if the slices are drawn clockwise or anti clockwise

Pie chart example

```
# Create data for the graph.
x <- c(10, 20, 12)
labels <- c("good", "average", "bad")

# Give the chart file a name.
png(file = "feedback.jpg")

# Plot the chart.
pie(x,labels)

# Save the file.
dev.off()

## pdf
##   2
```

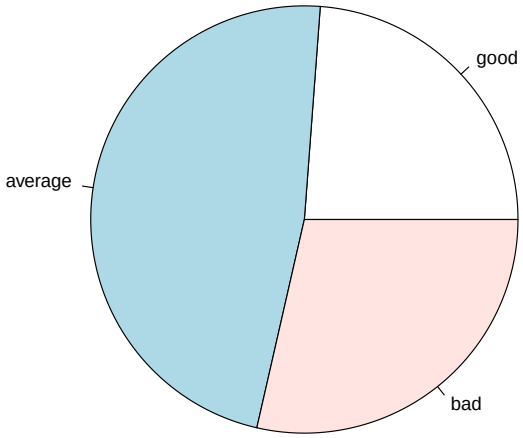


Figure 1.7: *A pie-chart in R.*

```
# Show the chart in the R Graphics screen
pie(x,labels)
```

1.5.2 Bar Charts

The function barplot()

Definition 5 .:. barplot() .:.

`barplot(height, width=1, xlab=NULL, ylab=NULL, main=NULL, names.arg=NuLL, col=NULL, ...)` Some parameters:

- `height`: is the vector or matrix containing numeric values used in chart
- `xlab`: the label for the x-axis
- `ylab`: is the label for y axis
- `main`: is the title of the chart
- `names.arg`: is a vector of names of each bar
- `col`: is used to give colors to the bars in the graph.

An example for barplot()

```
sales <- c(100,200,150,50,125)
regions <- c("France", "Poland", "UK", "Spain", "Belgium")
barplot(sales, width=1,
        xlab="Regions", ylab="Sales in EUR",
        main="Sales 2016", names.arg=regions,
        border="blue", col="brown")
```

Stacked bar charts

```
# Create the input vectors.
colors <- c("orange","green","brown")
regions <- c("Mar","Apr","May","Jun","Jul")
product <- c("Licence","Maintenance","Consulting")

# Create the matrix of the values.
Values <- matrix(c(20,80,0,50,140,10,50,80,20,10,30,
       10,25,60,50),nrow = 3,ncol = 5,byrow = FALSE)

# Create the bar chart.
barplot(Values,main = "Sales 2016",
   names.arg = regions,xlab = "region",
   ylab = "sales in EUR",col = colors)

# Add the legend to the chart.
legend("topright", product, cex = 1.3, fill = colors)
```

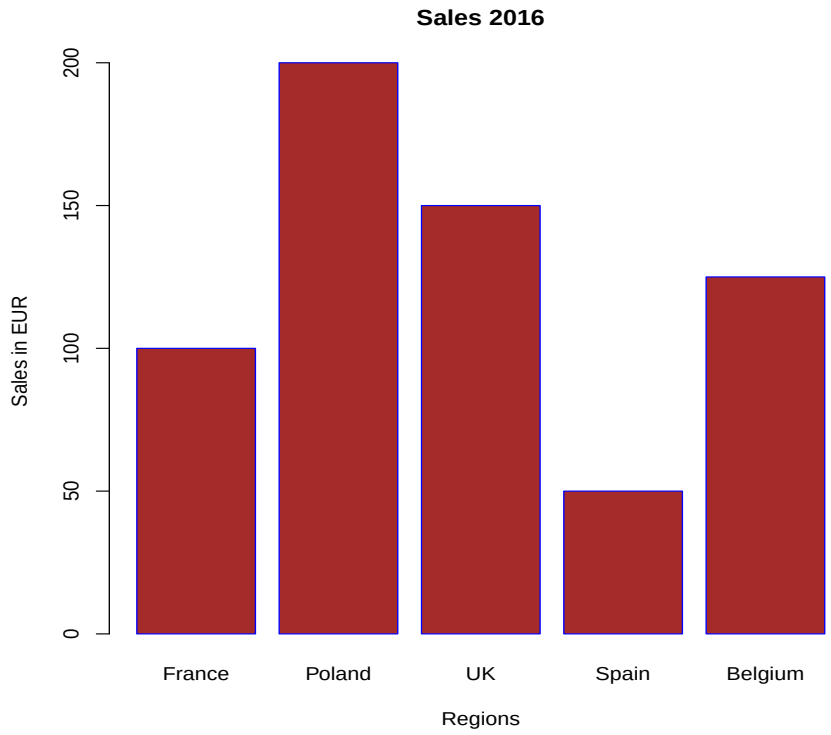


Figure 1.8: *A standard bar-chart based on a vector.*

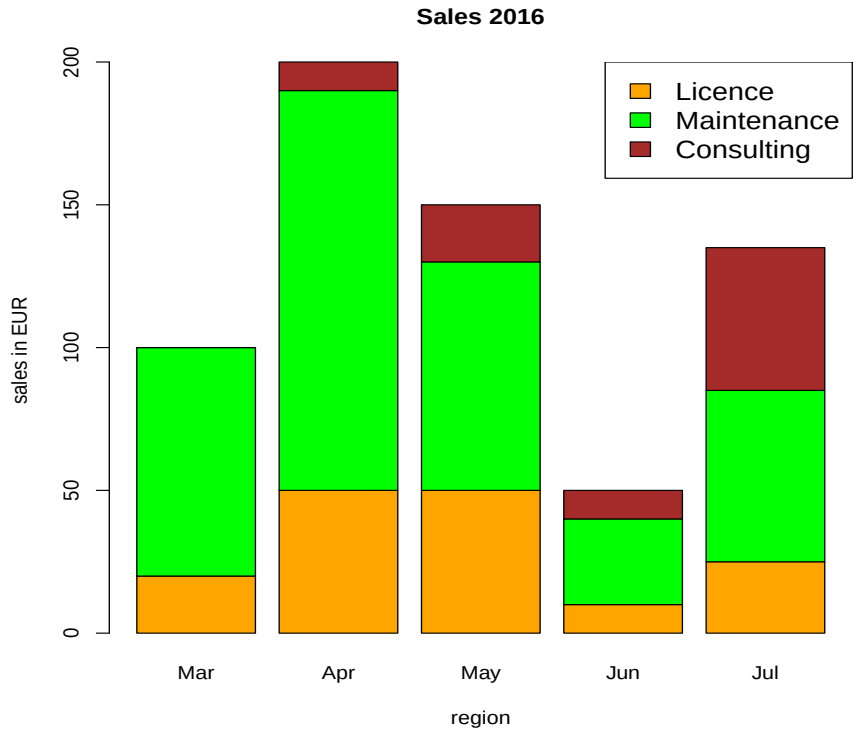


Figure 1.9: *A bar-chart based on a matrix will produce stacked bars.*

1.5.3 Boxplots

Boxplots

Definition 6 .:. boxplot() .:.

`boxplot(formula, data = NULL,notch = FALSE, varwidth = FALSE, names, main = NULL, ...` with: Following is the description of the parameters used

- `formula`: a vector or a formula.
- `data`: the data frame.
- `notch`: a logical value (set to TRUE to draw a notch)
- `varwidth`: a logical value (set to true to draw width of the box proportionate to the sample size)
- `names`: the group labels which will be printed under each boxplot.
- `main`: the title to the graph.

A boxplot example

Let's use the dataset `ships` (from the library "MASS")

```
library(MASS)
boxplot(incidents ~ type,data=ships,col="green", main="# of incidents in
```

```
boxplot(incidents/(76-year) ~ type,data=ships,col="green", main="# of inc
```

1.5.4 Histograms

Boxplots

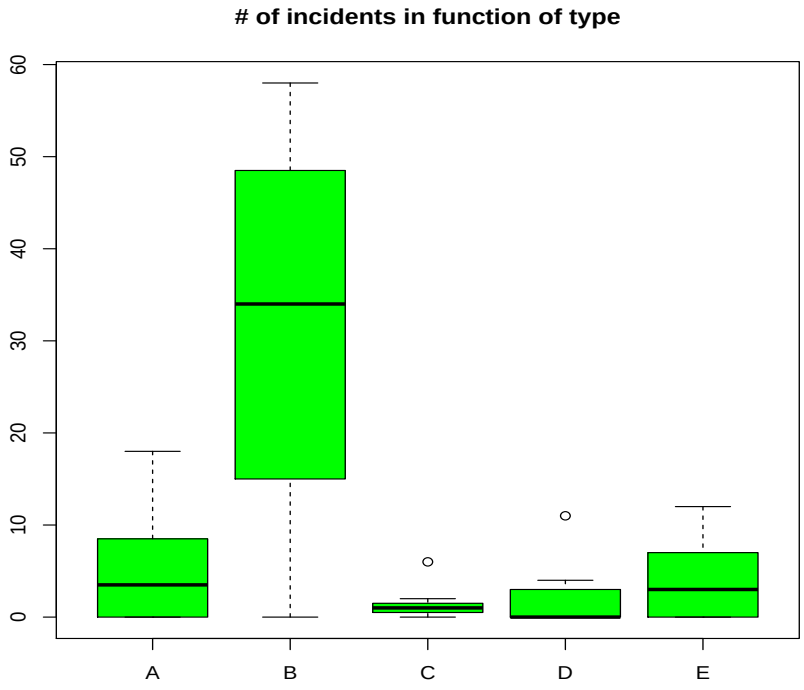


Figure 1.10: *Boxplots show information about the central tendency (median) as well as the spread of the data.*

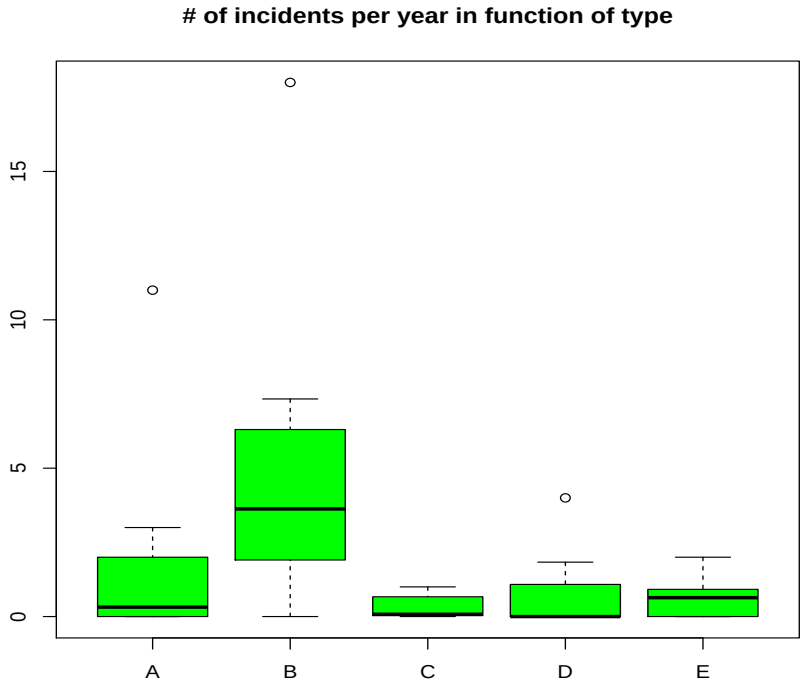


Figure 1.11: *In this boxplot the number of incidents is compared to the number of years a ship is in service.*

Definition 7 .:. hist() .:.

`hist(x, breaks = "Sturges", freq = NULL, probability = !freq, include.lowest = TRUE, right = TRUE, density = NULL, angle = 45, col = NULL, border = NULL, main = paste("Histogram of" , deparse(substitute(x))), xlim = range(breaks), ylim = NULL, xlab = deparse(substitute(x)), ylab, axes = TRUE, plot = TRUE, labels = FALSE, nclass = NULL, warn.unused = TRUE, ...)` with the most important parameters:

- `x`: the vector containing numeric values to be used in the histogram
- `main`: the title of the chart
- `col`: the color of the bars
- `border`: the border color of each bar
- `xlab`: the title of the x-axis
- `xlim`: the range of values on the x-axis
- `ylim`: the range of values on the y-axis
- `breaks`: one of
  - a vector giving the breakpoints between histogram cells,
  - a function to compute the vector of breakpoints,
  - a single number giving the number of cells for the histogram,

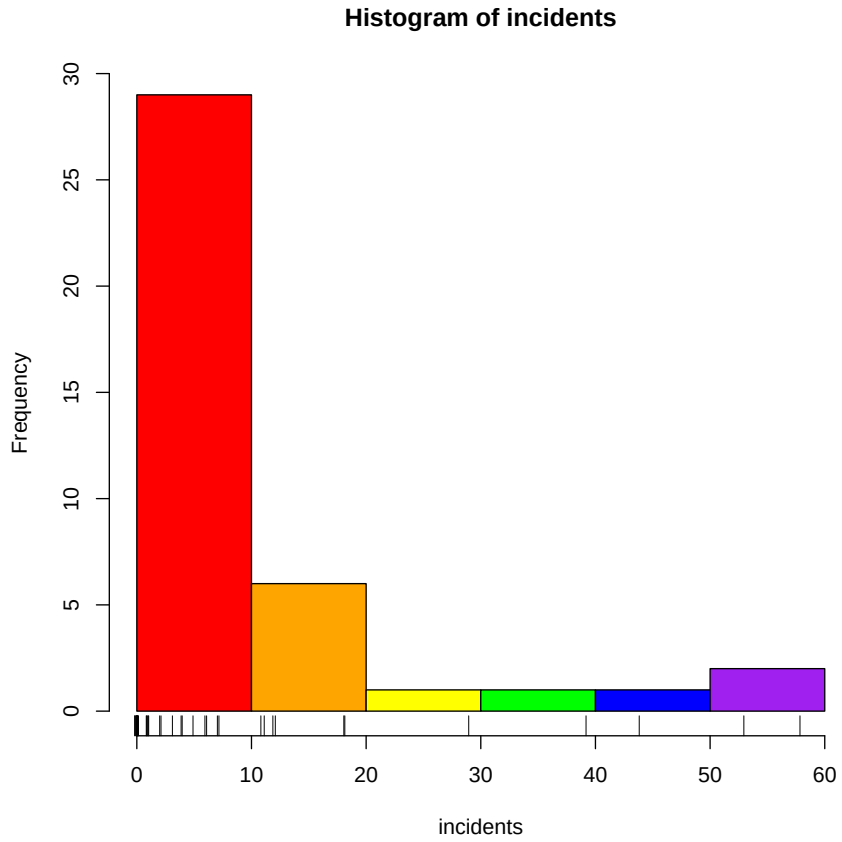


Figure 1.12: *A histogram in R is produced by the hist() function.*

Histogram example

```
library(MASS)
incidents <- ships$incidents
# figure 1
hist(incidents,
     col=c("red","orange","yellow","green","blue","purple"))
rug(jitter(incidents))  # add the tick-marks
```

```
# figure 2
hist(incidents,
    col=c("red","orange","yellow","green","blue","purple"),
    ylim=c(0,0.3), breaks=c(0,2,5,10,20,40,80),freq=FALSE)
```

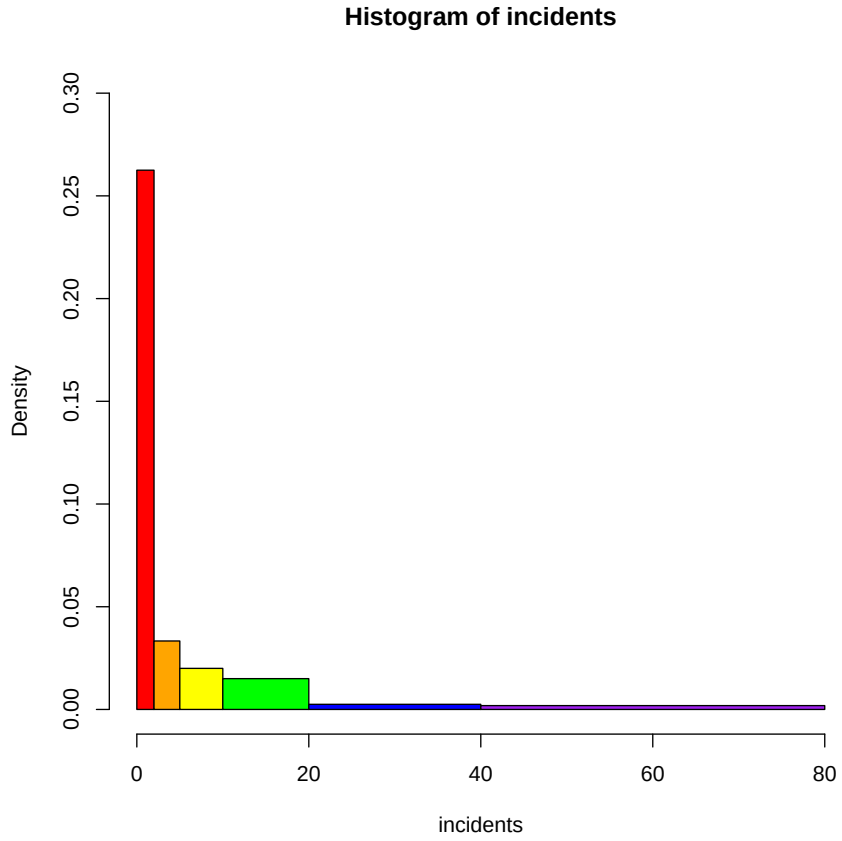


Figure 1.13: *In this histogram the breaks are changed and the y-axes is now calibrated as a probability. Note that leaving freq=TRUE would give the wrong impression that there are more observations in the wider brackets.*

1.5.5 Scatterplots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the combination of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

Making scatterplots

Definition 8 .:. scatterplot .:.

`plot(x, y, main, xlab, ylab, xlim, ylim, axes, ...)` with

- `x`: the data set for the horizontal axis
- `y`: the data set for the vertical axis
- `main`: the tile of the graph
- `xlab`: the title of the x-axis
- `ylab`: the title of the x-axis
- `xlim`: the range of values on the x-axis
- `ylim`: the range of values on the y-axis
- `axes`: indicates whether both axes should be drawn on the plot.

Scatterplot example

```
# prepare the data
library(MASS)
mpg2l <- function(mpg = 0) {
  100 * 3.785411784 / 1.609344 / mpg}
mtcars$l <- mpg2l(mtcars$mpg)

# Plot
plot(x = mtcars$hp,y = mtcars$l,
   xlab = "Horse Power", ylab = "L per 100km",
   main = "Horse Power vs Milage")
```

1.5.6 Line Graphs

It might be sufficient to add lines to a scatterplot (with the lines() function), but other methods are available

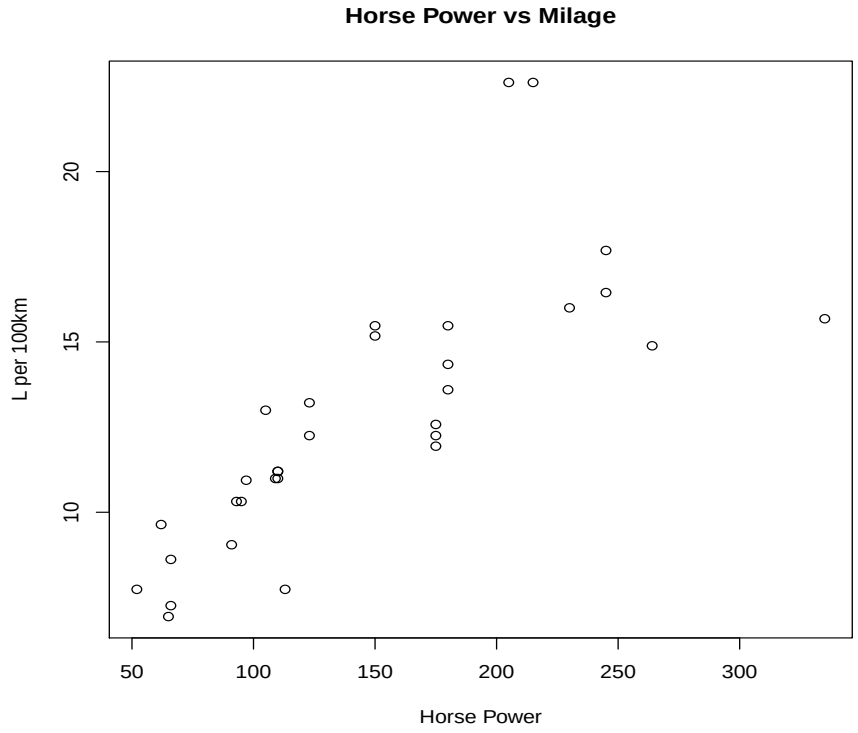


Figure 1.14: *A scatter-plot needs an x and a y variable.*

Making line plots

Definition 9 .:. line plots .:.

`plot(x, type , main, xlab, ylab, xlim, ylim, axes, sub, asp ...)` with

- `x`: the data set for the horizontal axis
- `y`: the data set for the vertical axis (optional)
- `type`: indicates the type of plot to be made:
  - "p" for *p*oints,
  - "l" for *l*ines,
  - "b" for *b*oth,
  - "c" for the lines part alone of "b",
  - "o" for both *o*verplotted,
  - "h" for *h*istogram like (or high-density) vertical lines,
  - "s" for stair *s*teps,
  - "S" for other *s*teps, see Details in the documentation,
  - "n" for no plotting.
- `main`: the tile of the graph
- `xlab`: the title of the x-axis
- `ylab`: the title of the x-axis
- `xlim`: the range of values on the x-axis
- `ylim`: the range of values on the y-axis
- `axes`: indicates whether both axes should be drawn on the plot.
- `sub`: the sub-title
- `asp`: the y/x aspect ratio

A line-plot example

```
# prepare the data
years <- c(2000,2001,2002,2003,2004,2005)
sales <- c(2000,2101,3002,2803,3500,3450)
# Plot
plot(x = years,y = sales, type = 'b',
   xlab = "Years", ylab = "Sales in USD",
   main = "The evolution of our sales")
```

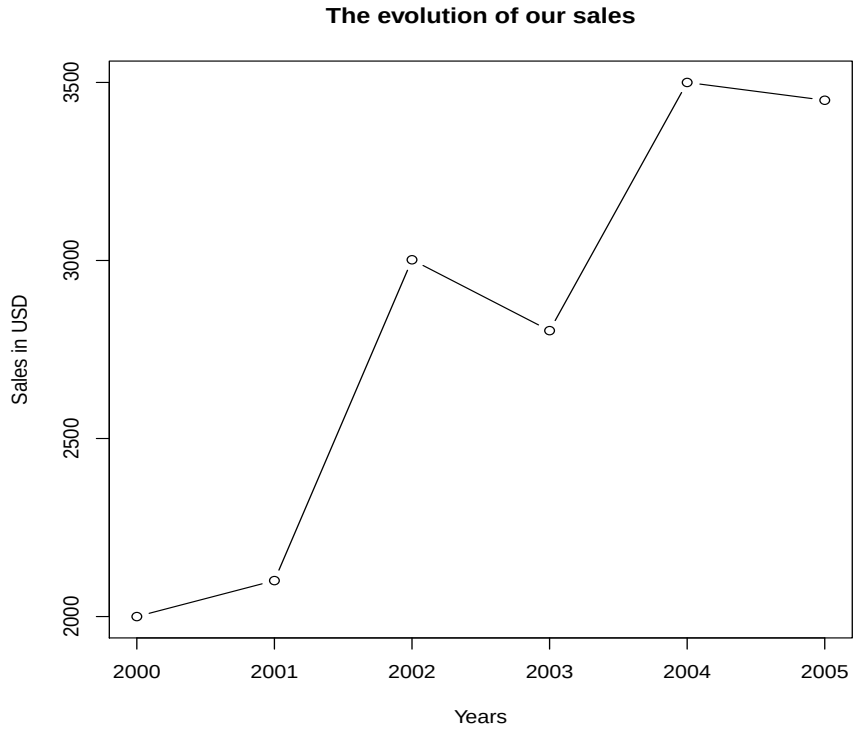


Figure 1.15: *A line plot of the type b.*

1.5.7 Plotting Functions

While the function plot() allows to draw functions, there is a specific function curve:

```
fn1 <- function(x) sqrt(1-(abs(x)-1)^2)
fn2 <- function(x) -3*sqrt(1-sqrt(abs(x))/sqrt(2))
curve(fn1,-2,2,ylim=c(-3,1),col="red",lwd=4)
curve(fn2,-2,2,add=TRUE,lw=4,col="red")
```

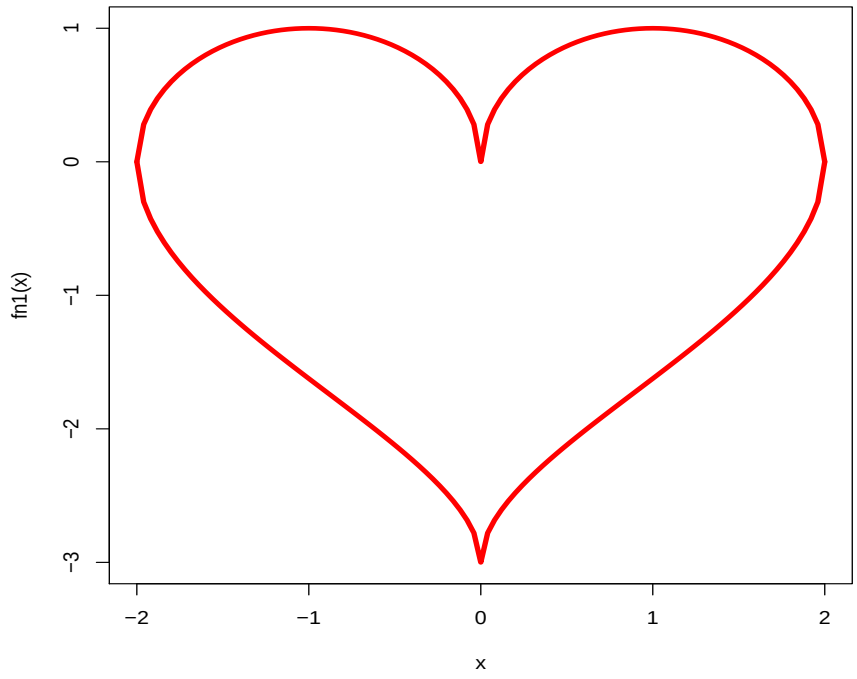


Figure 1.16: *Two line plots plotted by the function curve().*

PART I .:. CHAPTER 2 .:.

Selected Notions from Statistics

2.1 Measures of Central Tendency

A measure of central tendency is a single value that attempts to describe a set of data by identifying the central position within that set of data. As such, measures of central tendency are sometimes called measures of central location. They are also classed as summary statistics. The mean (often called the average) is most likely the measure of central tendency that you are most familiar with, but there are others, such as the median and the mode.

The mean, median and mode are all valid measures of central tendency, but under different conditions, some measures of central tendency become more appropriate to use than others. In the following sections, we will look at the mean, mode and median, and learn how to calculate them and under what conditions they are most appropriate to be used.

2.1.1 Mean

Arithmetic Mean

Definition 10 .:. Arithmetic Mean .:.

$$\bar{x} = \sum_{n=1}^{N} P(x).x \qquad \text{(for discrete distributions)}$$

$$= \int_{-\infty}^{+\infty} x.f(x)\,\mathrm{d}x \qquad \text{(for continuous distributions)}$$

The unbiased estimator of the mean for K observations x_k is:

$$E[\bar{x}] = \frac{1}{K}\sum_{k=1}^{K} x_k$$

the mean in R

```
# The mean of a vector
x <- c(1,2,3,4,5,60)
mean(x)

## [1] 12.5

mean(ships$service)

## [1] 4089.35

# Ignoring missing values
x <- c(1,2,3,4,5,60,NA)
mean(x)

## [1] NA

mean(x,na.rm = TRUE)

## [1] 12.5

# This works also for a matrix
M <- matrix(c(1,2,3,4,5,60), nrow=3)
mean(M)

## [1] 12.5
```

Generalized means

Definition 11 .:. f-mean .:.

$$\bar{x} = f^{-1}\left(\frac{1}{n}.\sum_{k=1}^{K} f(x_k)\right)$$

Popular choices for are:

- $f(x) = x$: arithmetic mean,
- $f(x) = \frac{1}{x}$: harmonic mean,
- $f(x) = x^m$: power mean,
- $f(x) = \ln x$: geometric mean, so $\bar{x} = \left(\prod_{k=1}^{K} x_k\right)^{\frac{1}{K}}$

The Power Mean

One particular generalized mean is the power mean or Hlder mean. It is defined for a set of K positive numbers x_k by

$$\bar{x}(m) = \left(\frac{1}{n} \cdot \sum_{k=1}^{K} x_k^m \right)^{\frac{1}{m}}$$

by choosing particular values for m one can get the quadratic, arithmetic, geometric and harmonic means.

- $m \to \infty$: maximum of x_k
- $m = 2$: quadratic mean
- $m = 1$: arithmetic mean
- $m \to 0$: geometric mean
- $m = 1$: harmonic mean
- $m \to -\infty$: minimum of x_k

Which mean makes most sense?

What is the average return when you know that the share price had the following returns: $-50\%, +50\%, -50\%, -50\%$. Try the arithmetic mean and the mean of the log-returns.

```
returns <- c(0.5,-0.5,0.5,-0.5)

# arithmetic mean
aritmean <- mean(returns)

# the ln-mean
log_returns <- returns
for(k in 1:length(returns)) {
  log_returns[k] <- log( returns[k] + 1)
  }
logmean <- mean(log_returns)
exp(logmean) - 1

## [1] -0.1339746

# What is the value of the investment after these returns
V_0 <- 1
V_T <- V_0
for(k in 1:length(returns)) {
  V_T <- V_T * (returns[k] + 1)
  }
V_T
```

```
## [1] 0.5625

# Compare this to our predictions
## mean of log-returns
V_0 * (exp(logmean) - 1)

## [1] -0.1339746

## mean of returns
V_0 * (aritmean + 1)

## [1] 1
```

2.1.2 Median

The median

The median is the middle-value so that 50% of the observations are lower and 50% are larger.

```
# The median of an R-object
x <- c(1,2,3,4,5,60,NA)
median(x)

## [1] NA

median(x,na.rm = TRUE)

## [1] 3.5
```

2.1.3 Arithmetic Mode

The mode

The mode is the value that has highest probability to occur. For a series of observations, this should be the one that occurs most often. Note that the mode is also defined for variables that have no order-relation (even labels such as "green", "yellow", etc. have a mode, but not a mean or median — without further abstraction or a numerical representation).

In R the function `mode()` or `storage.mode()` returns a character string describing how a variable is stored. In fact, R does not have a standard function to calculate mode, so let's create our own:

```
my_mode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
   }

# now test this function
x <- c(1,2,3,3,4,5,60,NA)
my_mode(x)

## [1] 3

x1 <- c("relevant", "N/A", "undesired", "great", "N/A",
        "undesired", "great", "great")
my_mode(x1)

## [1] "great"

# consider this text from https://www.r-project.org/about.html
t <- "R is available as Free Software under the terms of the
Free Software Foundations GNU General Public License in source
code form. It compiles and runs on a wide variety of UNIX
platforms and similar systems (including FreeBSD and Linux),
Windows and MacOS."
v <- unlist(strsplit(t,split=" "))
my_mode(v)

## [1] "and"
```

2.2 Measures of Variation or Spread

Definition 12 .:. variance .:.

$$VAR(X) = E\left[\left(X - \bar{X}\right)^2\right]$$

Standard Deviation

Definition 13 .:. standard deviation .:.

$$\widehat{SD(X)} := \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}\left(X_n - \bar{X}\right)^2}$$

```
t <- test.data$score
var(t)

## [1] 29.2

sd(t)

## [1] 5.403702

sqrt(var(t))

## [1] 5.403702

sqrt(sum((t - mean(t))^2)/(length(t) - 1))

## [1] 5.403702
```

Median Absolute Deviation

Definition 14 .:. MAD .:.

$$MAD(X) = \frac{1}{1.4826} median\left(|X - median(X)|\right)$$

```
mad(t)

## [1] 5.9304

mad(t,constant=1)

## [1] 4
```

The default "constant = 1.4826" (approximately $\frac{1}{\Phi^{-1}(\frac{3}{4})} = \frac{1}{qnorm(3/4)}$ ensures consistency, i.e.,

$$E[mad(X_1, ..., X_n)] = \sigma$$

for X_i distributed as N(μ, σ^2) and large n.

2.3 Measures of Covariation

The basic measure for linear interdependence is covariance, defined as

$$\begin{aligned} covar(X) &= E\left[(X - E[X])\,(Y - E[Y])\right] \\ &= E[XY] - E[X]E[Y] \end{aligned}$$

An important metric for linear relationship is the Pearson correlation coefficient ρ.

Pearson correlation

Definition 15 .:. Pearson Correlation Coefficient .:.

$$\begin{aligned} \rho_{XY} &= \frac{covar(X,Y)}{\sigma_X \sigma_Y} \\ &= \frac{(X - E[X])(Y - E[Y])}{\sqrt{(X - E[X])(Y - E[Y])}} \\ &=: covar(x,y) \end{aligned}$$

```
cor(mtcars$hp,mtcars$wt)

## [1] 0.6587479
```

The Spearman correlation

The Spearman correlation is the correlation applied to the ranks of the data. It is one if an increase in the variable X is always accompanied with an increase in variable Y.

```
cor(rank(mtcars$hp),rank(mtcars$wt))

## [1] 0.7746767
```

The Spearman correlation checks for a relationship that can be more general than only linear. It will be one if X increases when Y increases.

Exercise: correlation

Question

Consider the vectors $X = (1, 2, 33, 44)$ and $Y = (22, 23, 100, 200)$. Plot Y in function of X. What is their Pearson correlation? What is their Spearman correlation? How do you understand that?

. . . . . .

. . . . . .

. . . . . .

. . . . . .

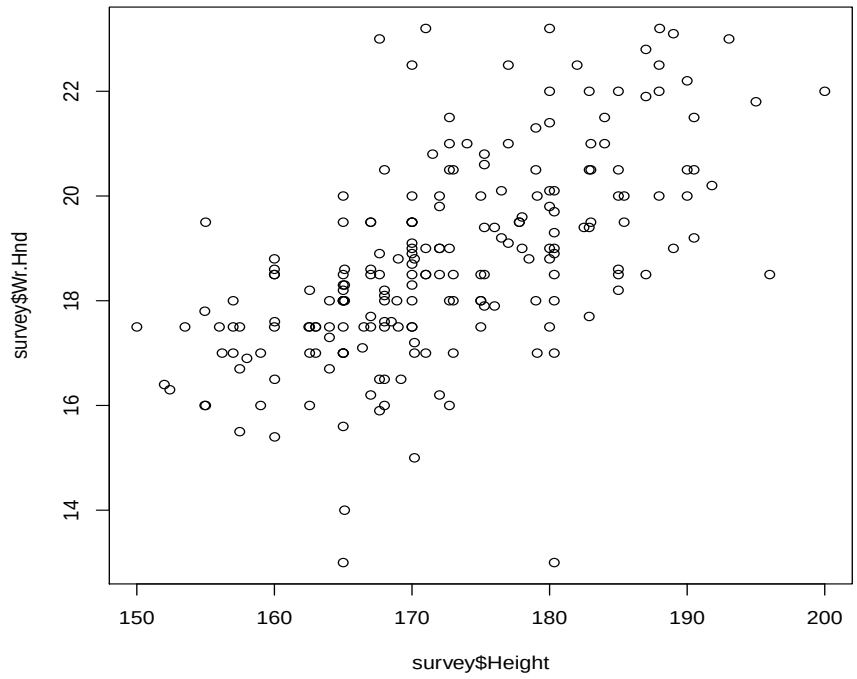


Figure 2.1: *A scatter-plot generated by the line "plot(survey$Height, survey$Wr.Hnd)".*

2.4 Regression Models

2.4.1 Linear Regression

Linear Regression

With a linear regression we try to estimate an unknown variable y based on a known variable x and some constants (a and b). It's form is

$$y = ax + b$$

```
library(MASS)

# Explore the data
plot(survey$Height, survey$Wr.Hnd)
```

```
# Create the model
lm1 <- lm (formula = Wr.Hnd ~ Height, data = survey)
summary(lm1)

##
## Call:
## lm(formula = Wr.Hnd ~ Height, data = survey)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.6698 -0.7914 -0.0051  0.9147  4.8020
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.23013    1.85412  -0.663    0.508
## Height       0.11589    0.01074  10.792   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.525 on 206 degrees of freedom
##   (29 observations deleted due to missingness)
## Multiple R-squared:  0.3612,Adjusted R-squared:  0.3581
## F-statistic: 116.5 on 1 and 206 DF,  p-value: < 2.2e-16

# predictions
h <- data.frame(Height = 150:200)
Wr.lm <- predict(lm1,h)
plot(survey$Height, survey$Wr.Hnd,col="red")
lines(t(h),Wr.lm,col="blue")
```

```
# Or use the function abline()
plot(survey$Height, survey$Wr.Hnd,col = "red",
     main = "Hand span in function of Height",
     abline(lm(survey$Wr.Hnd ~ survey$Height )),
     cex = 1.3,pch = 16,
     xlab = "Height",ylab ="Hand Span")
```

Exercise: linear regression

Question

Consider the data-set mtcars from the library MASS. Make a linear regression of the fuel consumption in function of the parameter that according to you has the most explanatory power. Study the residuals. What is your conclusion?

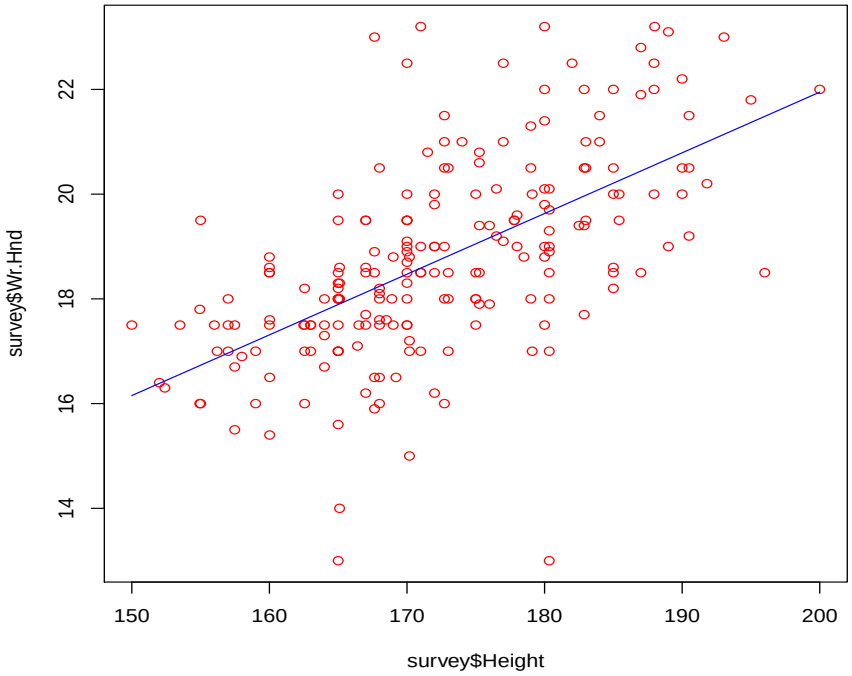


Figure 2.2: *A lot visualizing the linear regression model (the data in red and the regression in blue.*

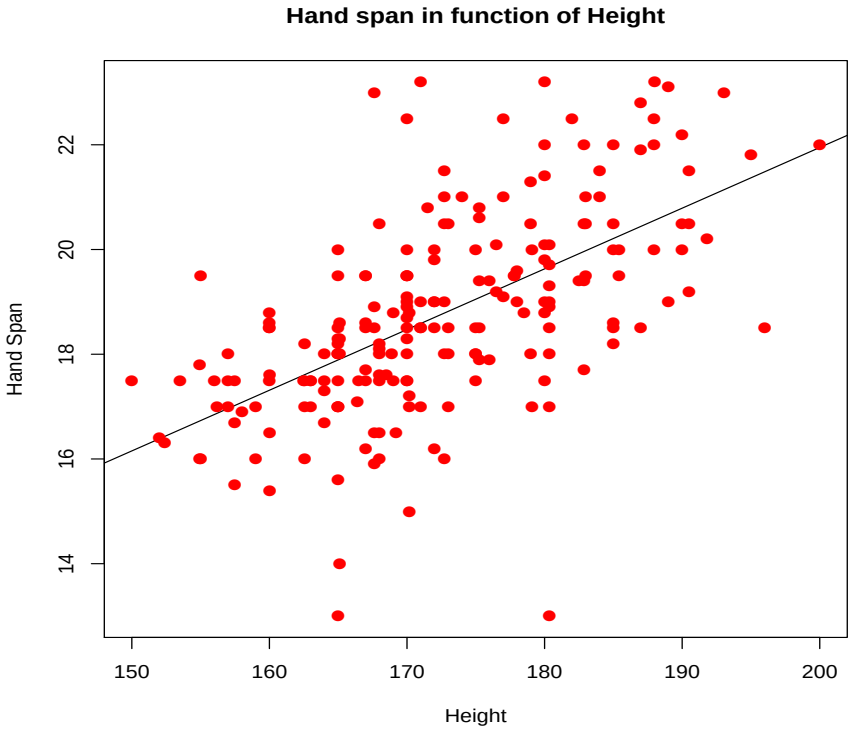


Figure 2.3: *Using the function abline() and cleaning up the titles.*

.....

.....

.....

.....

2.4.2 Multiple Linear Regression

Multiple Linear Regression

Multiple regression is a relationship between more than two known variables to predict one variable.

$$y = b + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$$

In R the `lm()` function will handle this too.

```
# We use our modified version of mtcars from the library MASS
model <- lm(l~disp+hp+wt, data = mtcars)
print(model)

##
## Call:
## lm(formula = l ~ disp + hp + wt, data = mtcars)
##
## Coefficients:
## (Intercept)         disp           hp           wt
##    2.233668     0.005776     0.013793     2.227208

# Accessing the coefficients
print(a)

## [1] "       100 000 000.000"

a_disp <- coef(model)[2]
a_hp   <- coef(model)[3]
a_wt   <- coef(model)[4]

print(a_disp)

##        disp
## 0.005776108

print(a_hp)

##         hp
## 0.01379265
```

```
print(a_wt)

##       wt
## 2.227208

# This allows us to manually predict the fuel consumption
# eg. for the Mazda Rx4
2.23 + a_disp * 160 + a_hp * 110 + a_wt * 2.62

##     disp
## 10.50665
```

Exercise: multiple linear regression

Question

Consider the data-set mtcars from the library MASS. Make a linear regression that predicts the fuel consumption of a car. Make sure to include only significant variables and remember that the significance of a variable depends on the other variables in the model.

.....

.....

.....

.....

2.4.3 Logistic Regression

Logistic regression (aka logit regression) is a regression model where the unknown variable is categorical (can have only a limited number of values). For example it can be binoial: where it can take only two values, "0" and "1", which represent outcomes such as repay/default, pass/fail, win/lose, survive/die or healthy/sick.

Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choices.

Logistic Regression

The general model of a logistic regression is:

$$y = \frac{1}{1 + e^{-(b+a_1x_1+a_2x_2+a_3x_3+...)}}$$

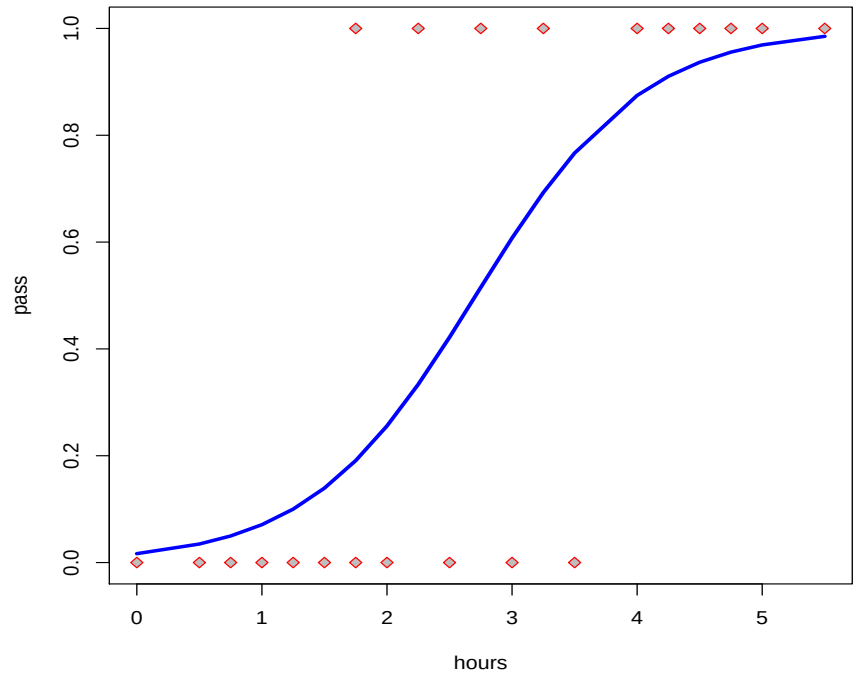


Figure 2.4: *The grey diamonds with red border are the data-points (not passed is 0 and passed is 1) and the blue line represents the logistic regression model (or the probability to succeed the exam in function of the hours studied*

```
# Consider the relation between the hours studied and passing
# an exam
hours <- c(0,0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00,
         2.25, 2.50, 2.75, 3.00, 3.25, 3.50, 4.00, 4.25,
           4.50, 4.75, 5.00, 5.50)
pass  <- c(0,0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1,
         1, 1, 1, 1)
d <- data.frame(cbind(hours,pass))
m <- glm(formula=pass ~ hours, family = binomial, data = d)

# Visualize the results
plot(hours,pass,col="red",pch = 23,bg="grey")
pred <- 1 / (1+ exp(-(-4.078 + hours * 1.505)))
lines(hours,pred,col="blue",lwd=3)
```

2.4.4 Poisson Regression

The Poisson regression can be useful where the unknown variable (response variable) can never be negative, for example in the case of predicting counts (eg. numbers of events).

Poisson Regression

Definition 16 .:. Poisson Regression .:.

The general form of the Poisson Regression is

$$\log(y) = b + a_1 x_1 + a_2 x_2 + b_n x_n$$

with:

- y: the predicted variable (aka response variable or unknown variable)
- a and b are the numeric coefficients.
- x is the known variable (or the predictor variable)

the Poisson Regression in R

Also the Poisson Regression can be handled by the function `glm()` in R

```
glm(formula,data,family)
```

where:

- `formula` is the symbol presenting the relationship between the variables,
- `data` is the data-set giving the values of these variables,
- `family` is R object to specify the details of the model and for the Poisson Regression it's value is 'Poisson'

Example

For example we will check if we can estimate the number of cylinders of a car based on its horse power and weight, using the dataset `mtcars`

```
m <- glm(cyl ~ hp + wt, data = mtcars, family = "poisson")
summary(m)

##
## Call:
## glm(formula = cyl ~ hp + wt, family = "poisson", data = mtcars)
```

```
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.59240  -0.31647  -0.00394   0.29820   0.68731
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.064836   0.257317   4.138  3.5e-05 ***
## hp          0.002220   0.001264   1.756    0.079 .
## wt          0.124722   0.090127   1.384    0.166
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 16.5743  on 31  degrees of freedom
## Residual deviance:  4.1923  on 29  degrees of freedom
## AIC: 126.85
##
## Number of Fisher Scoring iterations: 4
```

Weight does not seem to be relevant, we we try only using horse power:

```
m <- glm(cyl ~ hp, data = mtcars, family = "poisson")
summary(m)

##
## Call:
## glm(formula = cyl ~ hp, family = "poisson", data = mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.97955  -0.30748  -0.03387   0.28155   0.73433
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.3225669  0.1739422   7.603 2.88e-14 ***
## hp          0.0032367  0.0009761   3.316 0.000913 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 16.5743  on 31  degrees of freedom
## Residual deviance:  6.0878  on 30  degrees of freedom
## AIC: 126.75
```

```
##
## Number of Fisher Scoring iterations: 4
```

2.4.5 Non-Linear Regression

In many cases one will observe that the relation between the unknown variable and the known variables is not simply linear. Whenever we plot the data and see that the relation is not a straight line but rather a curve the relation is non-linear. It is possible to model this by applying a function such as squaring, a sine, a logarithm or exponential to the known variable(s) and then running a linear regression. However, R, has a specific function for this: `nls()`.

In Least Square regression, we establish a regression model in which the sum of the squares of the vertical distances of different points from the regression curve is minimized. We generally start with a defined model and assume some values for the coefficients. We then apply the nls() function of R to get the more accurate values along with the confidence intervals.

Syntax of Non-Linear Regression

Definition 17 .:. nls() .:.

`nls(formula, data, start)` with

1. `formula` a non-linear model formula including variables and parameters,
2. `data` the data-frame used to optimize the model,
3. `start` a named list or named numeric vector of starting estimates.

Example for nls()

```
# consider observations for dt = d0 + v0 t + 1/2 a t^2
t  <- c(1,2,3,4,5,1.5,2.5,3.5,4.5,1)
dt <- c(8.1,24.9,52,89.2,136.1,15.0,37.0,60.0,111.0,8)

# Plot these values.
plot(t,dt,xlab="time",ylab="distance")

# Take the assumed values and fit into the model.
model <- nls(dt ~ d0 + v0 * t + 1/2 * a * t^2,
             start = list(d0 = 1,v0 = 3,a = 10))

# plot the model curve
```

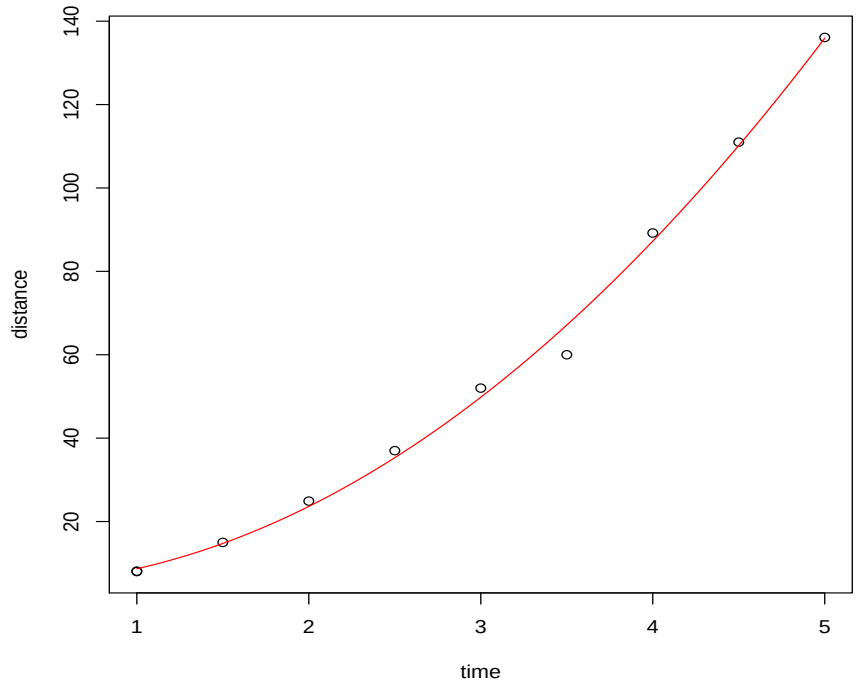


Figure 2.5: *The results of the non-linear regression with nls(). This plot indicates that there is one outlier and you might want to re-run the model without this observation.*

```
simulation.data <- data.frame(t = seq(min(t),max(t),len = 100))
lines(simulation.data$t,predict(model,newdata = simulation.data), col="re
```

```
# Learn about the model
summary(model)                # the summary

##
## Formula: dt ~ d0 + v0 * t + 1/2 * a * t^2
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## d0    4.981      4.660   1.069    0.321
## v0   -1.925      3.732  -0.516    0.622
## a    11.245      1.269   8.861 4.72e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.056 on 7 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.822e-07
```

```
print(sum(resid(model)^2))  # the squared sum of residuals

## [1] 65.39269

print(confint(model))       # confidence intervals

## Waiting for profiling to be done...

##          2.5%     97.5%
## d0  -6.038315 15.999559
## v0 -10.749091  6.899734
## a    8.244167 14.245927
```

2.5 The Model Performance

The choice of a model is a complicated task. One needs to consider:

- Simple predictive Model Quality (i.e. Height of the lift curve / AUC)
- Generalization ability of the Model (Difference of model quality between Creation set and Test set / Does the model overfit?)
- Explanatory Power of the model (Does the model make sense of the data? Can it explain something?)
- Model Stability (Whats the confidence interval of the model on the lift curves?)
- Is the model robust against the erosion of time?

In this section we will focus on the first issue: the intrinsic predictive quality of the model

2.5.1 R-Squared

R-squared

R-squared is the "fraction of variance explained by the model"; in a linear model of the type $y = \sum_{l=1}^{M} a_l x_l + b$, we define R^2 as

$$R^2 := \frac{\sum_{k=1}^{N} (\hat{y}_k - \bar{y})^2}{\sum_{k=1}^{N} (y_k - \bar{y})^2}$$

```
m <- lm(data=mtcars,formula=mpg~wt)
summary(m)

##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528,Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10

summary(m)$r.squared

## [1] 0.7528328
```

hint: get more information via `help(summary.lm)`

Exercise: model performance for linear regression

Question

Calculate and discuss the model performance of the linear models made in Chapter 2.4.1 on page 74 and in Chapter 2.4.2 on page 77.

.....

.....

.....

.....

Exercise: model performance for linear regression

Question

Use the dataset mtcars (from the library MASS), and try to find the model that best explains the consumption (mpg).

.....

.....

.....

.....

2.5.2 AUC or Gini for logistic regression

A logistic model for surviving the Titanic disaster

the data is already divided in a set to train the model (titanic_train) and a set to test the model (titanictest). The data was supplied by the website `https://www.kaggle.com/c/titanic/data`.

```
# if necessary: install.packages("titanic")
library(titanic)
library(pROC)

## Type 'citation("pROC")' for a citation.
## ## Attaching package:  'pROC'
## The following objects are masked from 'package:stats':## ##    cov,
smooth, var

d <-titanic_train
str(d)   # explore the data-set

## 'data.frame': 891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bra
##  $ Sex        : chr  "male" "female" "female" "female" ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : chr  "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : chr  "" "C85" "" "C123" ...
##  $ Embarked   : chr  "S" "C" "S" "S" ...

m1 <- glm(formula=Survived ~ Sex + Pclass, family = binomial,
          data = d)
```

```
log_reg <- function(df, size=10) {
  N <- nrow(df)
  size=10

  df <- df[sample(N),]

  num <- floor(N/size)
  rest <- N - num * size
  ncv <- cumsum(c(rep(size,num), rest))

  predictions <- data.frame(survived = df$survived, pred = NA)

  for(n in ncv) {
```

```
    v <- rep(TRUE, N)
    v[(n-size+1):n] <- FALSE

    lr <- glm(survived ~ ., data = df[v,],
              family = binomial(logit))
    predictions[!v,"pred"] <- predict(lr, newdata=df[!v,],
                                      type="response")
  }

  return(predictions)
}
```

Make predictions

```
# copy the function log_red from:
# https://github.com/joyofdata/joyofdata-articles/blob/master
#       /roc-auc/log_reg.R

d1 <- data.frame(cbind(d$Survived,d$Sex,d$Pclass))
names(d1) <- c("survived","sex","class")

predictions <- log_reg(d1, size=10)
str(predictions)

## 'data.frame': 891 obs. of  2 variables:
##  $ survived: Factor w/ 2 levels "0","1": 2 1 2 1 1 1 1 1 1 1 ...
##  $ pred    : num  0.9108 0.6096 0.9108 0.0978 0.6096 ...
```

Plot the distributions of the test

```
# copy: https://github.com/joyofdata/joyofdata-articles/blob/
# master/roc-auc/plot_pred_type_distribution.R
library(ggplot2)
plot_pred_type_distribution(predictions, 0.7)
```

If we consider survival as a positive (1) and death as a negative (0), then the plot illustrates the tradeoff we face upon choosing a threshold. As we remember from Figure 2.4 on page 79 the probability of a positive outcome slowly increases and there will be a grey zone. So we have to decide where we put the cutoff. If we increase the threshold the number of false positive (FP) results is lowered, however, the number of false negative (FN) results increases. In order to help us with that decision, we can consider a cost function that sums the cost of each false positive and each false negative.

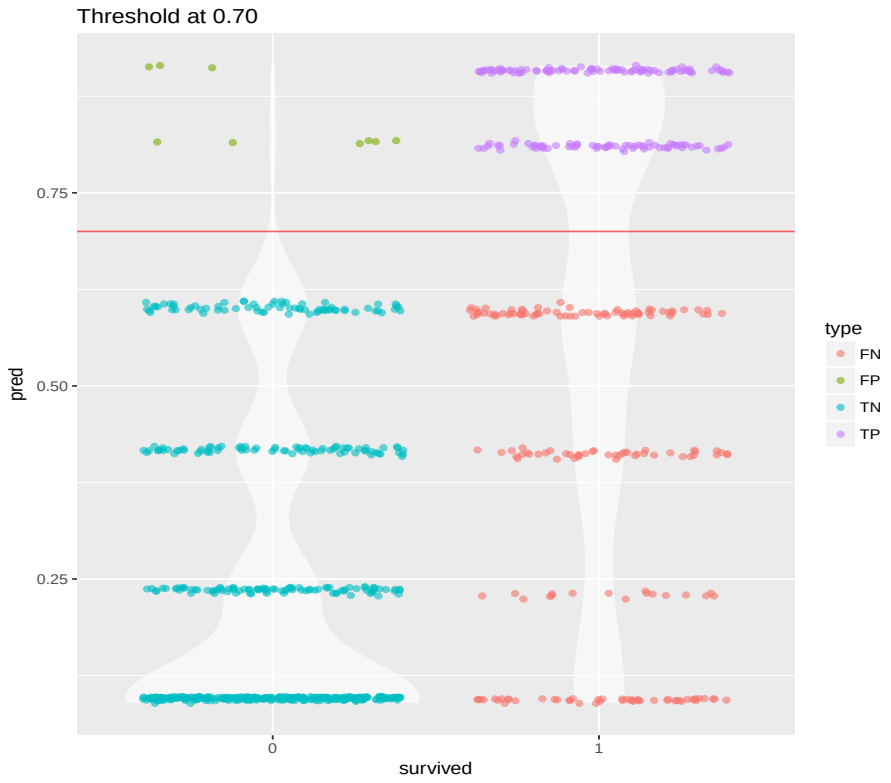


Figure 2.6: *A visualization of the predictions of a model for surviving the Titanic disaster. Note that FN = false negative, TN = True negative, FP = false positive, TP = true positive*

Receiver Operating Characteristic

The question of how to balance false positives and false negatives (depending on the cost/consequences of either mistake) was first addressed in a rigorous framework during World War II in context of interpretation of radar signals for identification of enemy air planes. In order to visualize and quantify the impact of a threshold on the FP/FN-tradeoff the ROC curve was used. The ROC curve is the interpolated curve made of points whose coordinates are functions of the threshold: threshold $= \theta \in \mathbb{R}$, here $\theta \in [0, 1]$

$$ROC_x(\theta) = FPR(\theta) = \frac{FP(\theta)}{FP(\theta) + TN(\theta)} = \frac{FP(\theta)}{\#N}$$
$$ROC_y(\theta) = TPR(\theta) = \frac{TP(\theta)}{FN(\theta) + TP(\theta)} = \frac{FP(\theta)}{\#P} = 1 - \frac{FN(\theta)}{\#P} = 1 - FNR(\theta)$$

In terms of hypothesis tests where rejecting the null hypothesis is considered a positive result the FPR (false positive rate) corresponds to the Type I error, the FNR (false negative rate) to the Type II error and (1 FNR) to the power. So the ROC for above distribution of predictions would be:

```
# https://github.com/joyofdata/joyofdata-articles/blob/master/
# roc-auc/calculate_roc.R
roc <- calculate_roc(predictions, 1, 2, n = 100)
```

Receiver Operating Characteristic II

```
# https://github.com/joyofdata/joyofdata-articles/blob/master/
# roc-auc/plot_roc.R
library(grid)
library(gridExtra)
plot_roc(roc, 0.7, 1, 2)
```

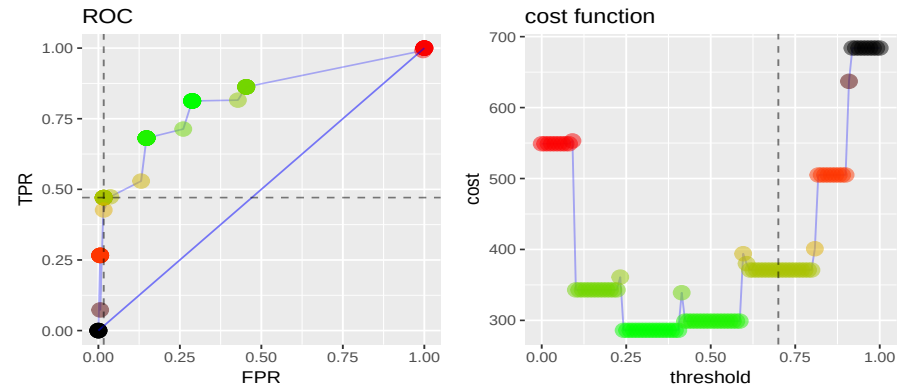
The dashed lines indicate the location of the (FPR, TPR) corresponding to a threshold of 0.7. Note that the low corner (0,0) is associated with a threshold of 1 and the top corner (1,1) with a threshold of 0.

The cost function and the corresponding colouring of the ROC points illustrate that an optimal FPR and TPR combination is determined by the associated cost. Depending on the use case false negatives might be more costly than false positive or vice versa. Here I assumed a cost of 1 for FP cases and a cost of 2 for FN cases.

Area Under (ROC) Curve

The optimal point on the ROC curve is (FPR, TPR) = (0,1). No false positives and all true positives. So the closer we get there the better. The second essential observation is that the curve is by definition monotonically increasing.

$$\theta' \implies TPR(\theta) \leq TPR(\theta')$$



hreshold at 0.70 − cost of FP = 1, cost of FN = 2

Figure 2.7: *The ROC and cost curves for our model on surviving the Titanic disaster.*

This inequality can be easily checked by looking at the first plot by mentally pushing the threshold (red line) up and down; it implies the monotonicity. Furthermore any reasonable models ROC is located above the identity line as a point below it would imply a prediction performance worse than random (in that case, simply inverting the predicted classes would bring us to the sunny side of the ROC space).

All those features combined make it apparently reasonable to summarize the ROC into a single value by calculating the area of the convex shape below the ROC curve this is the AUC. The closer the ROC gets to the optimal point of perfect prediction the closer the AUC gets to 1.

```
# AUC for the example
library(pROC)
auc(predictions$survived, predictions$pred)

## Area under the curve: 0.7844
```

Exercise: model performance for logistic regression

Question

Calculate and discuss the model performance of the linear models made in Chapter 2.4.1 on page 74 and in Chapter 2.4.2 on page 77.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2.6 Distributions

Distribution functions in R

The names of the functions related to statistical distributions in R are composed of two sections: the first letter refers to the function (see below) and the remainder is the distribution name.

- `d`: the pdf (probability density function)
- `p`: the cdf (cumulative probability density function)
- `q`: the quantile function
- `r`: the random number generator

distribution	R-name	distribution	R-name
normal	norm	weibull	weibull
exponential	exp	binomial	binom
log-normal	lnorm	negative binomial	nbinom
logistic	logis	χ^2	chisq
geometric	geom	uniform	unif
poisson	pois	gamma	gamma
t	t	cauchy	cauchy
f	f	hypergeometric	hyper
beta	beta		

Table 2.1: *Common distributions and their names in R.*

2.6.1 Normal Distribution

In a random collection of data from independent sources, usually the data is distributed normal. This distribution is also called the Gaussian Distribution.

When plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve is the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph.

The Normal Distribution in R

R has four in built functions to generate normal distribution. They are described below.

- `dnorm(x, mean, sd)`: the height of the probability distribution
- `pnorm(x, mean, sd)`: the cumulative distribution function (the probability of the observation to be lower than x)

- `qnorm(p, mean, sd)`: gives a number whose cumulative value matches the given probability value p
- `rnorm(n, mean, sd)`: generates normally distributed variables

with

- x: a vector of numbers
- p: a vector of probabilities
- n: the number of observations(sample size)
- $mean$: the mean value of the sample data (default is zero)
- sd: the standard deviation (default is 1).

Illustrating the normal distribution

```
obs <- rnorm(600,10,3)
hist(obs,col="yellow",freq=FALSE)
x <- seq(from=0,to=20,by=0.001)
lines(x, dnorm(x,10,3),col="blue",lwd=4)
```

Is the SP500 normally distributed?

```
library(MASS)
hist(SP500,col="yellow",freq=FALSE,border="red")
x <- seq(from=-5,to=5,by=0.001)
lines(x, dnorm(x,mean(SP500),sd(SP500)),col="blue",lwd=2)
```

QQ-plots

```
library(MASS)
qqnorm(SP500,col="red")
qqline(SP500,col="blue")
```

2.6.2 Binomial Distribution

The binomial distribution model deals with finding the probability of an event which has only two possible outcomes.

For example the probability of finding exactly 6 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

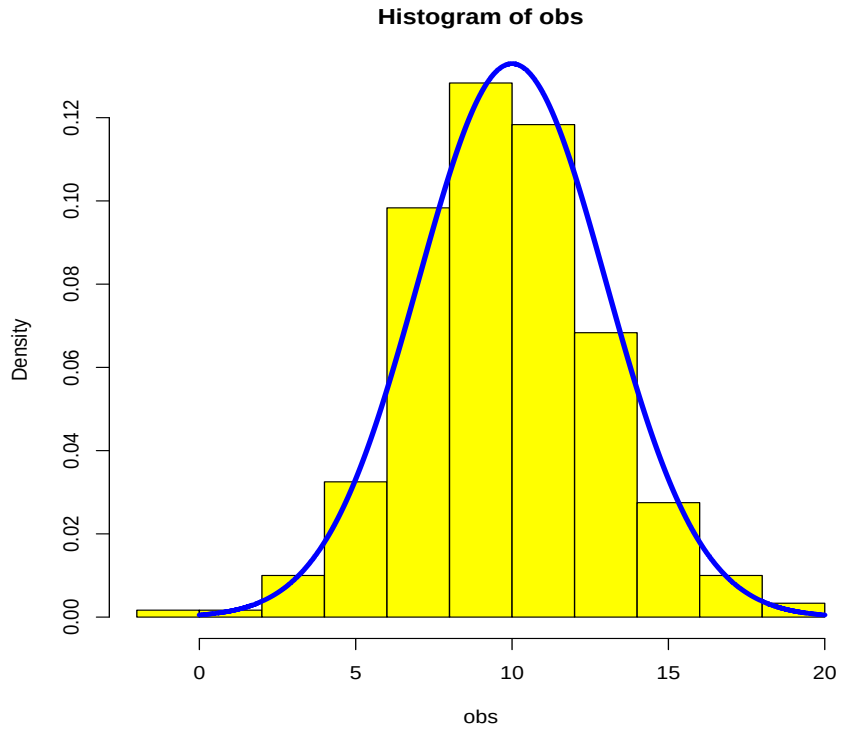


Figure 2.8: *A comparison between a set of random numbers drawn from the normal distribution (yellow) and the theoretical shape of the normal distribution in blue.*

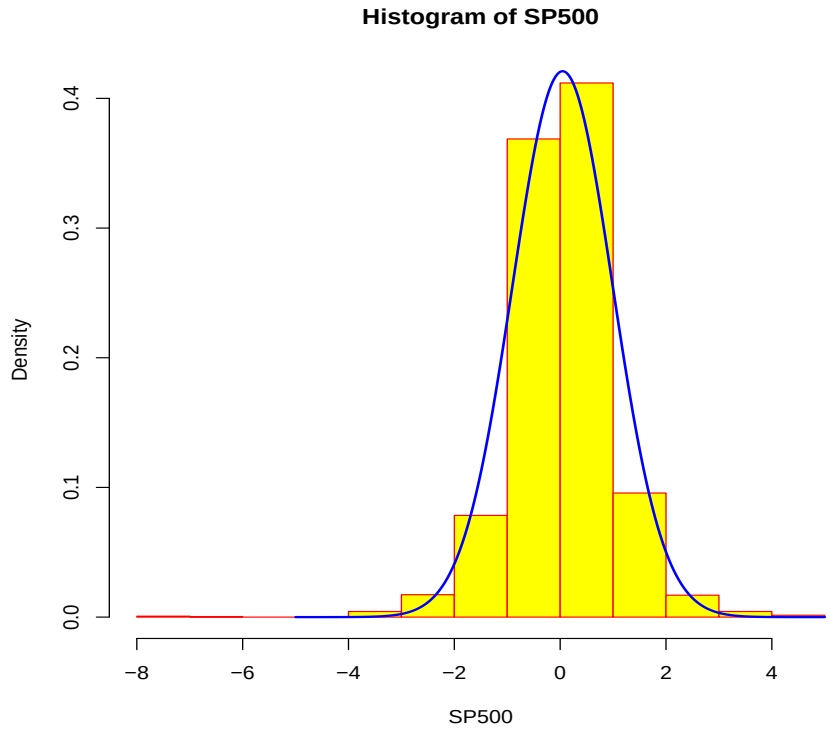


Figure 2.9: *The same plot for the returns of the SP500 index seems acceptable, though there are outliers (where the normal distribution converges fast to zero).*

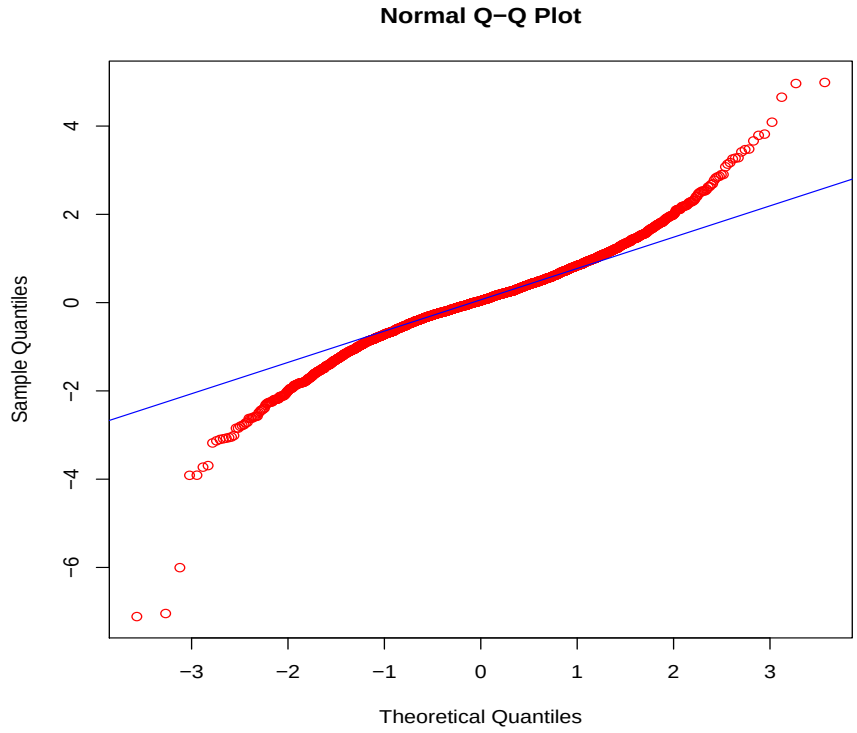


Figure 2.10: *A qq-plot is a better way to judge if a a set of observations is normally distributed or not.*

The Binomial Distribution in R

R has four in-built functions to generate binomial distribution. They are described below.

- `dbinom(x, size, prob)`: the density function
- `pbinom(x, size, prob)`: the cumulative probability of an event
- `qbinom(p, size, prob)`: gives a number whose cumulative value matches a given probability value
- `rbinom(n, size, prob)`: generates random variables following the binomial distribution

Following parameters are used:

- x: a vector of numbers
- p: is a vector of probabilities
- n: the number of observations
- $size$: the number of trials
- $prob$ the probability of success of each trial

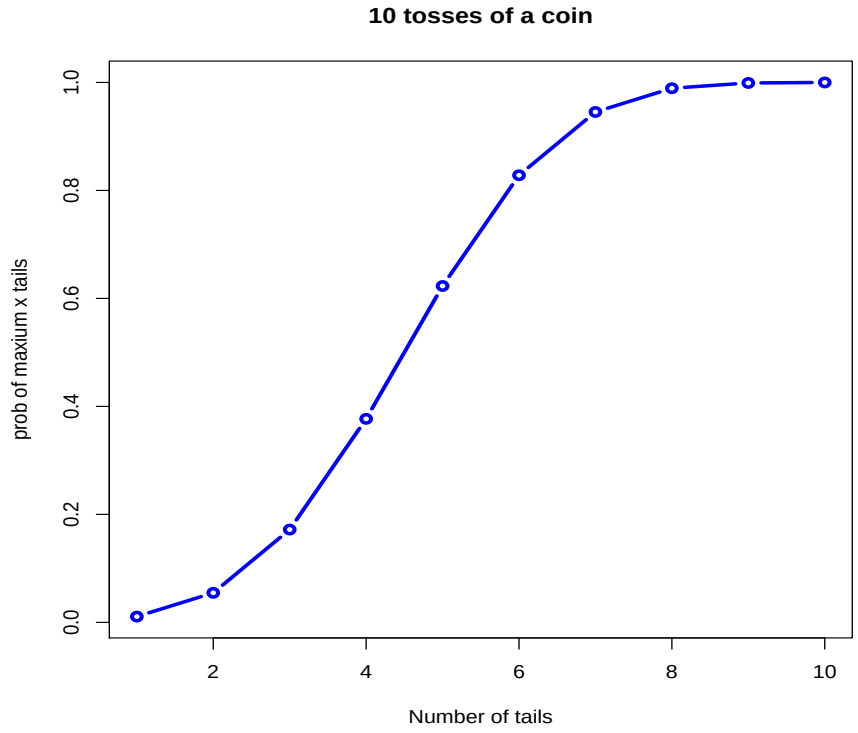


Figure 2.11: *The probability to get maximum x tails with the binomial distribution.*

An example

```
# Probability of getting 5 or less heads from 10 tosses of
# a coin.
pbinom(5,10,0.5)

## [1] 0.6230469

# visualize this for one to 10 numbers of tosses
x <- 1:10
y <- pbinom(x,10,0.5)
plot(x,y,type="b",col="blue", lwd=3,
    xlab="Number of tails",
    ylab="prob of maxium x tails",
    main="10 tosses of a coin")
```

```
# How many heads should we at least expect (with a probability
# of 0.25) when a coin is tossed 10 times.
qbinom(0.25,10,1/2)

## [1] 4
```

Generate random variables

```
# Find 20 random numbers of tails from and event of 10 tosses
# of a coin
rbinom(20,10,.5)

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

2.7 Analysis of Covariance

We use Regression analysis to create models which describe the effect of variation in predictor variables on the response variable. Sometimes, if we have a categorical variable with values like Yes/No or Male/Female etc. The simple regression analysis gives multiple results for each value of the categorical variable. In such scenario, we can study the effect of the categorical variable by using it along with the predictor variable and comparing the regression lines for each level of the categorical variable. Such an analysis is termed as Analysis of Covariance also called as ANCOVA. Example

Consider the R built in data set mtcars. In it we observer that the field "am" represents the type of transmission (auto or manual). It is a categorical variable with values 0 and 1. The miles per gallon value(mpg) of a car can also depend on it besides the value of horse power("hp").

We study the effect of the value of "am" on the regression between "mpg" and "hp". It is done by using the aov() function followed by the anova() function to compare the multiple regressions. Input Data

Create a data frame containing the fields "mpg", "hp" and "am" from the data set mtcars. Here we take "mpg" as the response variable, "hp" as the predictor variable and "am" as the categorical variable.

ANCOVA Analysis

We create a regression model taking "hp" as the predictor variable and "mpg" as the response variable taking into account the interaction between "am" and "hp". Model with interaction between categorical variable and predictor variable.

```
# Get the dataset.
input <- mtcars

# Create the regression model.
result <- aov(mpg~hp*am,data = input)
print(summary(result))

##              Df Sum Sq Mean Sq F value   Pr(>F)
## hp            1  678.4   678.4  77.391 1.50e-09 ***
## am            1  202.2   202.2  23.072 4.75e-05 ***
## hp:am         1    0.0     0.0   0.001    0.981
## Residuals    28  245.4     8.8
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This result shows that both horse power and transmission type has significant effect on miles per gallon as the p value in both cases is less than 0.05. But the interaction between these two variables is not significant as the p-value is more than 0.05. So, we can consider a model without these insignificant variables

A more robust model

```
# Get the dataset.
input <- mtcars

# Create the regression model.
result <- aov(mpg~hp+am,data = input)
print(summary(result))

##             Df Sum Sq Mean Sq F value   Pr(>F)
## hp           1  678.4   678.4   80.15 7.63e-10 ***
## am           1  202.2   202.2   23.89 3.46e-05 ***
## Residuals   29  245.4     8.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This result shows that both horse power and transmission type has significant effect on miles per gallon as the p value in both cases is less than 0.05. Comparing Two Models

anova

Now we can compare the two models to conclude if the interaction of the variables is truly in-significant. For this we use the anova() function.

```
# Get the dataset.
input <- mtcars

# Create the regression models.
result1 <- aov(mpg~hp*am,data = input)
result2 <- aov(mpg~hp+am,data = input)

# Compare the two models.
print(anova(result1,result2))

## Analysis of Variance Table
##
## Model 1: mpg ~ hp * am
## Model 2: mpg ~ hp + am
##   Res.Df    RSS Df  Sum of Sq     F Pr(>F)
## 1     28 245.43
## 2     29 245.44 -1 -0.0052515 6e-04 0.9806
```

As the p-value is greater than 0.05 we conclude that the interaction between horse power and transmission type is not significant. So the mileage per gallon will depend in a similar manner on the horse power of the car in both auto and manual transmission mode.

2.8 Time Series Analysis

2.8.1 Time Series in R

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame.

Time series

The time series object is created by using the `ts()` function.

```
ts(data = NA, start = 1, end = numeric(), frequency = 1,
        deltat = 1, ts.eps = getOption("ts.eps"), class = ,
        names = )
```

with

- `data`: a vector or matrix containing the values used in the time series.
- `start`: the start time for the first observation in time series.
- `end`: the end time for the last observation in time series.
- `frequency` the number of observations per unit time.
  - frequency = 12 pegs the data points for every month of a year
  - frequency = 4 pegs the data points for every quarter of a year
  - frequency = 6 pegs the data points for every 10 minutes of an hour
  - frequency = 24*6 pegs the data points for every 10 minutes of a day

Except the parameter "data" all other parameters are optional. To check if an object is a timeseries, we can use the function `is.ts()` and `as.ts(x)` will coerce the variable x into a time series object.

Time-series example

```
# Start from the SP500 data (from the MASS library)
# It is just a vector
head(SP500)

## [1] -0.2588908 -0.8650307 -0.9804139  0.4504321 -1.1856666 -0.6629097
```

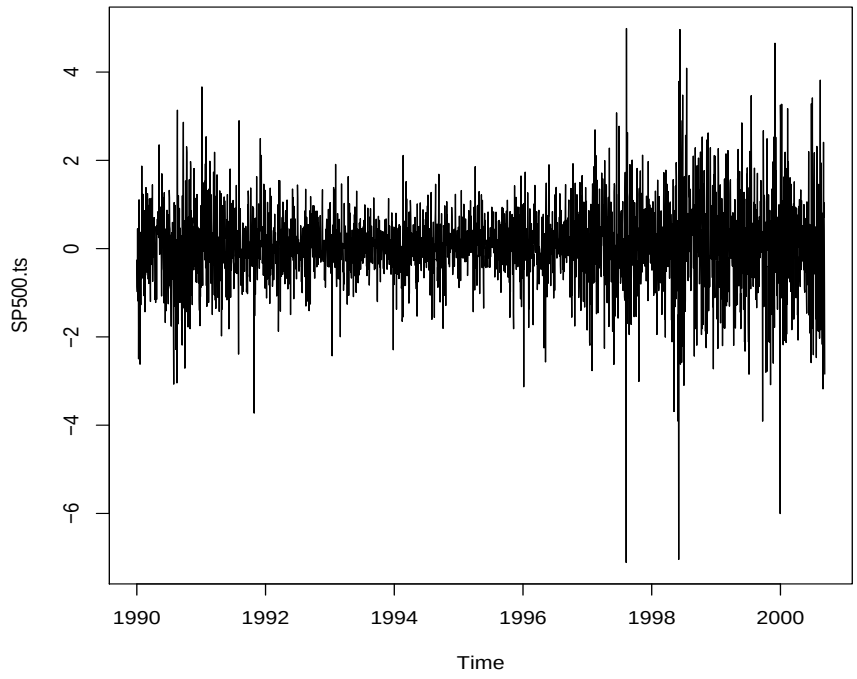


Figure 2.12: *The returns of the SP500 index in the 1990s.*

```
# Convert it to a time series object.
SP500.ts <- ts(SP500,start = c(1990,1),frequency = 260)

plot(SP500.ts)
```

Multiple time-series in one object

```
val = c(339.97)
for (k in 2:length(SP500)){
  val[k] = val[k-1] * (SP500[k-1] / 100 + 1)
}
# Convert both series to a matrix
M <- matrix(c(SP500,val),nrow=length(SP500))

# Convert the matrix to a time-series object
SP <- ts(M, start=c(1990,1),frequency=260)
colnames(SP) <- c("Daily Return in Pct","Value")
plot.ts(SP,type="l", main="SP500 in the 1990s")
```

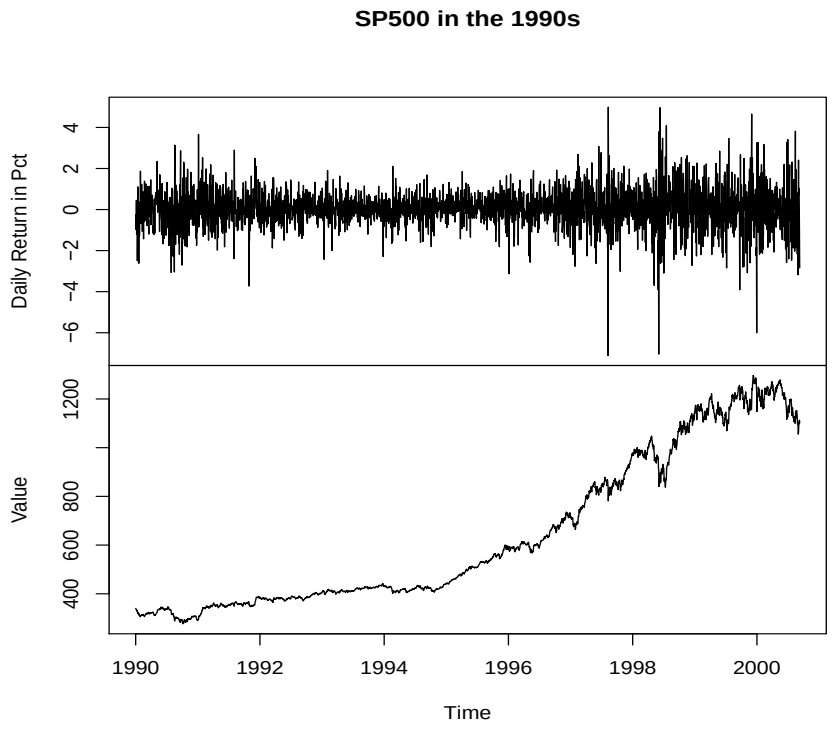


Figure 2.13: *The function plot.ts() will keep the the x-axis for both variables the same.*

2.8.2 Forecasting

Forecasting is the process of making predictions about the future and it is one of the main reasons to do statistics. The idea is that the the idea that the past holds valuable clues about the future and that by studying the past one can make reasonable suggestions about the future.

Every company will need to make forecasts in order to plan, every investor needs to forecast market and company performance in order to make decisions, etc. For example, if the profit grew the last 4 years every year between 15 and 20% it is reasonable to forecast a hefty growth rate based on this endogenous data. Of course this makes abstraction of exogenous data, such as an economic depression in the making.

No forecast is accurate and the only thing that we know for sure is that the future will hold surprises. It is of course possible to attach some degree of certainty to our forecast. This is referred to as "confidence intervals" and this is more valuable than a precise prediction.

There are a number of quantitative techniques that can be utilized to generate forecasts. While some are straightforward, others might for example incorporate exogenous factors and become necessarily more complex.

Our brain is in essence an enormous pattern-recognition machine that is so efficient

that it will even see patterns where there are none[1].

Typically one relies on the following key concepts:

- Trend: a trend is a long-term increase or decrease (despite short-term fluctuations). This is a specific form of auto-correlation (see below).
- Seasonality: a seasonal pattern is a pattern that repeats itself (eg. sales of ice-cream in the summer)
- Autocorrelation: this refers to the the pheneomena whereby values at time t are influenced by previous values. (R provides an autocorrelation plot that helps to find the proper lag structure and the nature of auto correlated values
- Stationary: a time series is said to be stationary if there is no systematic trend, no systematic change in variance, and if strictly periodic variations or seasonality do not exist
- Random: a time series that shows no specific pattern (eg. a random walk or Brownian motion). Though note that there can still be a systematic shift (a long term trend).

Moving average

In absence of a clear trend or as a starting point, the moving average is a versatile tool.

```
require(forecast)

## Loading required package:  forecast

g <- read.csv('../data/gdp/gdp_pol_sel.csv') # get some data
attach(g)    # the names of the data are now directly available
plot(year,GDP.per.capitia.in.current.USD,type='b',lwd=3)
```

```
# make the forecast with the movinga average (ma)
g.data <- ts(g$GDP.per.capitia.in.current.USD,start=c(1990))    # optional
g.movav = forecast(ma(g.data, order=3), h=5)

## Warning in ets(object, lambda = lambda, allow.multiplicative.trend
= allow.multiplicative.trend, :  Missing values encountered.  Using
longest contiguous portion of time series

# show the result:
plot(g.movav,col="blue",lw=4,
     main="Forecast of GDP per capita of Poland for moving average",
     ylab="income in current USD")
lines(year,GDP.per.capitia.in.current.USD,col="red",type='b')
```

[1] This refers to the bias "hot hand fallacy" – see eg. De Brouwer (2012)

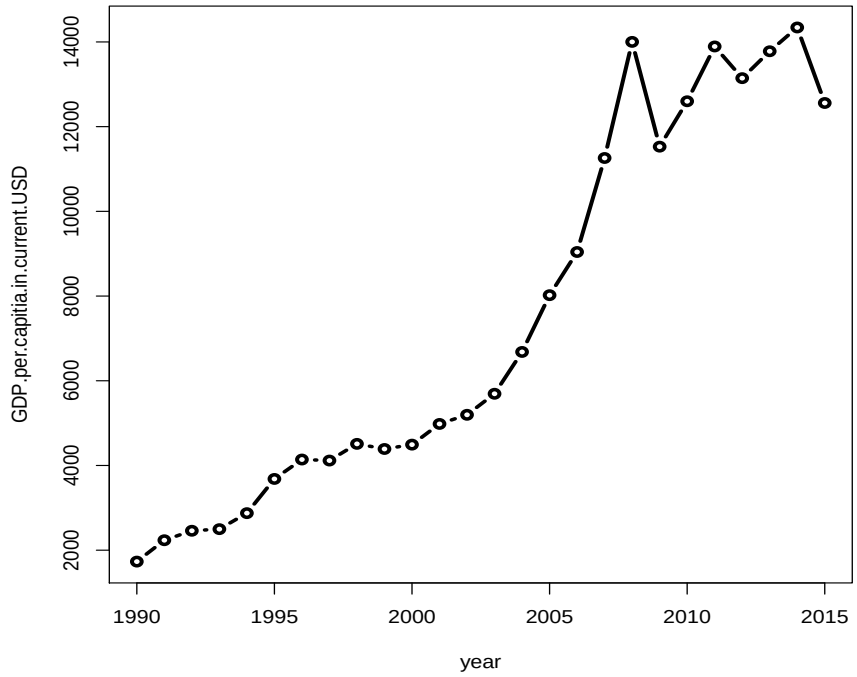


Figure 2.14: *A first plot to show the data before we start. This will allow us to select a suitable method for forecasting.*

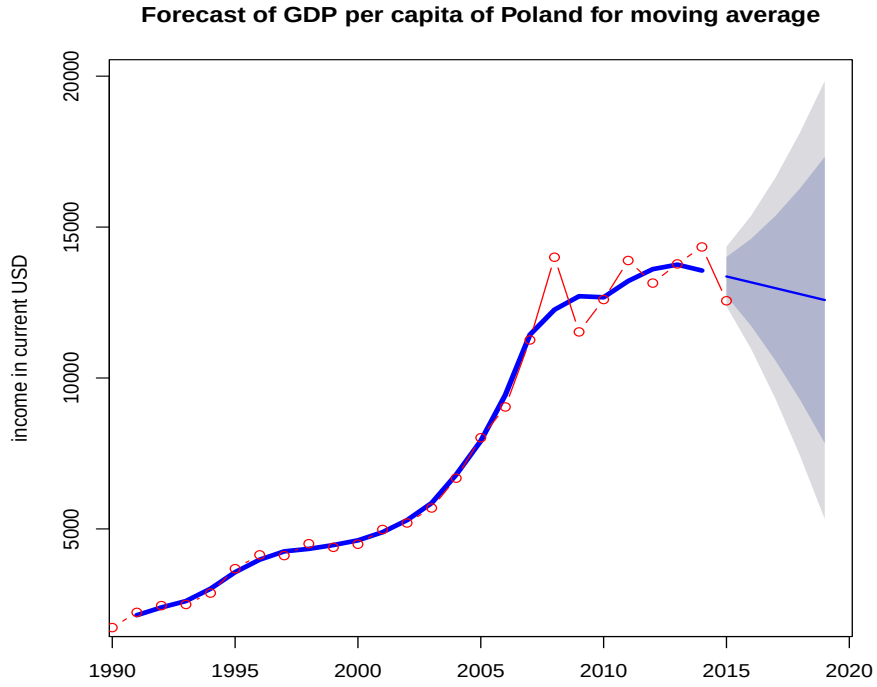


Figure 2.15: *Using the library forecast with a simple moving average.*

Testing the accuracy of forecasts – backtesting

```
# testing accuracy of the model by sampling
g.ts.tst <- ts(g.data[1:20],start=c(1990))
g.movav.tst <- forecast(ma(g.ts.tst,order=3),h=5)

## Warning in ets(object, lambda = lambda, allow.multiplicative.trend
= allow.multiplicative.trend, :  Missing values encountered.  Using
longest contiguous portion of time series

accuracy(g.movav.tst,g.data[22:26])

##                      ME      RMSE       MAE         MPE      MAPE
## Training set    28.8983  342.4069  220.7202  0.6165671  3.364013 0.370
## Test set     -1206.5907 1925.5699 1527.0191 -9.2928769 11.599210 2.564
##                     ACF1
## Training set -0.05974669
## Test set              NA

plot(g.movav.tst,col="blue",lw=4,
     main="Forecast of GDP per capita of Poland for moving average",
     ylab="income in current USD")
lines(year,GDP.per.capitia.in.current.USD,col="red",type='b')
```

Testing the accuracy of forecasts – backtesting

In the forecast package, there is an automatic forecasting function that will run through possible models and select the most appropriate model give the data. This could be an auto regressive model of the first oder (AR(1)), an ARIMA model with the right values for p, d, and q, or something else that is more appropriate.

```
train = ts(g.data[1:20],start=c(1990))
test  = ts(g.data[21:26],start=c(2010))
arma_fit <- auto.arima(train)
arma_forecast <- forecast(arma_fit, h = 6)
arma_fit_accuracy <- accuracy(arma_forecast, test)
arma_fit; arma_forecast; arma_fit_accuracy

## Series: train
## ARIMA(0,1,0) with drift
##
## Coefficients:
##          drift
##       515.5991
## s.e.  231.4786
##
## sigma^2 estimated as 1074618:  log likelihood=-158.38
```

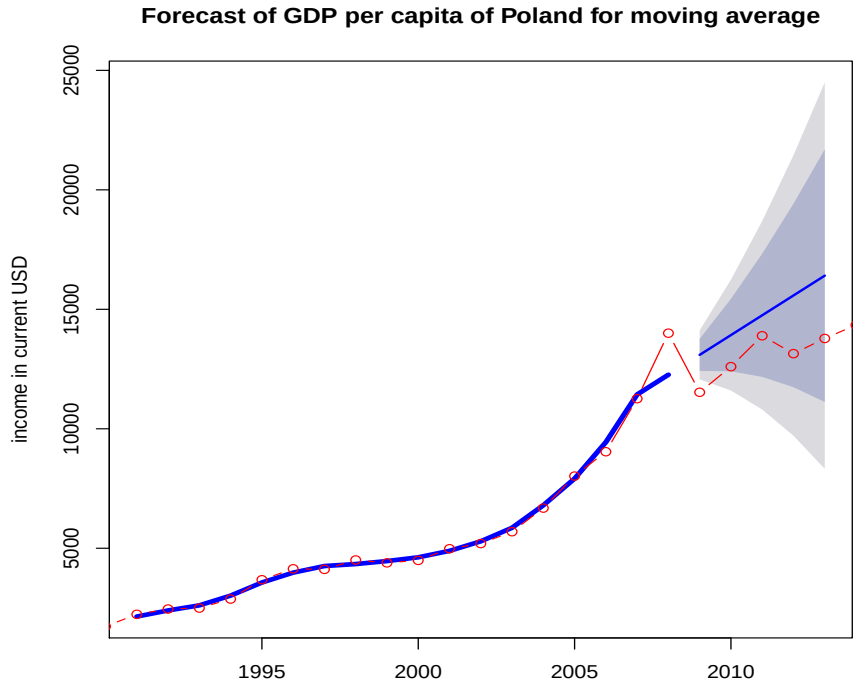


Figure 2.16: *A backtest for our forecast.*

```
## AIC=320.75   AICc=321.5   BIC=322.64
##      Point Forecast    Lo 80    Hi 80     Lo 95    Hi 95
## 2010       12043.19 10714.69 13371.70 10011.419 14074.97
## 2011       12558.79 10680.00 14437.58  9685.431 15432.15
## 2012       13074.39 10773.35 15375.43  9555.257 16593.52
## 2013       13589.99 10932.98 16247.00  9526.444 17653.54
## 2014       14105.59 11134.96 17076.22  9562.406 18648.77
## 2015       14621.19 11367.03 17875.35  9644.381 19598.00
##                       ME      RMSE      MAE        MPE     MAPE       M
## Training set  0.06078049  983.4411 602.2902 -2.3903997 8.585820 0.7612
## Test set     54.36338215 1036.0989 741.8024  0.1947719 5.668504 0.9375
##                     ACF1 Theil's U
## Training set -0.03066223        NA
## Test set      0.04756832 0.9928242

plot(arma_forecast, col="blue",lw=4,
     main="Forecast of GDP per capita of Poland for moving average",
     ylab="income in current USD")
lines(year,GDP.per.capitia.in.current.USD,col="red",type='b')
```

Note that ARIMA stands for autoregressive integrated moving average model. It is a generalization of an autoregressive moving average (ARMA) model. Both of these

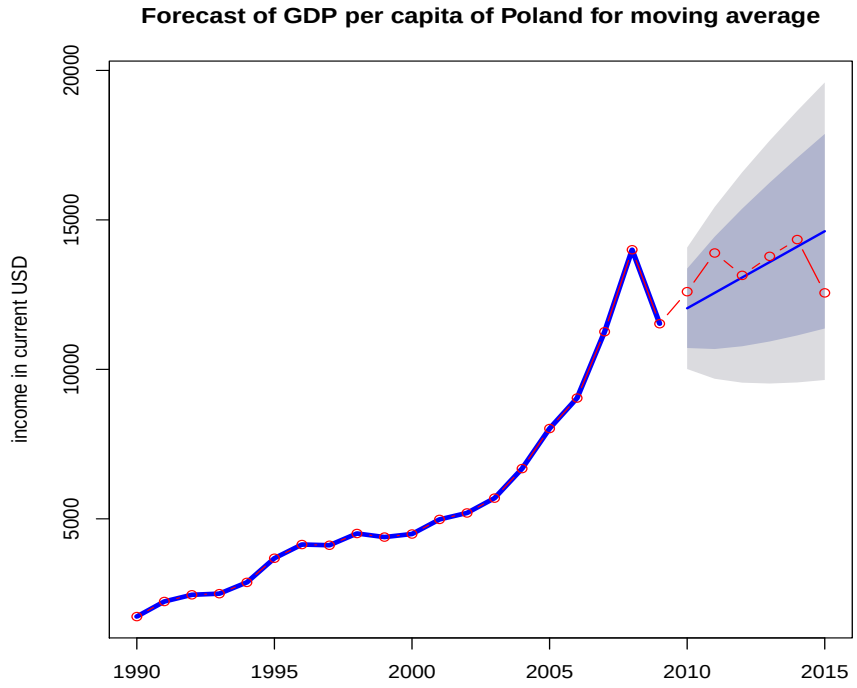


Figure 2.17: *Optimal moving average forecast.*

models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting). ARIMA models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the ”integrated” part of the model) can be applied one or more times to eliminate the non-stationarity.

Exponential smoothing

Exponential smoothing assigns higher weights to the most recent observations (the weight will decrease exponentially. The effect will be that a new dramatic event has a much faster impact, and that the “memory of it” will decrease exponentially.

```
g.exp <- ses(g.data,5,initial="simple")
g.exp

##      Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
## 2016       12558.87 11144.352 13973.39 10395.550 14722.19
## 2017       12558.87 10558.438 14559.30  9499.473 15618.27
## 2018       12558.87 10108.851 15008.89  8811.889 16305.85
## 2019       12558.87  9729.832 15387.91  8232.229 16885.51
## 2020       12558.87  9395.909 15721.83  7721.538 17396.20

plot(g.exp,col="blue",lw=4,
     main="Forecast of GDP per capita of Poland",
```

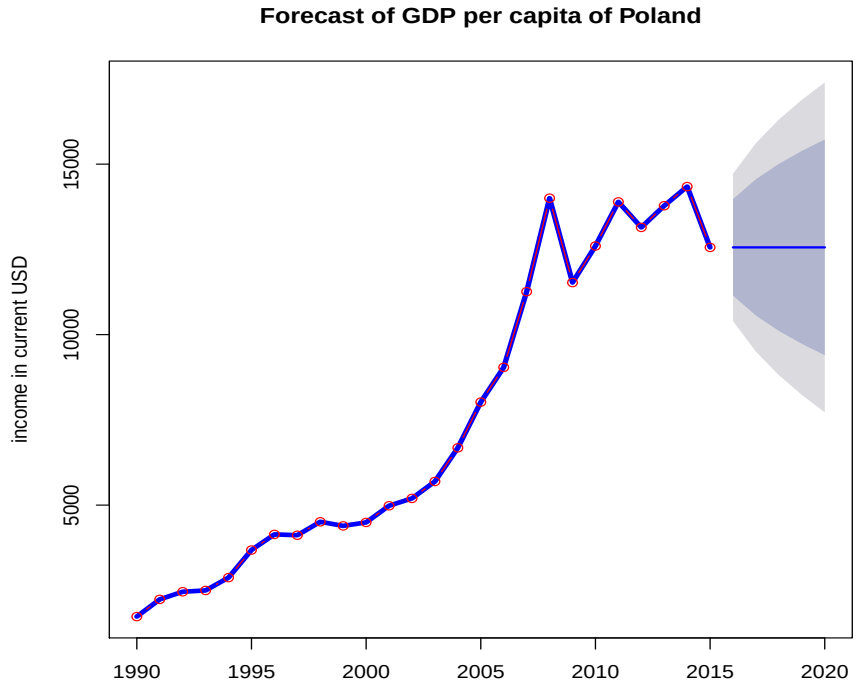


Figure 2.18: *Forecasting with an exponentially smoothed moving average.*

```
     ylab="income in current USD")
lines(year,GDP.per.capitia.in.current.USD,col="red",type='b')
```

Seasonal Decomposition

The Seasonal Trend Decomposition using Loess (STL) is an algorithm that was developed to help to divide up a time series into three components namely: the trend, seasonality and remainder. The methodology was presented by Robert Cleveland, William Cleveland, Jean McRae and Irma Terpenning in the Journal of Official Statistics in 1990. The STL is available within R via the `stl()` function.

Note that a series with multiplicative effects can often by transformed into series with additive effects through a `log` transformation: `new.ts <- log(old.ts)`.

```
# we use the data nottem
# Average Monthly Temperatures at Nottingham, 1920-1939
nottem.stl = stl(nottem, s.window="periodic")
plot(nottem.stl)
```

The four graphs are the original data, seasonal component, trend component and the remainder and this shows the periodic seasonal pattern extracted out from the original data and the trend that moves around between 47 and 51 degrees Fahrenheit. There is a bar at the right hand side of each graph to allow a relative comparison of

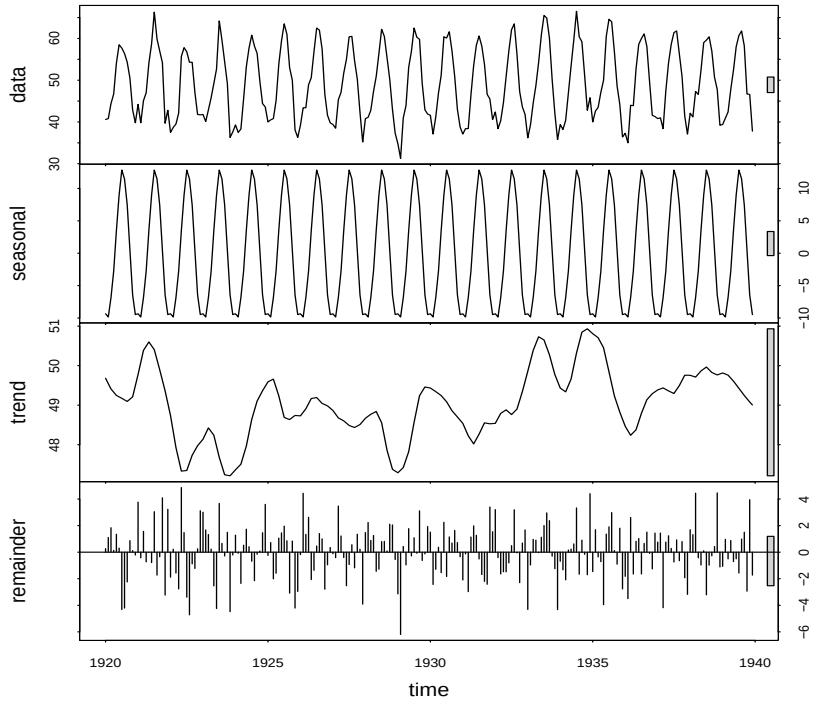


Figure 2.19: *Using the stl-function to decompose data in a seasonal part and a trend*

the magnitudes of each component. For this data the change in trend is less than the variation doing to the monthly variation.

Exponential Models

Both the HoltWinters() function in the base installation, and the ets() function in the forecast package, can be used to fit exponential models.

```
# simple exponential - models level
fit <- HoltWinters(g.data, beta=FALSE, gamma=FALSE)

# double exponential - models level and trend
fit <- HoltWinters(g.data, gamma=FALSE)

# triple exponential - models level, trend, and seasonal components
#fit <- HoltWinters(g.data)  # fails on our data-set

# predictive accuracy
library(forecast)
accuracy(forecast(fit,5))

##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -69.84485 1051.488 711.7743 -2.775476 9.016881 0.8422587
```

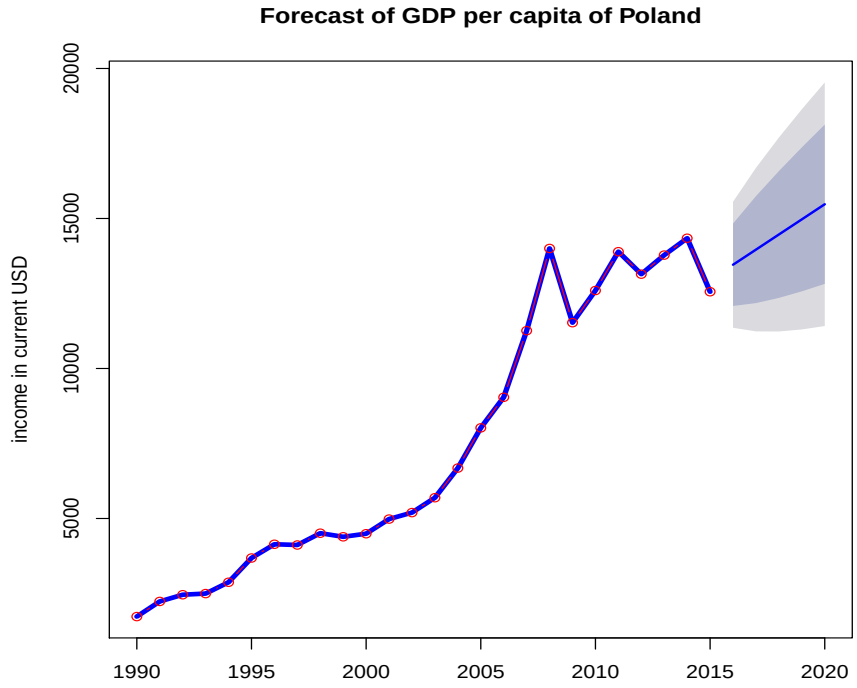


Figure 2.20: *The Holt-HoltWinters model fits an exponential trend. Here we plot the double exponential model.*

```
##                       ACF1
## Training set 0.008888197

# predict next 5 future values
forecast(fit, 5)

##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2016       13457.63 12084.15 14831.11 11357.07 15558.18
## 2017       13961.96 12179.69 15744.23 11236.21 16687.71
## 2018       14466.29 12352.87 16579.71 11234.10 17698.49
## 2019       14970.62 12571.33 17369.91 11301.23 18640.02
## 2020       15474.95 12820.41 18129.50 11415.17 19534.74

plot(forecast(fit, 5),col="blue",lw=4,
     main="Forecast of GDP per capita of Poland",
     ylab="income in current USD")
lines(year,GDP.per.capitia.in.current.USD,col="red",type='b')
```

```
# Use the Holt-Winters method for the temperatures
n.hw <- HoltWinters(nottem)
n.hw.fc <- forecast(n.hw,50)
```

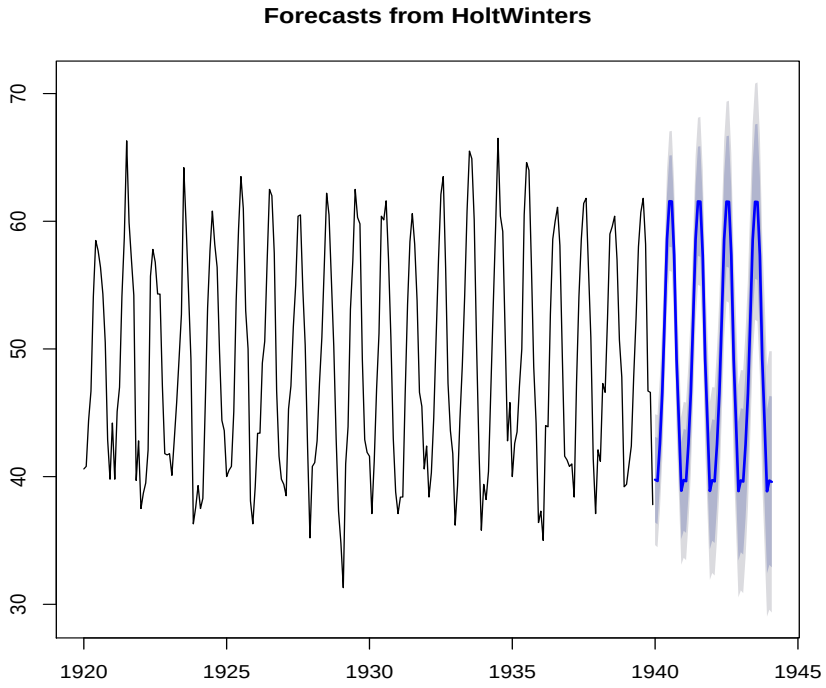


Figure 2.21: *The Holt-HoltWinters model applied to the temperatures in Nottingham.*

```
plot(n.hw.fc)
```

Exercise

Question

Use the moving average method on the the temperatures in Nottingham (nottam). Does it work? Which model would work better?

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2.9 Decision Tree

A decision tree can be thought of as slicing up a data-set into sub-sets that are more similar with respect to one variable. Typically this variable is a binary variable such as defaulting on a loan or paying it back, surviving or dying, good or bad, etc.

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions. A decision tree is related to some machine learning techniques and in data mining.

R has packages that provide tools to optimize trees and visualize the results.

There are a few packages in R that provide excellent results, even

The package "tree"

```
if (!any(grepl("tree",installed.packages()))){
  install.packages("tree")}
require(tree)

## Loading required package:  tree
```

Example for the tree()

```
frm <- l ~ cyl + disp + hp + drat + wt + qsec + am + gear
tr = tree(frm, data=mtcars)
summary(tr)

##
## Regression tree:
## tree(formula = frm, data = mtcars)
## Variables actually used in tree construction:
## [1] "cyl" "wt"  "hp"
## Number of terminal nodes:  5
## Residual mean deviance:  3.127 = 84.44 / 27
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.1040 -0.8384 -0.1382  0.0000  0.8012  4.6260

plot(tr); text(tr)
```

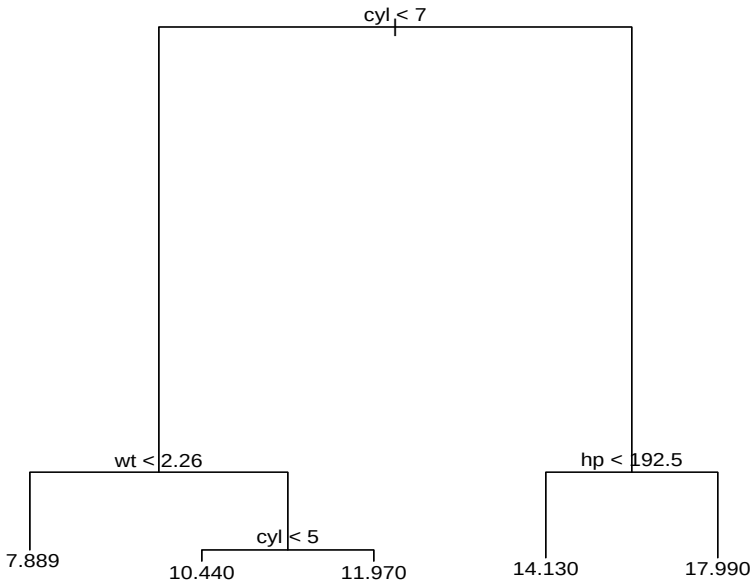


Figure 2.22: *This tree learns us how to estimate the fuel consumption of a car based on the number of cylinders, weight and horse power.*

2.10 Random Forest

Decision trees are a popular method for various machine learning tasks. Tree learning is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models.

However, they are seldom accurate. In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Random forests are very good in that it is an ensemble learning method used for classification and regression. It uses multiple models for better performance than just using a single tree model. In addition because many samples are selected in the process a measure of variable importance can be obtain and this approach can be used for model selection and can be particularly useful when forward/backward stepwise selection is not appropriate and when working with an extremely high number of candidate variables that need to be reduced.

Random Forest

```
library(randomForest)

## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
## ## Attaching package:  'randomForest'
## The following object is masked from 'package:gridExtra':## ##
combine
## The following object is masked from 'package:ggplot2':## ##    margin

mtcars$l <- NULL  # remove our variable
frm <- mpg ~ .
fit.rf = randomForest(frm, data=mtcars)
print(fit.rf)

##
## Call:
##  randomForest(formula = frm, data = mtcars)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 5.923383
##                     % Var explained: 83.17
```

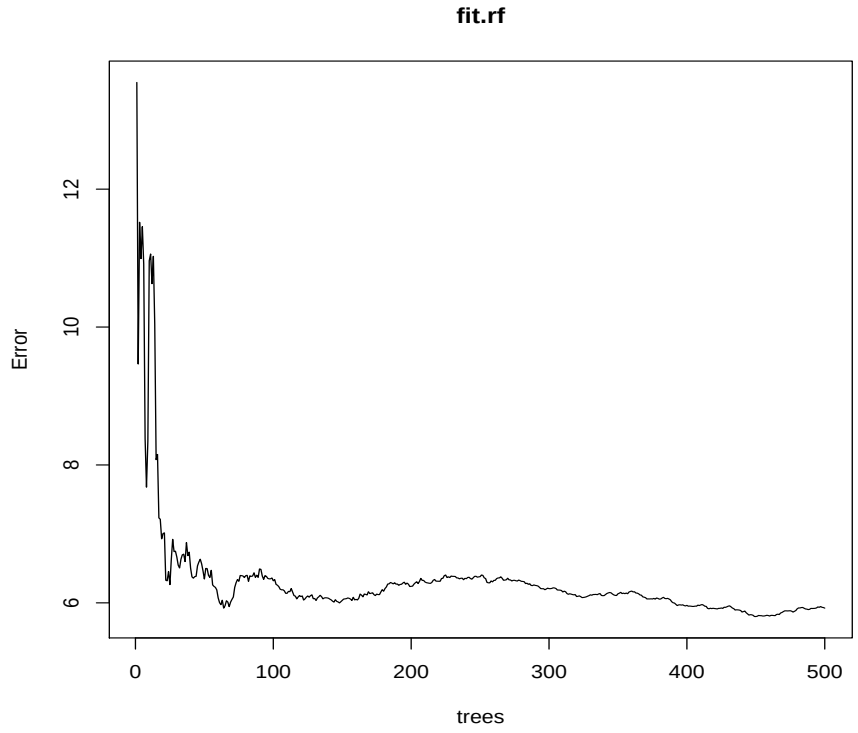


Figure 2.23: *The plot of a randomForest object shows how the model improves in function of the number of trees used.*

```
importance(fit.rf)

##       IncNodePurity
## cyl       185.91529
## disp      255.92784
## hp        178.55441
## drat       65.94957
## wt        255.80371
## qsec       29.55617
## vs         22.37504
## am         16.26893
## gear       16.84478
## carb       33.04909

plot(fit.rf)
```

```
plot( importance(fit.rf), lty=2, pch=16)
lines(importance(fit.rf))
```

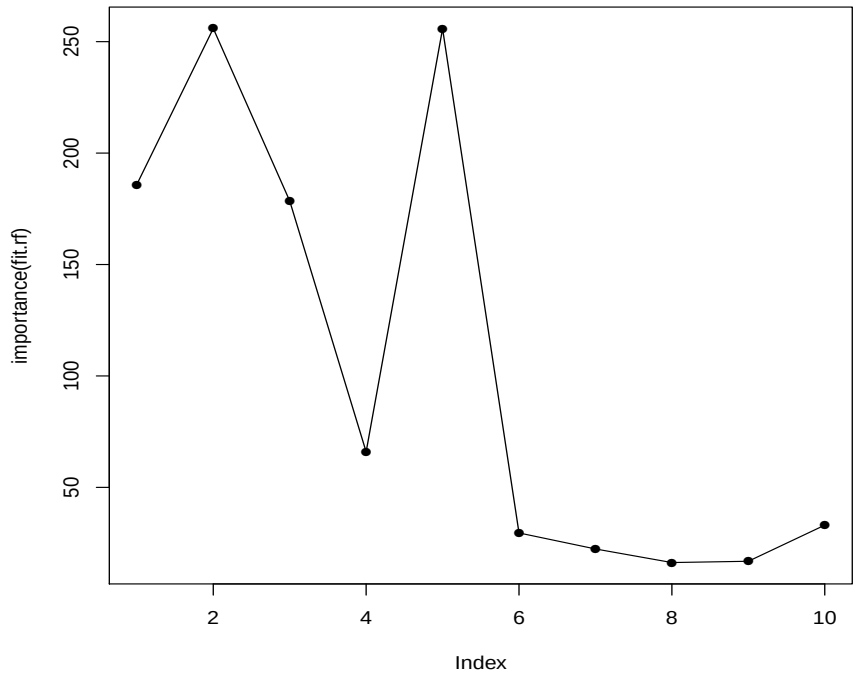


Figure 2.24: *The importance of each variable in the random-forest model.*

```
imp = importance(fit.rf)

impvar = rownames(imp)[order(imp[, 1], decreasing=TRUE)]
op = par(mfrow=c(1, 3))
for (i in seq_along(impvar)) {
partialPlot(fit.rf, mtcars, impvar[i], xlab=impvar[i],
main=paste("Partial Dependence on", impvar[i]))

# visualization of the RF
getTree(fit.rf, 1, labelVar=TRUE)
}
```

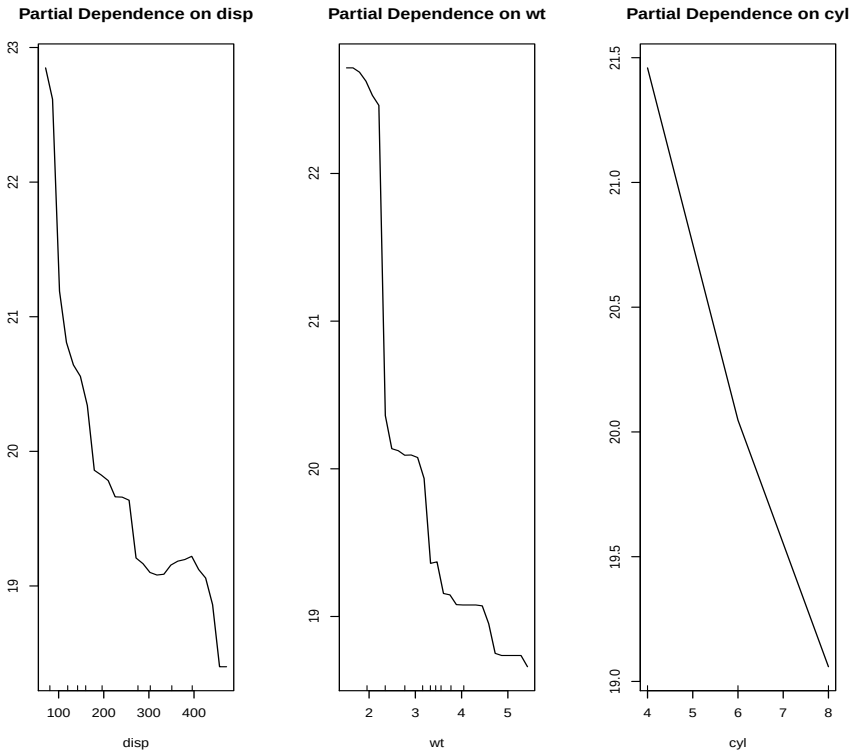


Figure 2.25: *Partial dependence on the variables.*

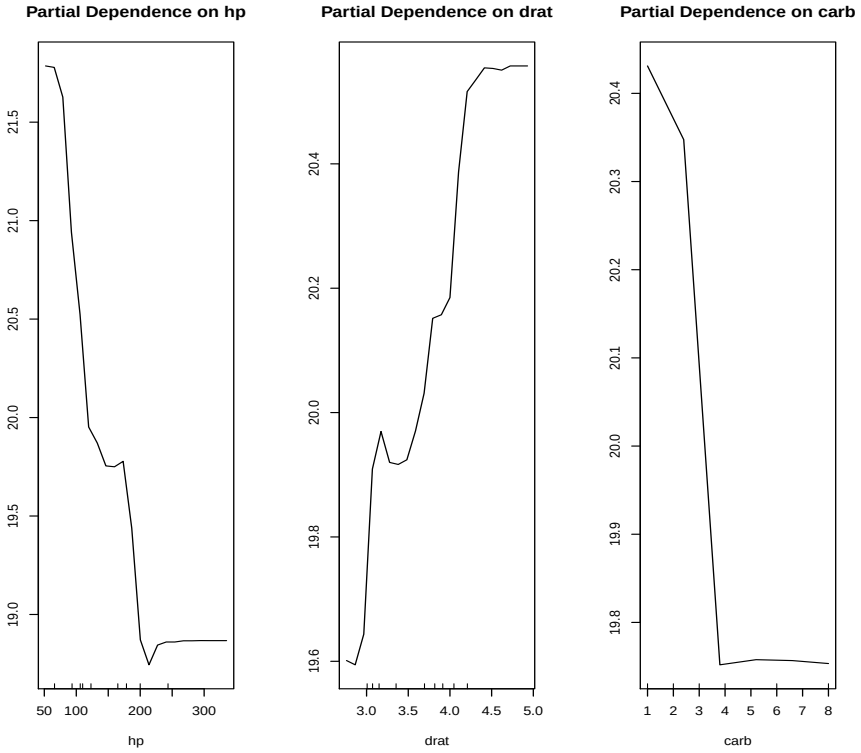


Figure 2.26: *The plot of a randomForest object shows how the model improves in function of the number of trees used.*

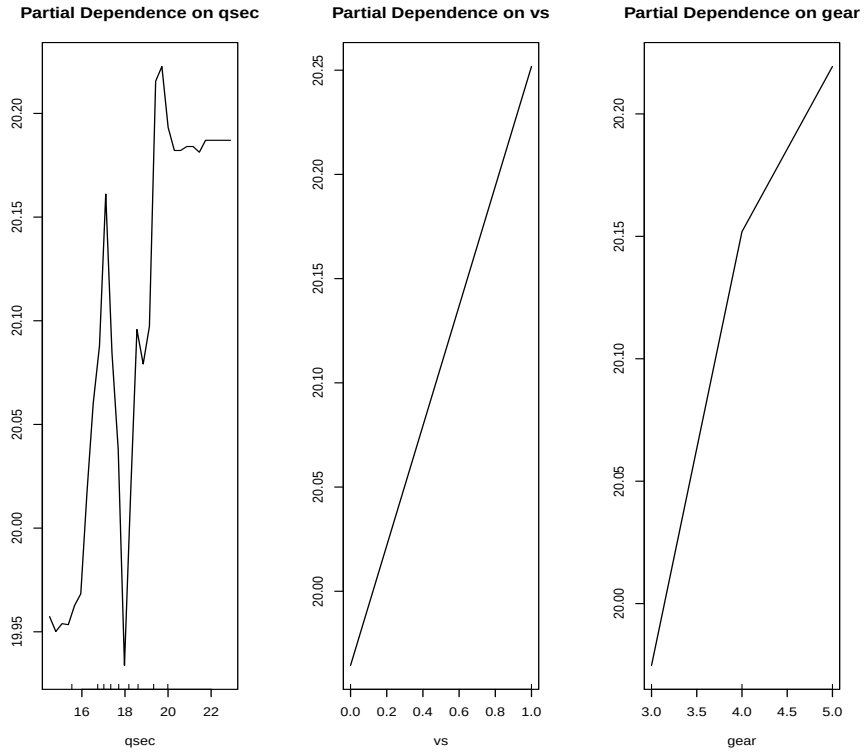


Figure 2.27: *The importance of each variable in the random-forest model.*

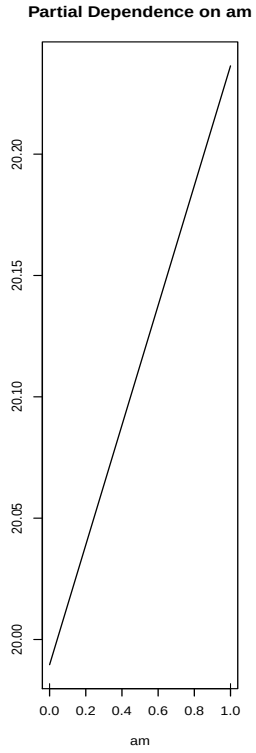


Figure 2.28: *Partial dependence on the variables.*

2.11 Chi Square Tests

Chi-Square test is a statistical method to determine if two categorical variables have a significant correlation between them. Both those variables should be from same population and they should be categorical like Yes/No, Male/Female, Red/Green etc.

For example, we can build a data set with observations on people's ice-cream buying pattern and try to correlate the gender of a person with the flavor of the ice-cream they prefer. If a correlation is found we can plan for appropriate stock of flavors by knowing the number of gender of people visiting.

Chi-Square test in R

Definition 18

`chisq.test(data)`
where `data` is the data in form of a table containing the count value of the variables

An example for chisq.test()

```
# we use the data-set mtcars from MASS
df <- data.frame(mtcars$cyl,mtcars$am)
chisq.test(df)

## Warning in chisq.test(df): Chi-squared approximation may be
incorrect

##
##  Pearson's Chi-squared test
##
## data:  df
## X-squared = 25.077, df = 31, p-value = 0.7643
```

conclusion: the p-value is higher than 0.05, so there is no significant correlation

2.12 Bootstrapping

Bootstrapping is the process of working on a subset of the data and calculating parameters of mean, spread and even calibrating models on that subset.

The reasons to do this is

- the whole data-set is too large to work with
- to test the robustness of the model we calibrate it on a subset of the data and see how it performs on another set of data.

Bootstrapping in R

The function "sample()" takes a sample from data

Definition 19 .:. sample() .:.

`sample(x, size, replace = FALSE, prob = NULL)` with

- `x`: either a vector of one or more elements from which to choose, or a positive integer.
- `size`: the number of items to select from x
- `replace`: set to TRUE if sampling is to be done with replacement
- `prob`: a vector of probability weights for obtaining the elements of the vector being sampled

Example: Sampling the SP500 data

```
# create the sample
SP500.sample <- sample(SP500,size=100)

# visualize the sample
par(mfrow=c(2,2))
hist(SP500,main="Histogram of all data",fr=FALSE,
     breaks=c(-9:5),ylim=c(0,0.4))
hist(SP500.sample,main="Histogram of the sample",
     fr=FALSE,breaks=c(-9:5),ylim=c(0,0.4))
boxplot(SP500,main="Boxplot of all data",ylim=c(-9,5))
boxplot(SP500.sample,main="Boxplot of the sample",ylim=c(-9,5))
```

```
mean(SP500)

## [1] 0.04575267
```

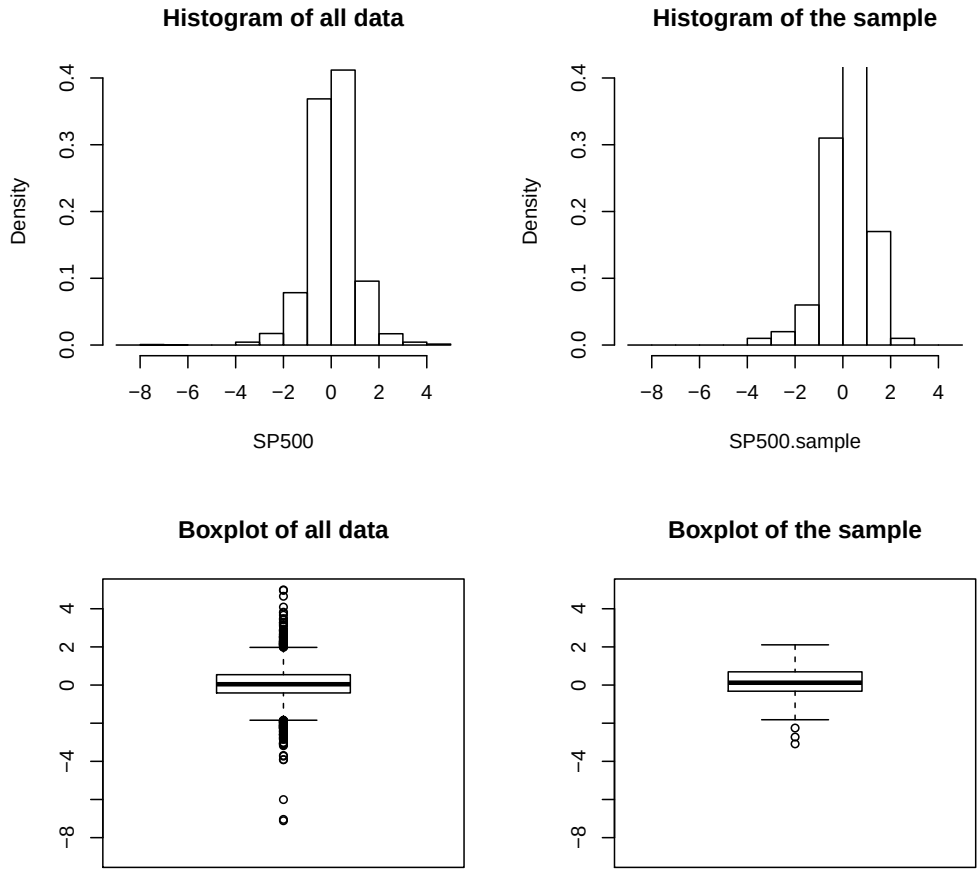


Figure 2.29: *Bootstrapping the returns of the S&P500 index.*

```
mean(SP500.sample)

## [1] 0.112804

sd(SP500)

## [1] 0.9477464

sd(SP500.sample)

## [1] 0.9111311
```

PART I .:. CHAPTER 3 .:.

Examples

3.1 Predicting Awards

An example from UCLA

after: *https://stats.idre.ucla.edu/r/dae/poisson-regression/*

```
myURL <- "https://stats.idre.ucla.edu/stat/data/poisson_sim.csv"
p <- read.csv(myURL)
p <- within(p, {
  prog <- factor(prog, levels=1:3,
  labels=c("General", "Academic", "Vocational"))
  id <- factor(id)
  })
summary(p)
```

```
##        id        num_awards          prog          math      
##  1      :  1   Min.   :0.00   General   : 45   Min.   :33.00  
##  2      :  1   1st Qu.:0.00   Academic  :105   1st Qu.:45.00  
##  3      :  1   Median :0.00   Vocational: 50   Median :52.00  
##  4      :  1   Mean   :0.63                    Mean   :52.65  
##  5      :  1   3rd Qu.:1.00                    3rd Qu.:59.00  
##  6      :  1   Max.   :6.00                    Max.   :75.00  
##  (Other):194
```

Each variable has 200 valid observations and their distributions seem quite reasonable. The unconditional mean and variance of our outcome variable are not extremely different. Our model assumes that these values, conditioned on the predictor variables, will be equal (or at least roughly so).

Checking the data

We can use the `tapply()` function to display the summary statistics by program type. The table below shows the average numbers of awards by program type and seems to suggest that program type is a good candidate for predicting the number of awards, our outcome variable, because the mean value of the outcome appears to

vary by prog. Additionally, the means and variances within each level of progthe conditional means and variancesare similar. A conditional histogram separated out by program type is plotted to show the distribution.

```
# Install the necessary packages
require(ggplot2)
require(sandwich)

## Loading required package:  sandwich

require(msm)  #also requires 'Matrix, survival, expm

## Loading required package:  msm

# Show the mean and standard deviation for each program
with(p, tapply(num_awards, prog, function(x) {
  sprintf("M (SD) = %1.2f (%1.2f)", mean(x), sd(x))
  }))

##                 General                Academic              Vocational
## "M (SD) = 0.20 (0.40)" "M (SD) = 1.00 (1.28)" "M (SD) = 0.24 (0.52)"

# Show the data
ggplot(p, aes(num_awards, fill = prog)) +
  geom_histogram(binwidth=.5, position="dodge")
```

Analysis methods you might consider

Below is a list of some analysis methods you may have encountered. Some of the methods listed are quite reasonable, while others have either fallen out of favor or have limitations.

- Poisson regression Poisson regression is often used for modeling count data. Poisson regression has a number of extensions useful for count models.

- Negative binomial regression Negative binomial regression can be used for over-dispersed count data, that is when the conditional variance exceeds the conditional mean. It can be considered as a generalization of Poisson regression since it has the same mean structure as Poisson regression and it has an extra parameter to model the over-dispersion. If the conditional distribution of the outcome variable is over-dispersed, the confidence intervals for Negative binomial regression are likely to be narrower as compared to those from a Poisson regression.

- Zero-inflated regression model Zero-inflated models attempt to account for excess zeros. In other words, two kinds of zeros are thought to exist in the data, true zeros and excess zeros. Zero-inflated models estimate two equations simultaneously, one for the count model and one for the excess zeros.

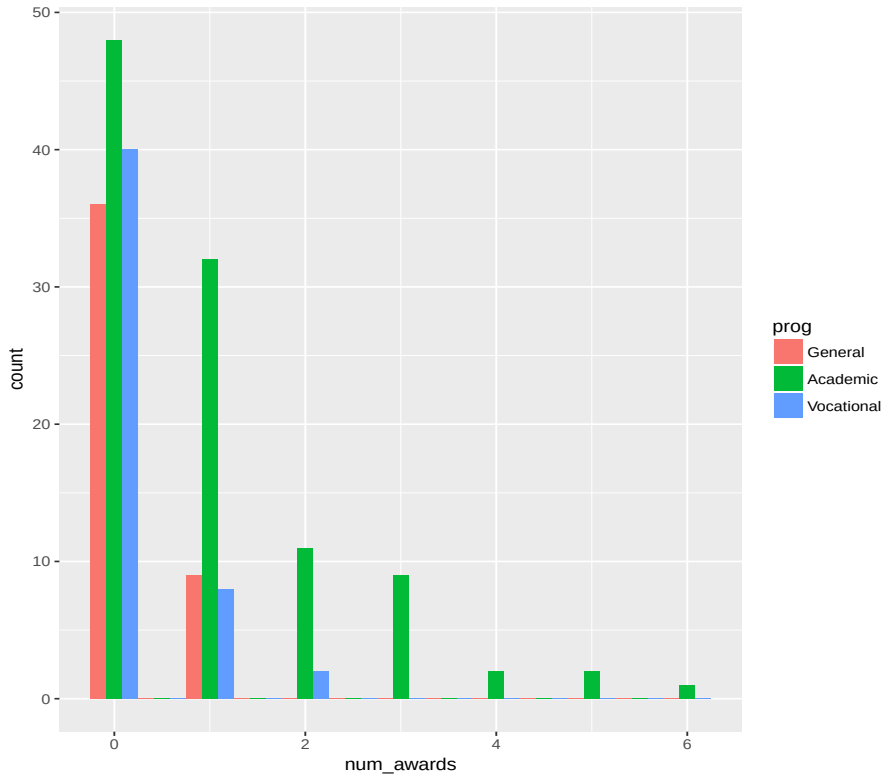


Figure 3.1: *The histograms for each program.*

- OLS regression Count outcome variables are sometimes log-transformed and analyzed using OLS regression. Many issues arise with this approach, including loss of data due to undefined values generated by taking the log of zero (which is undefined) and biased estimates.

At this point, we are ready to perform our Poisson model analysis using the glm function. We fit the model and store it in the object m1 and get a summary of the model at the same time.

Visualizing the data

```
# Poisson Regression:
m1 <- glm(num_awards ~ prog + math, family="poisson", data=p)
summary(m1)

##
## Call:
## glm(formula = num_awards ~ prog + math, family = "poisson", data = p)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2043  -0.8436  -0.5106   0.2558   2.6796
##
```

```
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -5.24712    0.65845  -7.969 1.60e-15 ***
## progAcademic    1.08386    0.35825   3.025  0.00248 **
## progVocational  0.36981    0.44107   0.838  0.40179
## math            0.07015    0.01060   6.619 3.63e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 287.67  on 199  degrees of freedom
## Residual deviance: 189.45  on 196  degrees of freedom
## AIC: 373.5
##
## Number of Fisher Scoring iterations: 6
```

Cameron and Trivedi (2009) recommended using robust standard errors for the parameter estimates to control for mild violation of the distribution assumption that the variance equals the mean. We use R package sandwich below to obtain the robust standard errors and calculated the p-values accordingly. Together with the p-values, we have also calculated the 95% confidence interval using the parameter estimates and their robust standard errors.

Covariance

```
cov.m1 <- vcovHC(m1, type="HC0")
std.err <- sqrt(diag(cov.m1))
r.est <- cbind(Estimate= coef(m1), "Robust SE" = std.err,
"Pr(>|z|)" = 2 * pnorm(abs(coef(m1)/std.err), lower.tail=FALSE),
LL = coef(m1) - 1.96 * std.err,
UL = coef(m1) + 1.96 * std.err)

r.est

##                  Estimate  Robust SE     Pr(>|z|)          LL
## (Intercept)    -5.2471244 0.64599839 4.566630e-16 -6.51328124 -3.98096
## progAcademic    1.0838591 0.32104816 7.354745e-04  0.45460476  1.71311
## progVocational  0.3698092 0.40041731 3.557157e-01 -0.41500870  1.15462
## math            0.0701524 0.01043516 1.783975e-11  0.04969947  0.09060
```

Now lets look at the output of function glm more closely.

- The output begins with echoing the function call. The information on deviance residuals is displayed next. Deviance residuals are approximately normally distributed if the model is specified correctly.In our example, it shows a little bit of skeweness since median is not quite zero.

- Next come the Poisson regression coefficients for each of the variables along with the standard errors, z-scores, p-values and 95% confidence intervals for the coefficients. The coefficient for math is .07. This means that the expected log count for a one-unit increase in math is .07. The indicator variable progAcademic compares between prog = Academic and prog = General, the expected log count for prog = Academic increases by about 1.1. The indicator variable prog.Vocational is the expected difference in log count ((approx .37)) between prog = Vocational and the reference group (prog = General).
- The information on deviance is also provided. We can use the residual deviance to perform a goodness of fit test for the overall model. The residual deviance is the difference between the deviance of the current model and the maximum deviance of the ideal model where the predicted values are identical to the observed. Therefore, if the residual difference is small enough, the goodness of fit test will not be significant, indicating that the model fits the data. We conclude that the model fits reasonably well because the goodness-of-fit chi-squared test is not statistically significant. If the test had been statistically significant, it would indicate that the data do not fit the model well. In that situation, we may try to determine if there are omitted predictor variables, if our linearity assumption holds and/or if there is an issue of over-dispersion.

```
with(m1, cbind(res.deviance = deviance, df = df.residual,
  p = pchisq(deviance, df.residual, lower.tail=FALSE)))

##      res.deviance  df         p
## [1,]     189.4496 196 0.6182274
```

We can also test the overall effect of prog by comparing the deviance of the full model with the deviance of the model excluding prog. The two degree-of-freedom chi-square test indicates that prog, taken together, is a statistically significant predictor of num_awards.

```
## update m1 model dropping prog
m2 <- update(m1, . ~ . - prog)
## test model differences with chi square test
anova(m2, m1, test="Chisq")

## Analysis of Deviance Table
##
## Model 1: num_awards ~ math
## Model 2: num_awards ~ prog + math
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       198     204.02
## 2       196     189.45  2   14.572 0.0006852 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sometimes, we might want to present the regression results as incident rate ratios and their standard errors, together with the confidence interval. To compute the standard error for the incident rate ratios, we will use the Delta method. To this end, we make use the function deltamethod implemented in R package msm.

The output above indicates that the incident rate for prog = Academic is 2.96 times the incident rate for the reference group (prog = General). Likewise, the incident rate for prog = Vocational is 1.45 times the incident rate for the reference group holding the other variables at constant. The percent change in the incident rate of num_awards is by 7% for every unit increase in math. For additional information on the various metrics in which the results can be presented, and the interpretation of such, please see Regression Models for Categorical Dependent Variables Using Stata, Second Edition by J. Scott Long and Jeremy Freese (2006).

Sometimes, we might want to look at the expected marginal means. For example, what are the expected counts for each program type holding math score at its overall mean? To answer this question, we can make use of the predict function. First off, we will make a small data set to apply the predict function to it.

```
(s1 <- data.frame(math = mean(p$math),
  prog = factor(1:3, levels = 1:3, labels = levels(p$prog))))

##     math       prog
## 1 52.645    General
## 2 52.645   Academic
## 3 52.645 Vocational

predict(m1, s1, type="response", se.fit=TRUE)

## $fit
##         1         2         3
## 0.2114109 0.6249446 0.3060086
##
## $se.fit
##          1          2          3
## 0.07050108 0.08628117 0.08833706
##
## $residual.scale
## [1] 1
```

In the output above, we see that the predicted number of events for level 1 of prog is about .21, holding math at its mean. The predicted number of events for level 2 of prog is higher at .62, and the predicted number of events for level 3 of prog is about .31. The ratios of these predicted counts ((frac.625.211 = 2.96), (frac.306.211 = 1.45)) match what we saw looking at the IRR.

We can also graph the predicted number of events with the commands below. The graph indicates that the most awards are predicted for those in the academic program (prog = 2), especially if the student has a high math score. The lowest number of predicted awards is for those students in the general program (prog = 1). The graph

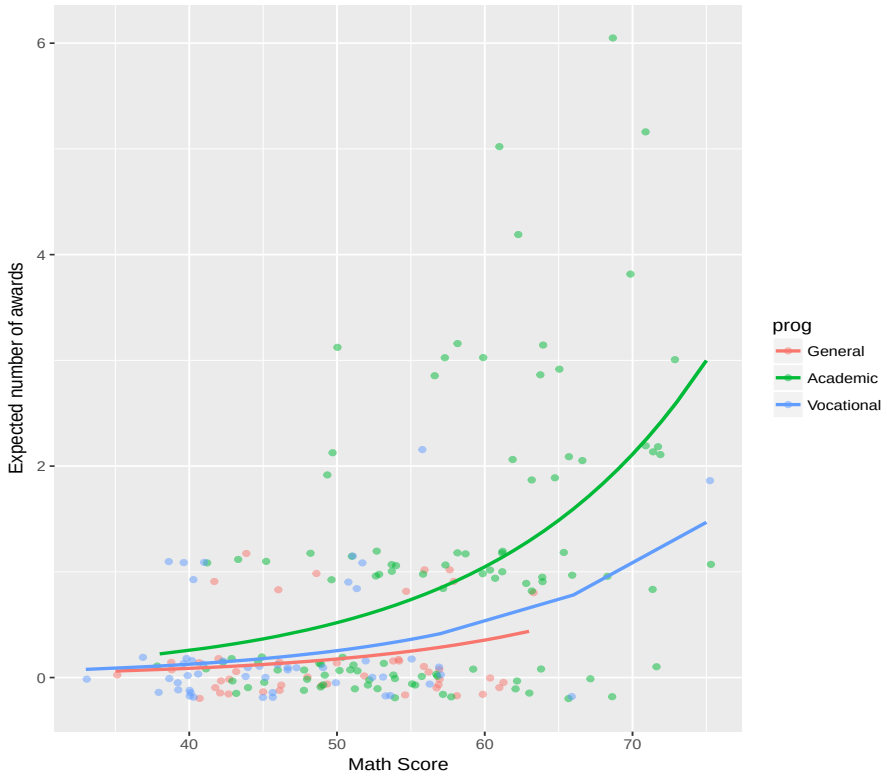


Figure 3.2: *An overview of the data and model.*

overlays the lines of expected values onto the actual points, although a small amount of random noise was added vertically to lessen overplotting.

```
## calculate and store predicted values
p$phat <- predict(m1, type="response")

## order by program and then by math
p <- p[with(p, order(prog, math)), ]

## create the plot
ggplot(p, aes(x = math, y = phat, colour = prog)) +
  geom_point(aes(y = num_awards), alpha=.5,
  position=position_jitter(h=.2)) +  geom_line(size = 1) +
  labs(x = "Math Score", y = "Expected number of awards")
```

Things to consider

- When there seems to be an issue of dispersion, we should first check if our model is appropriately specified, such as omitted variables and functional forms. For example, if we omitted the predictor variable prog in the example above, our model would seem to have a problem with over-dispersion. In other words, a misspecified model could present a symptom like an over-dispersion problem.

- Assuming that the model is correctly specified, the assumption that the conditional variance is equal to the conditional mean should be checked. There are several tests including the likelihood ratio test of over-dispersion parameter alpha by running the same model using negative binomial distribution. R package pscl (Political Science Computational Laboratory, Stanford University) provides many functions for binomial and count data including odTest for testing over-dispersion.
- One common cause of over-dispersion is excess zeros, which in turn are generated by an additional data generating process. In this situation, zero-inflated model should be considered.
- If the data generating process does not allow for any 0s (such as the number of days spent in the hospital), then a zero-truncated model may be more appropriate.
- Count data often have an exposure variable, which indicates the number of times the event could have happened. This variable should be incorporated into a Poisson model with the use of the offset option.
- The outcome variable in a Poisson regression cannot have negative numbers, and the exposure cannot have 0s.
- Many different measures of pseudo-R-squared exist. They all attempt to provide information similar to that provided by R-squared in OLS regression, even though none of them can be interpreted exactly as R-squared in OLS regression is interpreted. For a discussion of various pseudo-R-squares, see Long and Freese (2006) or our FAQ page What are pseudo R-squareds?.
- Poisson regression is estimated via maximum likelihood estimation. It usually requires a large sample size.

References

- Cameron, A. C. and Trivedi, P. K. 2009. Microeconometrics Using Stata. College Station, TX: Stata Press.
- Cameron, A. C. and Trivedi, P. K. 1998. Regression Analysis of Count Data. New York: Cambridge Press.
- Cameron, A. C. Advances in Count Data Regression Talk for the Applied Statistics Workshop, March 28, 2009. http://cameron.econ.ucdavis.edu/racd/count.html .
- Dupont, W. D. 2002. Statistical Modeling for Biomedical Researchers: A Simple Introduction to the Analysis of Complex Data. New York: Cambridge Press.
- Long, J. S. 1997. Regression Models for Categorical and Limited Dependent Variables. Thousand Oaks, CA: Sage Publications.
- Long, J. S. and Freese, J. 2006. Regression Models for Categorical Dependent Variables Using Stata, Second Edition. College Station, TX: Stata Press.

Assignments

4.1 Returns

Assignment

Question

Assume that returns on the stock exchange are normally distributed with an average return of 5% and a volatility of 20%. Investigate the distribution after 10 years. Is that still a normal distribution?
What if you would assume that the log-returns were normally distributed?

Note that return $R := \frac{V_f}{V_i} - 1$ and that a log-return is defined as $r := \log\left(\frac{V_f}{V_i} - 1\right)$.

4.2 Crimes in the USA

Assignment

Question

Based on the data-set UScrime (in the package MASS), what would you recommend to reduce crime? Write also about the limitations of your findings.

4.3 What influences the GDP-growth?

Assignment

Question

Find out what governments should focus on in order to improve the GDP per capita, using public data.

Suggestions:

1. use `https://data.oecd.org` to download data,
2. think of a simple method to make it work (correlate the result to the chosen indicators N-years in the past (better still, use averages, etc.)

4.4 What makes a good model?

Assignment

Question

Re-read our section on model performance (see Chapter 2.5 on page 85) and prepare a presentation on what makes a good model.

4.5 BYOD: bring your own data

Assignment

Question

Consider your company or business, identify a problem worth solving, get the data and present a solution.

This is possible the most interesting assignment, however be careful. This look deceivingly easy, so you might want to read the section on MCDA before starting this.

PART II

Appendices

Other Resources

- https://www.tutorialspoint.com/r/r_lists.htm
- http://www.r-tutor.com/r-introduction
- https://www.math.columbia.edu/ ik/tutor.pdf
- http://www.cyclismo.org/tutorial/R/time.html

PART II .:. CHAPTER 6 .:.

Levels of Measurement

Levels of Measurement

Introduction

It is customary to refer to the theory of scales as having been developed by Stevens (1946). In that paper he argues that all measurement is done by assuming a certain scale type. He distinguished four different types of scale: nominal, ordinal, interval, and ratio scales.

6.1 Nominal Scale

Nominal Scale

The nominal scale is the simplest form of classification. It simply contains labels that do not even assume an order. Examples include asset classes, first names, countries, days of the month, weekdays, etc. It is not possible to use statistics such as average or median, and the only thing that can be measured is which label occurs the most (modus of mode).

Scale Type	Nominal
Characterization	labels (e.g. asset classes, stock exchanges)
Permissible Statistics	mode (not median or average), chi-square
Permissible Scale Transformation	equality
Structure	unordered set

Table 6.1: *Characterization of the Nominal Scale of Measurement.*

Note that it is possible to use numbers as labels, but that this is very misleading. When using an nominal scale, none of the traditional metrics (such as averages) can be used.

6.2 Ordinal Scale

Ordinal Scale

This scale type assumes a certain order. An example is a set of labels such as very safe, moderate, risky, very risky. Bond rating such as AAA, BB+, etc. also are ordinal scales: they indicate a certain order, but there is no way to determine if the distance between, say, AAA and AA- is similar to the distance between BBB and BB-. It may make sense to talk about a median, but it does not make any sense to calculate an average (as is sometimes done in the industry and even in regulations)

Scale Type	**Ordinal Scale**
Characterization	ranked labels (e.g. ratings for bonds from rating agencies)
Permissible Statistics	median, percentile
Permissible Scale Transformation	order
Structure	(strictly) ordered set

Table 6.2: *Characterization of the Ordinal Scale of Measurement.*

Ordinal labels can be replaced by others if the strict order is conserved (by a strict increasing or decreasing function). For example AAA, AA-, and BBB+ can be replaced by 1, 2 and, 3 or even by -501, -500, and 500,000. The information content is the same, the average will have no meaningful interpretation.

6.3 Interval Scale

Interval Scale

This scale can be used for many quantifiable variables: temperature (in degrees Celsius). In this case, the difference between 1 and 2 degrees is the same as the difference between 100 and 101 degrees, and the average has a meaningful interpretation. Note that the zero point has only an arbitrary meaning, just like using a number for an ordinal scale: it can be used as a name, but it is only a name.

Scale Type	Interval Scale
Characterization	difference between labels is meaningful (e.g. the Celsius scale for temperature)
Permissible Statistics	mean, standard deviation, correlation, regression, analysis of variance
Permissible Scale Transformation	affine
Structure	affine line

Table 6.3: *Characterization of the Interval Scale of Measurement.*

Rescaling is possible and remains meaningful. For example, a conversion from Celsius to Fahrenheit is possible via the following formula, $T_f = \frac{9}{5}T_c + 32$, with T_c the temperature in Celsius and T_f the temperature in Fahrenheit.

An affine transformation is a linear transformation of the form $\boldsymbol{y} = \boldsymbol{A}.\boldsymbol{x} + \boldsymbol{b}$. In Euclidean space an affine transformation will preserve collinearity (so that lines that lie on a line remain on a line) and ratios of distances along a line (for distinct collinear points p_1, p_2, p_3, the ratio $||p_2 - p_1||/||p_3 - p_2||$ is preserved).

In general, an affine transformation is composed of linear transformations (rotation, scaling and/or shear) and a translation (or "shift"). An affine transformation is an internal operation and several linear transformations can be combined into one transformation.

6.4 Ratio Scale

Ratio Scale

Using the Kelvin scale for temperature allows us to use a ratio scale: here not only the distances between the degrees but also the zero point is meaningful. Among the many examples are profit, loss, value, price, etc. Also a coherent risk measure is a ratio scale, because of the property translational invariance implies the existence of a true zero point.

Scale Type	**Ratio Scale**
Characterization	a true zero point exists (e.g. VAR, VaR, ES)
Permissible Statistics	geometric mean, harmonic mean, coefficient of variation, logarithms, etc.
Permissible Scale Transformation	multiplication
Structure	field

Table 6.4: *Characterization of the Ratio Scale of Measurement.*

References

Bibliography

De Brouwer, P. J. S. (2012). *Maslowian Portfolio Theory, a Coherent Approach to Strategic Asset Allocation*. Brussels: VUBPress.

Stevens, S. S. (1946). On the theory of scales of measurement. *Science 103*(2684), 677–680.

Index

Nomenclature

$\bar{x}$	mean, page 65
x_k	an observation of the stochastic variable X, page 65
f(.)	probability density function, page 65
P(.)	probability, page 65
pdf	probability density function, page 65