Introduction to R

Lino AA Notarantonio (lino@tec.mx)

05/08/2019

Introduction to R(Studio)

R(Studio)

- R is the actual programming language
- RStudio is an IDE (Integrated Development Environment) for R.
- R is **case sensitive**; *e. g.*, `Mean` $\neq$ `mean`
- R(Studio) may not work very well when files (or directory containing working files) have accented characters. If the locale language of your filesystem is not in English, then some errors may occur in those cases.

Basic calculations

- R can be used as a calculator
- Mathematical constants:
  - pi $= 3.142$
  - exp(1) $= 2.718$
- Logarithms:
  - log(e) $= 1$
  - log10(100) $= 2$
  - log(16, base $=$ 4) $= 2$
- Getting help: in the console, put a question mark before the function name; RStudio will display the documentation: `?cos`
- Se set the display of decimal digits to 3, for a better output:

```
options(digits = 3)
```

Installing packages

One of the strengths of R is the increasing number of available packages (more than 14,000 of them): CRAN Packages

To install and use a package we have

- firstly to **install** it;
- then, to **load** it in the current session.

To install a package (*e.g.*, `tseries`), we can use the `install.packages("tseries")` function.

To load it in the current session, use `library(tseries)`

To install a package, we can also use the Tools option in the RStudio window and follow Install Packages...

Data Types and Data Structure

Data types

The data types used by R are

- numeric (double precision): 2.718, 1.4, ...
- integer: 1, -13, ...
- complex: $2 - 3i$, ...
- logical: TRUE, FALSE. Also NA is considered logical
- character: `"one plus two"`, `"Hello world!"`

Data Structure

- Vector: basic data structure in R. Its components have the same data type.
- Matrix: think of linear algebra.
  - A matrix as a collection of vectors.
- Dataframe: similar to a matrix, except that it is not necessarily homogeneous.
  - A collection of vectors (of possible different types) with the same length.
- List: generic data structure containing other **objects** (vectors, other lists), **not necessarily of the same length**.

Assignment operator

In R, we assign a value, `value`, to an object `x` by means of `<-`:

```
x <- value
```

It is also possible to use `=`, but the equal sign has lower priority than `<-`.

Check the discussion at StackExchange.

Vectors

The easiest way to create a vector in R is to use the `c()` function:

```
v <- c(1, 3, 5, 7, 9)
v
```

```
## [1] 1 3 5 7 9
```

Vectors: coercion

If we define a vector with components of different data types, the result will be coerced to the same data type

```
w <- c(1.56, "Hello World", 4, TRUE)
```

```
typeof(w)
```

```
## [1] "character"
```

Notice that

```
u <-  c(1.56, 4, TRUE)
typeof(u)
```

```
## [1] "double"
```

Vectors by sequences

- Create a vector with the first 30 integer numbers:

```
(x <- 1:30)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
## [24] 24 25 26 27 28 29 30
```

- In reverse order:

```
(x <- 30:1)
```

```
##  [1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
## [24]  7  6  5  4  3  2  1
```

Vectors by sequences

- Create a sequence of the first odd numbers, up to 11:

```
(y <-  seq(1, 11, 2))
```

```
## [1]  1  3  5  7  9 11
```

- Repeat the character “Hello” 5 times:

```
(z <- rep("Hello", 5))
```

```
## [1] "Hello" "Hello" "Hello" "Hello" "Hello"
```

- Repeat the vector y, defined above, 2 times:

```
(rep(y,2))
```

```
##  [1]  1  3  5  7  9 11  1  3  5  7  9 11
```

Length of a vector

The length of a vector (number of its components) is obtained with the function `length()`

```
x <- 1:30
length(x)
```

```
## [1] 30
```

```
y <-  seq(1, 11, 2)
length(y)
```

```
## [1] 6
```

```
z <- rep("Hello", 5)
length(z)
```

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

Subsetting

Let `y <- seq(1,11,2)`: 1, 3, 5, 7, 9, 11.

- Select the third component: `y[3]` = 5
- Exclude the fourth component: `y[-4]`: 1, 3, 5, 9, 11.
- Select the first four components: `y[c(1:4)]`: 1, 3, 5, 7
- Select the first, fifth and last element:

```
y[c(1, 5, length(y))]
```

```
## [1]  1  9 11
```

or

```
y[c(1, length(y) - 1, length(y))]
```

```
## [1]  1  9 11
```

Subsetting

- Remove the first four components: `y[-c(1:4)]`: 9, 11.
- Select the second, fifth and sixth components: `y[c(2, 5, 6)]`: 3, 9, 11.
- Select components by using a vector of logic type:

```
s <- c(TRUE, TRUE, FALSE, FALSE, TRUE, FALSE)
y[c(s)]
```

```
## [1] 1 3 9
```

Vectorization

Consider the vector

```
x <- 1:100
```

Then,

- `x + 4` = 5, 6, 7, 8, 9, 10 ...
- `x^2` = 1, 4, 9, 16, 25, 36 ...
- `2*x+3` = 5, 7, 9, 11, 13, 15 ...
- `sqrt(x)` = 1, 1.414, 1.732, 2, 2.236, 2.449 ...
- `log(x)` = 0, 0.693, 1.099, 1.386, 1.609, 1.792 ...

Vectorization

We can add up two vectors even when they have different lengths, provided that the length of one is an integer multiple of the other:

```
x <- seq(1,11,2)
y <- 1:12
x + y
```

```
##  [1]  2  5  8 11 14 17  8 11 14 17 20 23
```

Check for identity

```
all(x + y == rep(x,2) + y)
```

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

```
identical(x + y, rep(x,2) + y)
```

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

Logical operators

Each of the following operators returns either `TRUE` or `FALSE`.

- `==` (equality)
- `!=` (not equal to)
- `>` (greater than)
- `<` (less than)
- `>=` (greater than or equal to)
- `<=` (less than or equal to)
- `!x` (not `x`)
- `x | y` (`x` OR `y`)
- `x & y` (`x` AND `y`)

Logical operators

Examples

Let `x <- c(1,2,3,4,5,6)`

- `x > 4`: FALSE, FALSE, FALSE, FALSE, TRUE, TRUE
- `x == 4`: FALSE, FALSE, FALSE, TRUE, FALSE, FALSE
- `x != 3`: TRUE, TRUE, FALSE, TRUE, TRUE, TRUE
- `x == 4 | x != 3`: TRUE, TRUE, FALSE, TRUE, TRUE, TRUE
- `x == 4 & x != 3`: FALSE, FALSE, FALSE, TRUE, FALSE, FALSE
- `as.numeric(x == 4 | x != 3)`: 1, 1, 0, 1, 1, 1
- `sum(as.numeric(x == 4 | x != 3))`: 5

Logical operators

Logical operators and subsetting

Subsetting will extract the values of the components that satisfy the given logical condition.

Let `x <- c(1,2,3,4,5,6)`

- `x[x > 3]`: 4, 5, 6
- `x[x != 3]`: 1, 2, 4, 5, 6
- `x[x <= 0]`: numeric(0) (empty set)
- `x[x <= 0 | x > 3]`: 4, 5, 6
- `x[x <= 0 & x > 3]`: numeric(0) (empty set)
- `sum(x[x <= 0 & x>3])`: 0

Logical operators

Selection

```
y <- c(1, -2, 4, 6, 9, 2, 1)
```

- `which(y <= 4)`: 1, 2, 3, 6, 7
  It selects which entries satisfy the condition. Indexing of vectors starts from 1.
- `y[which(y <= 4)]`: 1, -2, 4, 2, 1
  It returns the values of the entries satisfying the condition.
- `which(y == max(y))`: 5 $<=$ **entry of the vector**
- `y[which(y == max(y))]`: 9 $<=$ **maximum value**

Matrices

A matrix A is specified by the number of its rows and columns, $m \times n$.

The **order** of rows and columns is **important**.

Matrix can be created by means of the `matrix()` function:

```
x <- 1:8
A <- matrix(x, nrow = 4, ncol = 2)
B <- matrix(x, nrow = 2, ncol = 4)
dim(A) # rows = 4; columns = 2
```

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

```
dim(B) # rows = 2; columns = 4
```

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

Matrices


```
A
```

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

```
B
```

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

Matrices: Subsetting

- Select the entry of the matrix A in the first row, second column:

```
A[1,2]
```

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

- Select the third column of the matrix B

```
B[,3]
```

```
## [1] 5 6
```

- Select the third row of the matrix A

```
A[3,]
```

```
## [1] 3 7
```

Matrices: Subsetting

- Select the first and fouth column of the matrix B

```
B[, c(1,4)]
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
```

Binding vectors: by columns

```
u <- 1:4
v = rev(u)
w <- rep(1,4)
C <- cbind(u,v,w)
rownames(C) <- c("1st", "2nd", "3rd", "4th")
C
```

```
##     u v w
## 1st 1 4 1
## 2nd 2 3 1
## 3rd 3 2 1
## 4th 4 1 1
```

Binding vectors: by rows

```
u <- 1:4
v = rev(u)
w <- rep(1,4)
D <- rbind(u,v,w)
colnames(D) <- c("1st", "2nd", "3rd", "4th")
D
```

```
##   1st 2nd 3rd 4th
## u   1   2   3   4
## v   4   3   2   1
## w   1   1   1   1
```

Matrix operations

- The sum of two compatible matrices is `A + B`
- The subtractino of two compatible matrices is `A - B`
- The product of two compatible matrices is `A %*% B`
- The transpose of a matrix A is `t(A)`
- The inverse of a square matrix, A, if it exists, is `solve(A)`

Inverse matrix

```r
x <- c(1,2,3,5, 4, 1, 2,2,1)
A <- matrix(x, nrow = 3, ncol = 3)
A1 <- solve(A)
A1 %*% A
```

```
##           [,1] [,2]     [,3]
## [1,] 1.00e+00    0 2.22e-16
## [2,] 8.88e-16    1 1.11e-15
## [3,] 0.00e+00    0 1.00e+00
```

The result is the identity matrix, up to round-off errors.

More matrices

- We can use the function `all.equal()` to compare the results of `solve(A)` and `A1 %*% A` with the identity matrix:

```
all.equal(A1 %*% A, diag(3))
```

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

- The function `diag()` can also be used to extract the diagonal elements of a matrix

```
diag(A)
```

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

More matrices


```r
diag((1:5)^(.5))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1 0.00 0.00    0 0.00
## [2,]    0 1.41 0.00    0 0.00
## [3,]    0 0.00 1.73    0 0.00
## [4,]    0 0.00 0.00    2 0.00
## [5,]    0 0.00 0.00    0 2.24
```

More matrices


```
diag(log(1):log(5))
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    1
```

```
diag(log(1:5))
```

```
##      [,1]  [,2] [,3] [,4] [,5]
## [1,]    0 0.000  0.0 0.00 0.00
## [2,]    0 0.693  0.0 0.00 0.00
## [3,]    0 0.000  1.1 0.00 0.00
## [4,]    0 0.000  0.0 1.39 0.00
## [5,]    0 0.000  0.0 0.00 1.61
```

More matrices

R has some built-in functions to deal with matrices. Consider

```
A
```

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

- `rowSums(A)` = 8, 8, 5
- `colSums(A)` = 6, 10, 5
- `rowMeans(A)` = 2.667, 2.667, 1.667
- `colMeans(A)` = 2, 3.333, 1.667

More matrices

The operator %*%

The operator %*% works differently on vectors and matrices.

- On vectors it computes the **dot product**
- On matrices, the matrix multiplication (matrix multiplication is a form of ordered, vectorized dot product)

```
a <- c(1,2,3)
b <- c(4,5,6)
a %*% b
```

```
##      [,1]
## [1,]   32
```

More matrices

Vector can be coerced to work as matrices; in this case, columns matrices:

```
as.matrix(a)
```

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

```
t(as.matrix(b)) %*% as.matrix(a) # dot product
```

```
##      [,1]
## [1,]   32
```

More matrices

```
as.matrix(a) %*% t(as.matrix(b))
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    6
## [2,]    8   10   12
## [3,]   12   15   18
```

Dataframes

- Statistical analysis is done using datasets. A dataset contains a certain number of variables and observations.
- It is a good practice to have each variable set as a column vector and each observation as a row vector.
- A dataframe, in R, is the data structure of an observed dataset.
- A dataframe can be thought of as a matrix in which different columns may have different data types.

Working with a Dataframe

Preliminary analysis

When loading a dataset for analysis, there are some aspects to consider.

1. Understand the data: what the dataset is about; what are its variables; how many obervation the dataset contains.
2. Determine whether there are missing observations; some functions will not work properly otherwise.
3. Visualize some of its variables.

Import the dataset with RStudio

Information about the dataset

- The dataset is taken from the UCI, Machine Learning Repository website.
- The dataset can be found here.
  - Clicking the link will download the file; we will load the dataset more conveniently later on, with a different set of tools.
- The dataset will be called `cleve` hereinafter in the presentation.
- The dataset format is `csv` (comma separated variables).

Import the dataset with RStudio

1. Set the working directory to where you have downloaded the folder of the workshop.
2. Open an R script and save it with a meaningful name.
3. Write in the script
   `cleve <- read.csv("cleve.csv", header = FALSE)`
   and hit **CTRL + Enter** (Windows); **CMD + Enter** (Mac OSX) to execute.
4. Try `head(cleve)` in the script and execute.

Working with the dataset

- Set the names of the variables:

```
cleve.names <- c("age", "sex", "cp", "trestbps",
                 "chol","fbs", "restcg", "thalac",
                 "exang","oldpeak","slope", "ca",
                 "thal", "diagnostic")
names(cleve) <- cleve.names
```

- The documentation of the dataset is found here
- To work with only one variable from the dataset, *e.g.*, *age*, we can **extract** it by means of $.

```
age <- cleve$age
```

Preliminary analysis

Properties of the dataset

The dataset is a dataframe

```
class(cleve)
```

```
## [1] "data.frame"
```

with

```
dim(cleve)
```

```
## [1] 303  14
```

nrow(cleve) = 303 observations and ncol(cleve) = 14 variables.

Properties of the dataset

The names of the variables can also be obtained by `colnames(cleve)`.

The first six observations can be displayed by

```
head(cleve)
```

(Output omitted because it does not fit the slide.)

Slicing

Display the first 3 observations of the first, third and seventh through ninth variables:

```
cleve[1:3,c(1,3,7:9)]
```

```
##   age cp restcg thalac exang
## 1  63  1      2    150     0
## 2  67  4      2    108     1
## 3  67  4      2    129     1
```

or with (omitted for space)

```
cleve[1:2, c("age", "cp", "restcg",
             "thalac", "exang")]
```

Slicing

Examples

1. Compute the number of individuals whose age is greater than, or equal to, 50 years:

```
length(cleve$age[cleve$age >= 50])
```

```
## [1] 216
```

2. Compute the number of individuals without a diagnosis of heart disease:

```
length(cleve$sex[cleve$diagnostic == 0])
```

```
## [1] 164
```

Remark The variable `diagnostic` is strictly positive if there is indication of a heart condition.

Slicing

Examples

3. Compute the number of individuals with a diagnosis of heart disease **and** with fasting blood sugar $>$ 120 mg/dl

```
length(cleve$sex[cleve$diagnostic > 0 &
                   cleve$fbs == 1])
```

```
## [1] 22
```

4. Compute the number of individuals with a diagnosis of heart disease **or** with fasting blood sugar $>$ 120 mg/dl

```
length(cleve$sex[cleve$diagnostic > 0 |
                   cleve$fbs == 1])
```

```
## [1] 162
```

Descriptive Statistics

Consider, *e.g.*, the variable *age*. Compute:

1. the mean (average)

```
mean(cleve$age)
```

```
## [1] 54.4
```

2. the median

```
median(cleve$age)
```

```
## [1] 56
```

3. the interquartile range

```
IQR(cleve$age) # Q3 - Q1
```

```
## [1] 13
```

Descriptive Statistics

4. the summary of the principal statistics

```
summary(cleve$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    29.0    48.0    56.0    54.4    61.0    77.0
```

Remark The function `summary()` does not return neither the variance nor the standard deviation.

5. Quantile distribution

```
quantile(cleve$age, c(.1, .25, .40, .60, .80))
```

```
## 10% 25% 40% 60% 80%
##  42  48  53  58  62
```

Descriptive Statistics

It is possible to apply the `summary()` function to a full dataset, or to several variables of it

```
summary(cleve[,c(1,5,8)])
```

```
##       age             chol          thalac
##  Min.   :29.0   Min.   :126   Min.   : 71
##  1st Qu.:48.0   1st Qu.:211   1st Qu.:134
##  Median :56.0   Median :241   Median :153
##  Mean   :54.4   Mean   :247   Mean   :150
##  3rd Qu.:61.0   3rd Qu.:275   3rd Qu.:166
##  Max.   :77.0   Max.   :564   Max.   :202
```

(Only three variables are selected for space.)

Descriptive Statistics

Variance and standard deviation

- The variance (resp., standard deviation) is computed by `var()` (resp., `sd()`).
- If the variable has `NA` (missing values), then `var()`, `sd()` return `NA`:

```
var(cleve$thal)
```

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

```
sd(cleve$thal)
```

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

Descriptive Statistics

Variance

We can apply the function `var()` to a single variable or several of them; in the latter case, we obtain the variance-covariance matrix of the selected variables:

```
var(cleve[, c(1, 3, 5, 6)])
```

```
##          age      cp     chol     fbs
## age   81.697  0.9037   97.787  0.3816
## cp     0.904  0.9218    3.595 -0.0137
## chol  97.787  3.5951 2680.849  0.1815
## fbs    0.382 -0.0137    0.181  0.1269
```

Descriptive Statistics

Standard deviation

The standard deviation `sd()`, on the other hand, can only be applied to a single variable:

```
sd(cleve$age)
```

```
## [1] 9.04
```

```
sd(cleve$chol)
```

```
## [1] 51.8
```

Descriptive Statistics

Standard deviation of a set of variables

We have to vectorize the function `sd()` by means of `apply()`:

```
apply(cleve[,11:13], 2, sd)
```

- The value 2 in the second parameter of the function `apply()` computes the standard deviation, `sd`, of each variable (column).
- The value 1 would compute the standard deviation of each row (observation).

Descriptive Statistics

Standard deviation of a set of variables

```
apply(cleve[,11:13], 2, sd)
```

```
## slope    ca  thal
## 0.616    NA    NA
```

There are missing values in the dataset.

Descriptive Statistics

Count and remove NA's

Let's count them:

```
sum(is.na(cleve))
```

```
## [1] 6
```

We can eliminate them (less than 2% of the observed values)

```
cleve <- na.omit(cleve)
sum(is.na(cleve))
```

```
## [1] 0
```

Descriptive Statistics

Standard deviation of a set of variables

```
apply(cleve[,11:13], 2, sd)
```

```
## slope    ca  thal
## 0.618 0.939 1.939
```

The variable `slope` has a slightly larger standard deviation now.

Contingency Tables

Write a contingency table of gender and diagnosis of heart disease:

```
table(cleve$sex, cleve$diagnostic)
```

```
##
##       0  1  2  3  4
##    0 71  9  7  7  2
##    1 89 45 28 28 11
```

Using the `with()` function:

```
with(cleve, table(age, diagnostic))
```

(Same result; output omitted for space.)

Contingency Tables

Alternative to `table`

Using the `xtabs()` function:

```
xtabs(~ cleve$sex + cleve$diagnostic)
```

```
##            cleve$diagnostic
## cleve$sex  0  1  2  3  4
##          0 71  9  7  7  2
##          1 89 45 28 28 11
```

Estimated Frequencies

Redefine the variable `diagnostic` as dummy, with value 1 if some heart problem is observed:

```
cleve$diagnostic[cleve$diagnostic >0] <- 1
```

Table of estimated frequencies:

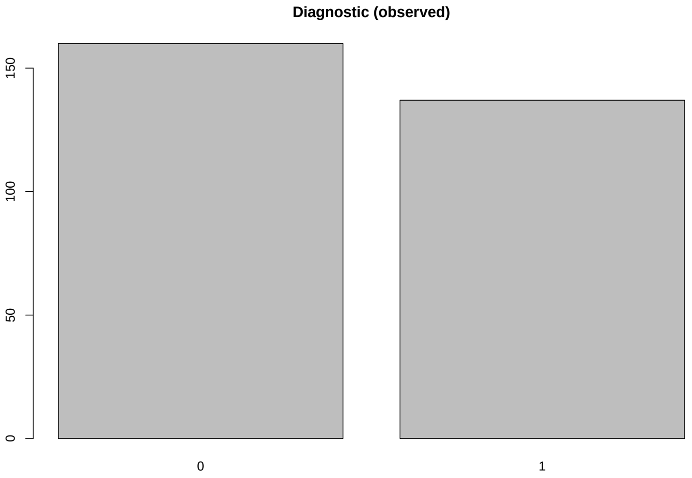
```
table(cleve$sex, cleve$diagnostic)/nrow(cleve)
```

```
##
##          0      1
##   0 0.2391 0.0842
##   1 0.2997 0.3771
```

Visualization: Barplot

```
barplot(table(cleve$diagnostic),
        main = "Diagnostic (observed)")
```

Visualization: Barplot



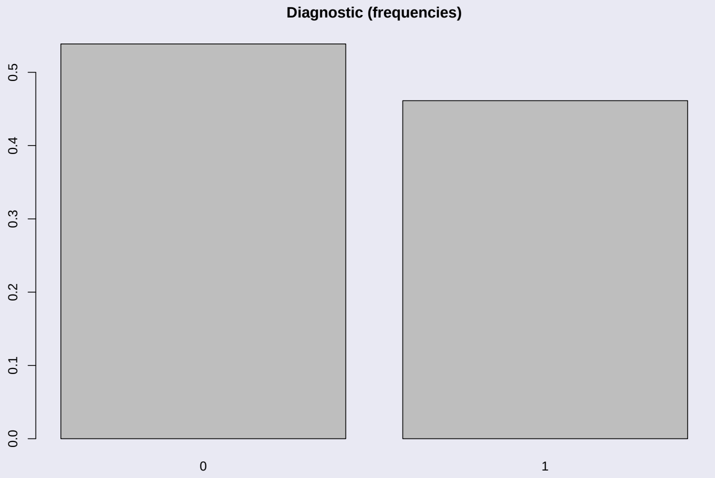
Visualization: Barplot

Barplot with observed frequencies

```
barplot(table(cleve$diagnostic)/nrow(cleve),
        main = "Diagnostic (frequencies)")
```

Visualization: Barplot

Barplot with observed frequencies



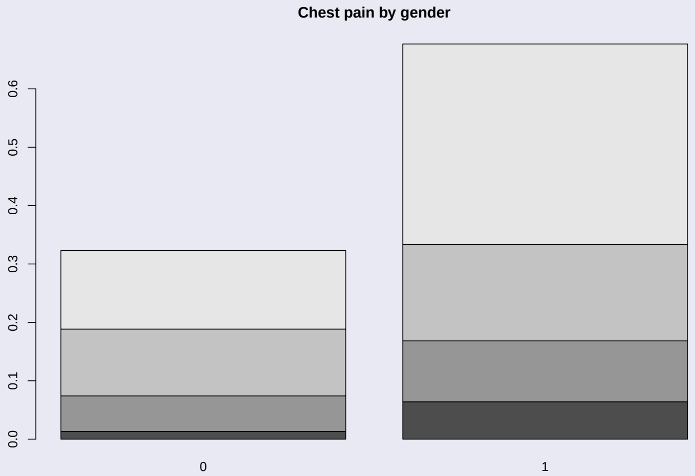
Visualization: Barplot

Barplot with a contingency table

```
barplot(table(cleve$cp,
              cleve$sex)/nrow(cleve),
        main = "Chest pain by gender")
```

Visualization: Barplot

Barplot with a contingency table



Visualization: Barplot

Grouped barplots

```
barplot(table(cleve$cp,
              cleve$sex)/nrow(cleve),
        main = "Diagnostic (frequencies)",
        beside = TRUE)
```

Visualization: Barplot

Grouped barplots

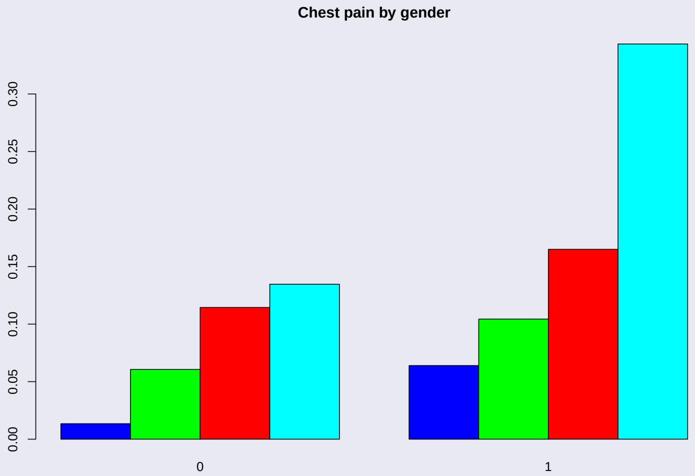
Visualization: Barplot

Grouped barplots with colors

```
barplot(table(cleve$cp,
              cleve$sex)/nrow(cleve),
        main = "Chest pain by gender",
        beside = TRUE,
        col = c("blue", "green", "red", "cyan"))
```

Visualization: Barplot

Grouped barplots with colors



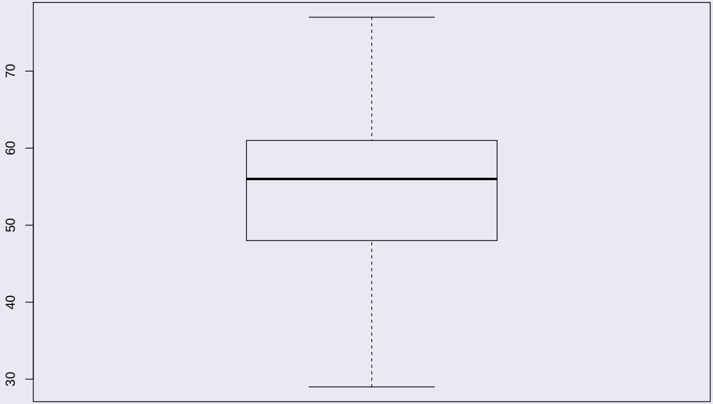
Visualization: Boxplot

Variable “age”

```
boxplot(cleve$age)
```

Visualization: Boxplot

Variable "age"



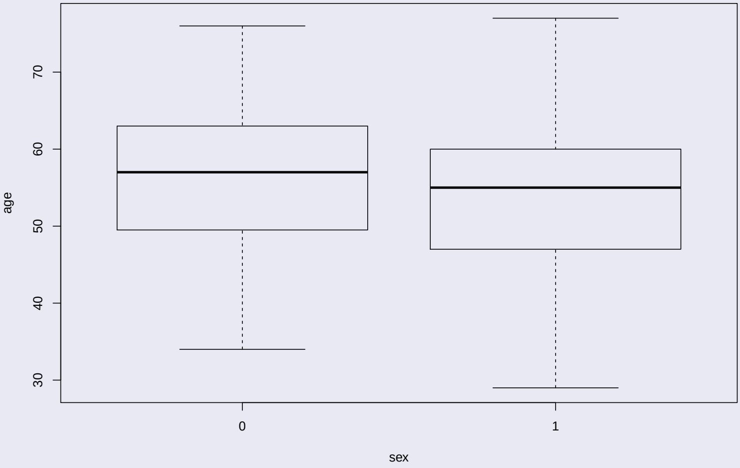
Visualization: Boxplot

Age as a function of gender

```
boxplot(age ~ sex, data = cleve)
```

Visualization: Boxplot

Age as a function of gender



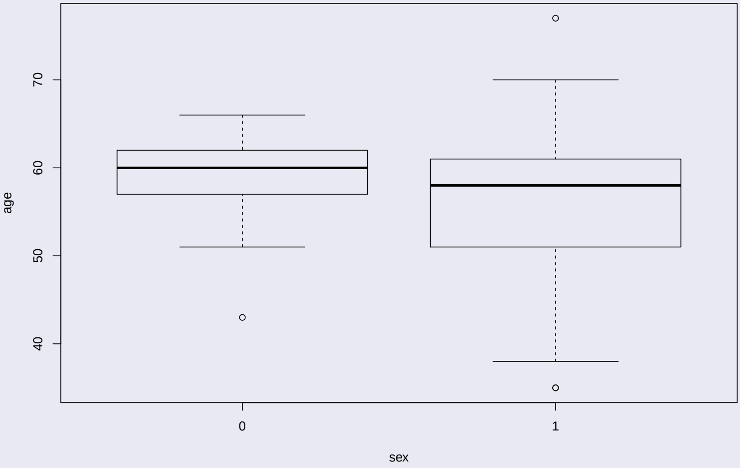
Visualization: Boxplot

Age as a function of gender, with positive diagnosis

```
boxplot(age ~ sex, data = cleve,
        subset = diagnostic == 1)
```

Visualization: Boxplot

Age as a function of gender, with positive diagnosis



Visualization: Boxplot with notch

Age as a function of gender

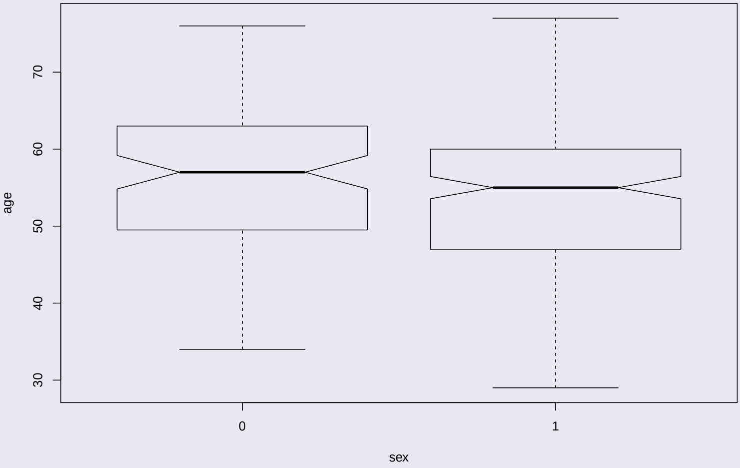
```
boxplot(age ~ sex, data = cleve,
        notch = TRUE)
```

Justification

See, *e.g.*, John M. Chalmers, William S. Cleveland, Beat Kleiner, Paul A. Tukey, “Graphical Methods for Data Analysis”, Wadsworth International Group, Duxbury Press, 1983, pp. 60-63.

Visualization: Boxplot with notch

Age as a function of gender



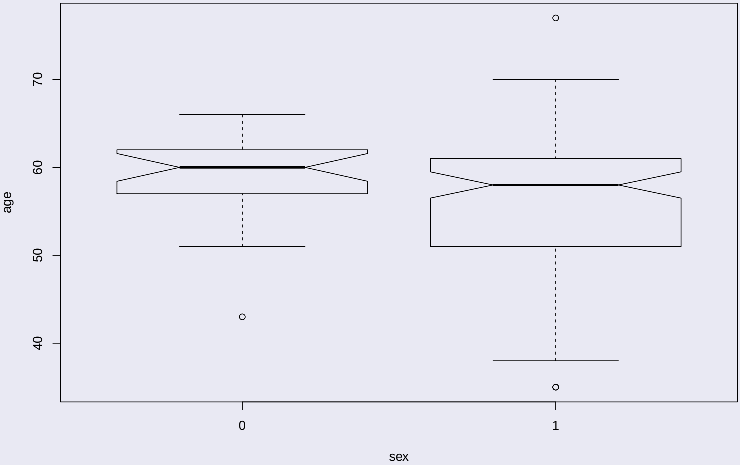
Visualization: Boxplot with notch

Age as a function of gender with positive diagnosis

```
boxplot(age ~ sex, data = cleve,
        notch = TRUE,
        subset = diagnostic == 1)
```

Visualization: Boxplot with notch

Age as a function of gender with positive diagnosis



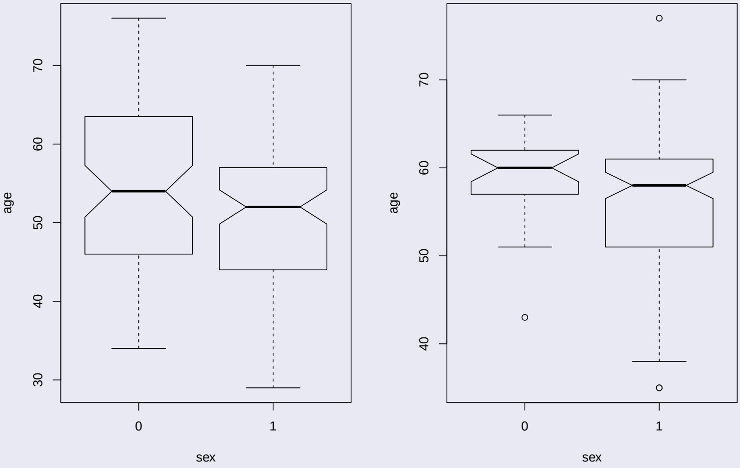
Visualization: Boxplot

Two graphs side by side

```
par(mfrow=c(1,2))
boxplot(age ~ sex, data = cleve,
        notch = TRUE,
        subset = diagnostic == 0)
boxplot(age ~ sex, data = cleve,
        notch = TRUE,
        subset = diagnostic == 1)
par(mfrow = c(1,1))
```

Visualization: Boxplot

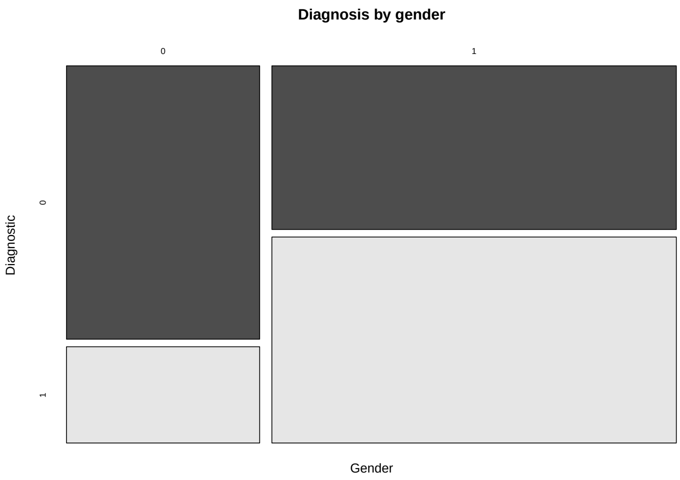
Two graphs side by side



Visualization: Mosaicplot


```r
mosaicplot(cleve$sex ~ cleve$diagnostic,
           shade=F, color=T,
  main = "Diagnosis by gender", xlab="Gender",
  ylab = "Diagnostic")
```

Visualization: Mosaicplot

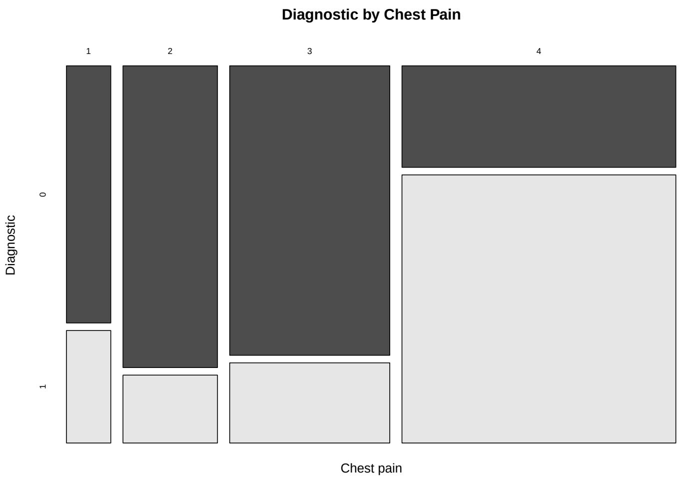


Visualization: Mosaicplot


```
mosaicplot(cleve$cp ~ cleve$diagnostic,
           shade=F, color=T,
  xlab = "Chest pain", ylab = "Diagnostic",
  main = "Diagnostic by Chest Pain")
```

The variable cp has four levels.

Visualization: Mosaicplot

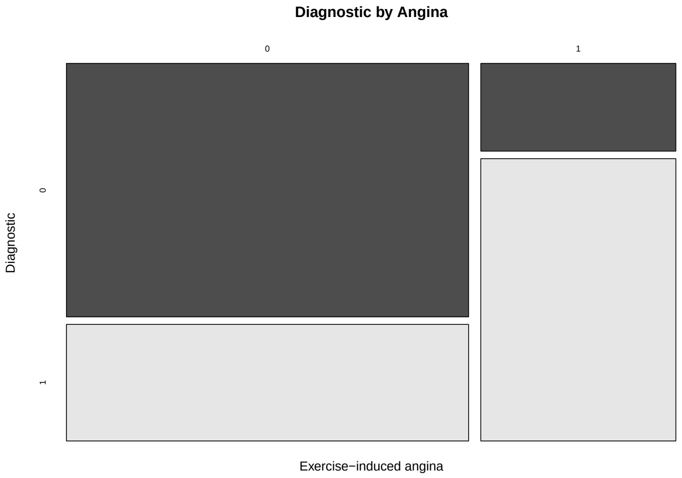


Visualization: Mosaicplot

```
mosaicplot(cleve$exang ~ cleve$diagnostic,
           shade=F, color=T,
  xlab = "Exercise-induced angina",
  ylab = "Diagnostic",
  main = "Diagnostic by Angina")
```

The variable `exang` (exercise induced angina): 1 = yes; 0 = no

Visualization: Mosaicplot



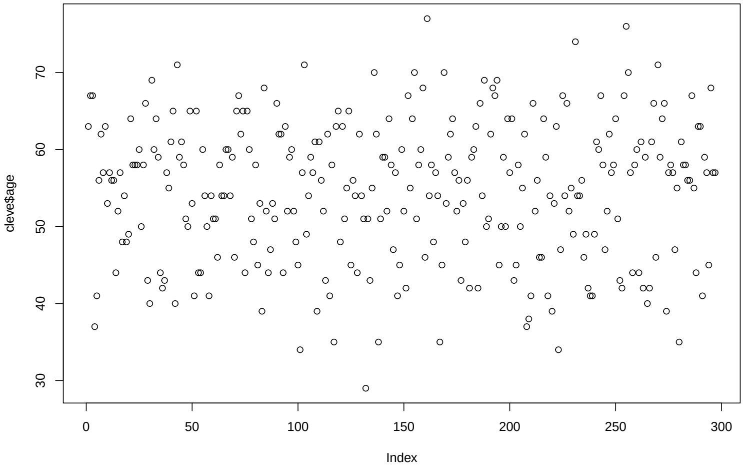
Visualization of continuous variables

Consider the variable *age*.

```
plot(cleve$age)
```

gives the scatterplot of the variable.

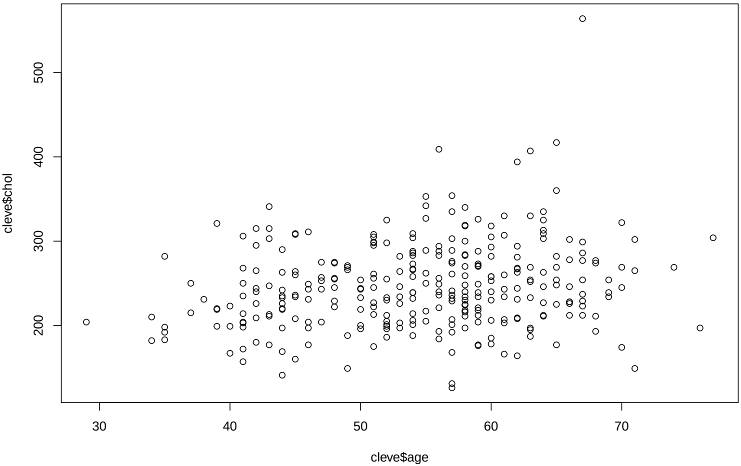
Visualization of continuous variables



Visualization of continuous variables


```
plot(cleve$age, cleve$chol)
```

Visualization of continuous variables

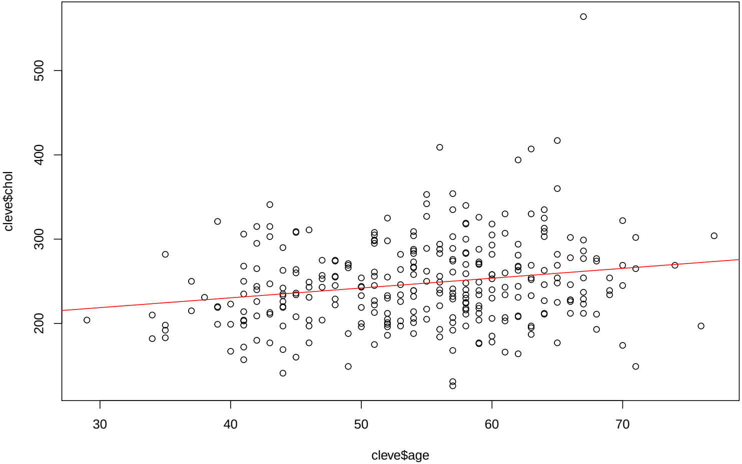


Visualization of continuous variables

With regression line:

```
plot(cleve$age, cleve$chol)
abline(lm(chol ~ age, data = cleve), col = "red")
```

Visualization of continuous variables



More Visualization

See the script `visualization.R`

Classification by K-means clustering

See the script `classification.R`

Statistical Inference

Hypothesis testing

We want to determine whether there is a difference in age between individuals with no heart condition and those with an indication of a heart condition.

```
age.pos <- cleve$age[cleve$diagnostic == 1]
age.neg <- cleve$age[cleve$diagnostic == 0]
```

Formally

$$H_0 : age.pos = age.neg$$
$$H_1 : age.pos \neq age.neg$$

(significance level: $\alpha = .05$)

Hypothesis testing

95% Confidence interval

```
t.test(age.pos, age.neg, mu = 0)
```

```
##
##  Welch Two Sample t-test
##
## data:  age.pos and age.neg
## t = 4, df = 295, p-value = 6e-05
## alternative hypothesis: true difference in means i
## 95 percent confidence interval:
##  2.12 6.11
## sample estimates:
## mean of x mean of y
##      56.8      52.6
```

Hypothesis testing

99% Confidence interval

```
t.test(age.pos, age.neg, mu = 0, conf.level = .99)
```

```
##
##  Welch Two Sample t-test
##
## data:  age.pos and age.neg
## t = 4, df = 295, p-value = 6e-05
## alternative hypothesis: true difference in means i
## 99 percent confidence interval:
##  1.49 6.74
## sample estimates:
## mean of x mean of y
##      56.8      52.6
```

Hypothesis testing

- The function `t.test()` uses by default the Welch *t*.test, with Welch-Satterthwaite correction for degrees of freedom.
- When the variances are equal, we can use

```
t.test(age.pos, age.neg, mu = 0,
       var.equal = TRUE)
```

- The test for equal variances is

```
var.test(age.pos, age.neg, mu = 0, ratio = 1)
```

Hypothesis testing

Power calculation

Having rejected the null hypothesis, we compute the power of the test under the alternative

$$H_1 : age.pos - age.neg = 2.5$$

A power calculation needs four parameters, α, *sd*, *n* (sample size), δ (true difference in mean). It must also specify whether the test is one sided or two sided.

Hypothesis testing

Power calculation

```
power.t.test(n = nrow(cleve), sd =
  sqrt(var(age.pos)+var(age.neg)),
  sig.level = .05, delta = 2.5,
  alternative = "two.sided"
)
```

Hypothesis testing

Power calculation

```
##
##      Two-sample t test power calculation
##
##               n = 297
##           delta = 2.5
##              sd = 12.4
##       sig.level = 0.05
##           power = 0.689
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Hypothesis testing

Power calculation

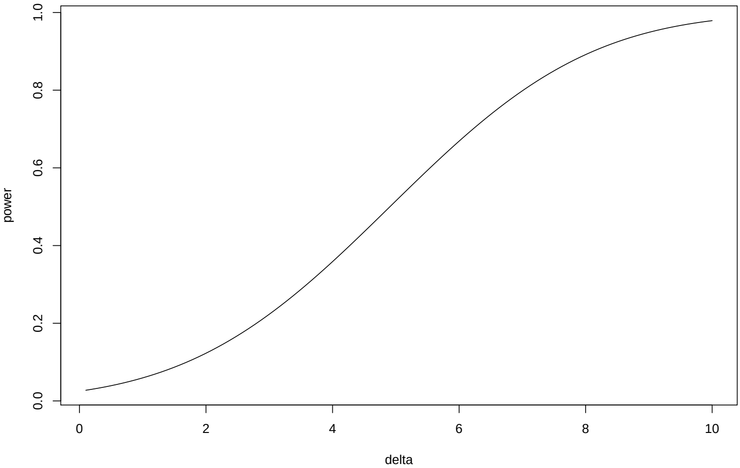
Compute the sample size needed so that the power of the test is .90, when `delta = 2.5`; `sd =` 12.395, with $\alpha = .05$ in a two sided alternative:

```
##
##      Two-sample t test power calculation
##
##               n = 518
##           delta = 2.5
##              sd = 12.4
##       sig.level = 0.05
##           power = 0.9
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Power curve

```
curve(power.t.test(n=50,delta = x,
    sd = sqrt(var(age.pos)+var(age.neg)),
            type="two.sample",
            alternative="two.sided")$power,
                from=.1, to=10, xlab="delta",
            ylab="power")
```

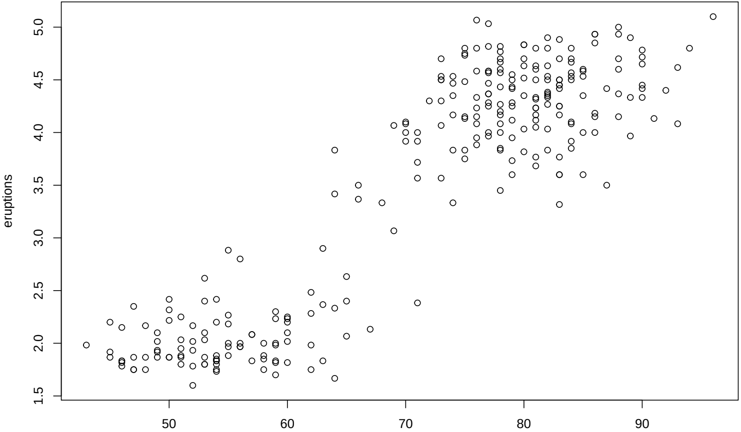
Power curve



Linear regression models

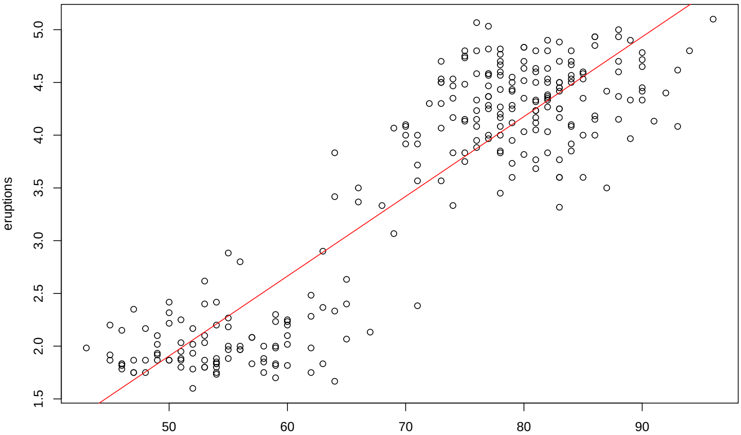
Simple linear regression model


```
attach(faithful)
plot(waiting, eruptions)
```



Simple linear regression model


```r
plot(waiting, eruptions)
abline(lm(eruptions ~ waiting), col = "red")
```



Simple linear regression model

Let us consider the model

$$eruptions = \beta_0 + \beta_1 waiting + u$$

```
slrm <- lm(eruptions ~ waiting, data = faithful)
```

Simple linear regression model

```
summary(slrm)
```

```
##
## Call:
## lm(formula = eruptions ~ waiting, data = faithful)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.2992 -0.3769  0.0351  0.3491  1.1933
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.87402    0.16014   -11.7   <2e-16 *
## waiting      0.07563    0.00222    34.1   <2e-16 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '
##
```

Simple linear regression model

Estimation

Estimate *eruptions*, when *waiting* $= 90$.

```
eruptions.fit.coef <- coefficients(slrm)
c <- c(1,90)
eruptions.fit.coef %*% c
```

```
##        [,1]
## [1,] 4.93
```

Simple linear regression model

Confidence interval

Find a 99% confidence interval for *eruptions*, when *waiting* $= 90$.

```
predict.lm(slrm, data.frame(waiting = 90),
           interval = "confidence",
             level = .99)
```

```
##    fit lwr  upr
## 1 4.93 4.8 5.07
```

Simple linear regression model

Prediction interval

Find a 99% confidence interval for a prediction of *eruptions*, when *waiting* $= 90$.

```
predict.lm(slrm, data.frame(waiting = 90),
           interval = "prediction",
             level = .99)
```

```
##    fit  lwr  upr
## 1 4.93 3.64 6.23
```

Multiple linear regression

Consider the data **hsb2.csv**, which is in the Documentation of the workshop.

The description of the variables can be found here.

Estimate the standardized math score by race:

$$math = \beta_0 + \beta_1 race + u.$$

The variable *race* is **categorical**.

Multiple linear regression

```
hsb2 <- read.csv("hsb2.csv")
m1 <- lm(math ~ factor(race), data = hsb2)
summary.m1 <- summary(m1)
typeof(summary.m1)
```

```
## [1] "list"
```

Multiple linear regression

```r
summary.m1$coefficients
```

```
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       47.417       1.82  26.018 1.72e-65
## factor(race)2      9.856       3.25   3.032 2.76e-03
## factor(race)3     -0.667       2.70  -0.247 8.05e-01
## factor(race)4      6.556       1.97   3.332 1.03e-03
```

Multiple linear regression

Vectorization

```
summary.m1$coefficients[2,]
```

```
##    Estimate Std. Error    t value    Pr(>|t|)
##     9.85606    3.25084    3.03185     0.00276
```

Multiple linear regression

Factor interaction

Estimate the standardized math score for gender, race.

R code

```
m2 <- lm(math ~ female*factor(race), data = hsb2)
summary.m2 <- summary(m2)
```

Multiple linear regression

Factor interaction

Standardized math score for gender, race.

Result

```
summary(m2)$coefficients
```

```
##                          Estimate Std. Error t value P
## (Intercept)                 49.23       2.49  19.760 3
## female                      -3.96       3.68  -1.076 2
## factor(race)2                9.44       5.75   1.640 1
## factor(race)3               -3.95       4.21  -0.937 3
## factor(race)4                4.96       2.72   1.824 6
## female:factor(race)2         2.04       7.11   0.287 7
## female:factor(race)3         6.21       5.59   1.111 2
## female:factor(race)4         3.55       3.97   0.893 3
```

In-sample and out-of-sample validation

We randomly split the data and use one set to estimate the model and the rest to estimate the goodness of fit. To ensure reproducibility, we set the RNG seed to `set.seed(13)`.

```
set.seed(13)
TrainRows <- sample(1:nrow(hsb2),
      .8*nrow(hsb2), replace = FALSE)
Train <- hsb2[TrainRows,]
Valid <- hsb2[-TrainRows, ]
linmodel.Train <-
  lm(math ~ female*factor(race), data = Train)
mathPredic <-
  predict(linmodel.Train, Valid)
actual.predic <-
  data.frame(cbind(actual = Valid$math,
                   predicted = mathPredic))
```

In-sample and out-of-sample validation

Correlation matrix

```
corr.accuracy <- cor(actual.predic)
corr.accuracy
```

```
##            actual predicted
## actual      1.000     0.278
## predicted   0.278     1.000
```

In-sample and out-of-sample validation

Accuracy: Min-Max

The Min-Max method compute the mean of the minimum over the mean of the maximum.
Values very close to 1 (Min-Max $>$ .90) denotes excellent accuracy.

R code

```
min.max.accuracy <-
  mean(apply(actual.predic,
              1, min)/apply(actual.predic, 1, max))
min.max.accuracy
```

```
## [1] 0.863
```

In-sample and out-of-sample validation

Accuracy: Mean Absolute Error (MAE)

If Ai are the actual (observed) values of the response and F_i are the forecast ones, then

$$MAE = \frac{1}{T} \sum_{i=1}^{T} |A_i - F_i| \,;$$

T is the sample size.

R code

```
mae <- mean(abs((actual.predic$predicted -
                    actual.predic$actual)))
mae
```

```
## [1] 7.94
```

In-sample and out-of-sample validation

Accuracy: Mean Absolute Percentage Error (MAPE)

The mean absolute percentage error (MAPE) is defined as

$$MAPE = \frac{1}{T}\sum_{i=1}^{T}\left|\frac{A_i - F_i}{A_i}\right|.$$

- Mape can be interpreted as the average percentage error.
- Sometimes, MAPE can be very large, enve though the forecast is reasonably good. If, *e.g.*, $A_i \approx 10^{-3}$ and $|F_i - A_i| \approx 10^{-1}$, entonces

$$\left|\frac{A_i - F_i}{A_i}\right| \approx 10^2$$

- If the forecast is exact, then $MAPE = 0$.

In-sample and out-of-sample validation

MAPE: R code

```
mape <- mean(
  abs((actual.predic$predicted -
   actual.predic$actual))/actual.predic$actual)
mape
```

```
## [1] 0.159
```

On the average, the error is of about 16%.

Diagnostics

The library `lmtest` permits to run tests to determine

- heteroskedasticity;
- serial autocorrelation;
- normality of errors, and
- correct specification of the model (RESET)

Diagnostics

Heteroskedasticity

The underlying hypothesis test is

$$H_0 : \text{the model is homoskedastic}$$
$$H_1 : \text{the model is heteroskedastic}$$

Function bptest()

Apply the function bptest() to the fitted model.

```
library(lmtest)
bptest(m2)
```

Diagnostics

Heteroskedasticity

```r
bptest(m2)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m2
## BP = 12, df = 7, p-value = 0.1
```

There is no evidence of heteroskedasticity.

Diagnostics

Serial autocorrelation

The purpose of the test is to determine whether there is any linear dependence among terms of the innovations.
We can apply the Durbin-Watson test, `dwtest()`, and the Breusch-Godfrey test, `bgtest()`.

Diagnostics

Serial autocorrelation

```
dwtest(m2)
```

```
##
##  Durbin-Watson test
##
## data:  m2
## DW = 2, p-value = 0.5
## alternative hypothesis: true autocorrelation is gr
```

Result

There is no evidence that $corr(u_t, u_{t-1}) \neq 0$.

Diagnostics

Serial autocorrelation

```
bgtest(m2, order = 10)
```

```
##
##  Breusch-Godfrey test for serial correlation of or
##
## data:  m2
## LM test = 15, df = 10, p-value = 0.1
```

Result

There is no evidence of linear dependence among the first 10 terms of the innovations.

Diagnostics

Normality

There are several normality test available.

- The Shapiro-Wilk test is available in the `base` library (loaded by default), but it cannot be applied to vectors with more than 5,000 observations.
- The Jarque-Bera test is available in the library `tseries`:

```
library(tseries)
```

- The normality of errors is less of a concern when the sample size is sufficiently large.

Diagnostics

Normality: Shapiro-Wilk test

```
shapiro.test(m2$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  m2$residuals
## W = 1, p-value = 0.06
```

Conclusion

There is no evidence of non-normality of errors.

Diagnostics

Jarque-Bera test

```
jarque.bera.test(m2$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  m2$residuals
## X-squared = 3, df = 2, p-value = 0.3
```

Conclusion

There is no evidence of non-normality of errors.

Diagnostics

RESET

In this case the test is not very important, as there are only factors as regressors.

R code


```r
resettest(m2, type = "fitted", power = 2:3)
```

```
##
##  RESET test
##
## data:  m2
## RESET = 0, df1 = 2, df2 = 190, p-value = 1
```

```r
resettest(m2, type = "regressor", power = 2:3)
```

```
##
##  RESET test
##
## data:  m2
## RESET = 0, df1 = 2, df2 = 190, p-value = 1
```

Diagnostics

Plots

We can also apply `plot()` to a `lm()` object to obtain these tests.
It is an interactive plot and it is convenient to do it in the console.

```
plot(m2)
```

Logit models

Logit model

```
modelo.logit <- glm(diagnostic ~ chol + age,
                    data = cleve,
                    family = "binomial")
# summary(modelo.logit)
probs.predicted <- predict(modelo.logit,
                           type = "response")
```

Logit model

```
mu  <- mean(cleve$diagnostic)
probs.predicted2 <-
  ifelse(probs.predicted > mu, 1, 0)
table(cleve$diagnostic,
                  probs.predicted2)/nrow(cleve)
```

```
##    probs.predicted2
##          0     1
##   0 0.323 0.215
##   1 0.148 0.313
```

Logit model

```
probs.predicted3 <-
  ifelse(probs.predicted > .5, 1, 0)
table(cleve$diagnostic,
                  probs.predicted3)/nrow(cleve)
```

```
##    probs.predicted3
##         0     1
##   0 0.374 0.165
##   1 0.226 0.236
```

Time series

Libraries

Needed libraries are `forecast`, `tseries`.

Download them,

```
install.packages("forecast", "tseries")
```

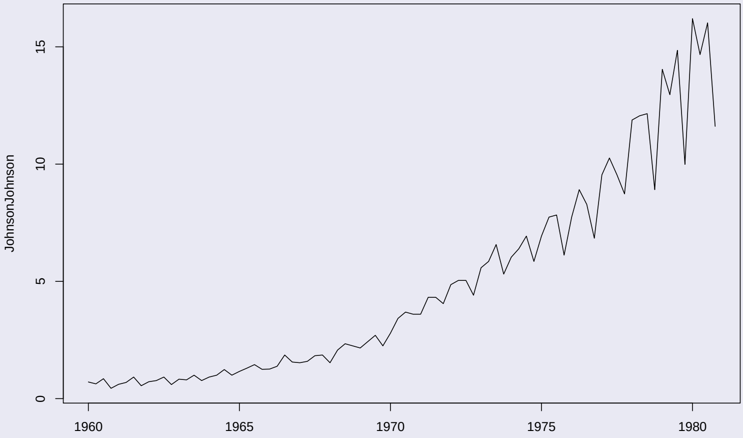
and then load then in your session

```
library(forecast)
library(tseries)
```

Time series

Plots (Multiplicative model)

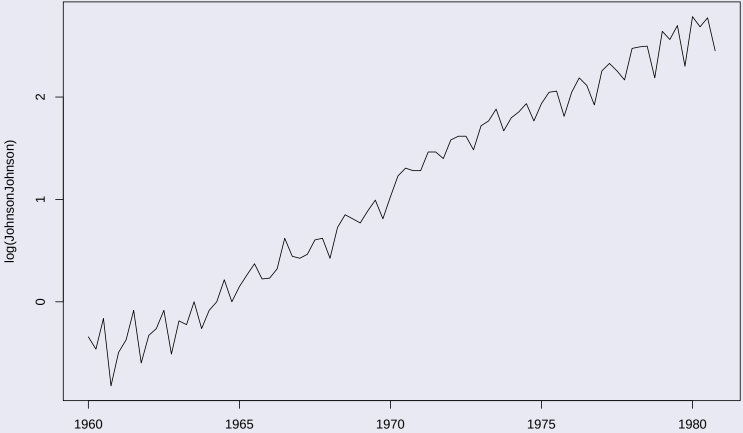
```
plot.ts(JohnsonJohnson)
```



Time series

Plots (Additive model)

```
plot.ts(log(JohnsonJohnson))
```



Exponential smoothing

Let y_t be a stochastic process.

The idea of exponential smoothing is to compute the *one-step ahead forecast*, $\widehat{y}_{T+1|T}$, as a weighted mean of the previous observed terms:

$$\begin{aligned}\widehat{y}_{T+1|T} &= \alpha y_T + \alpha(1-\alpha)y_{T-1} + \cdots \\ &= \alpha y_T + (1-\alpha)\widehat{y}_{T|T-1}, \qquad 0 \le \alpha < 1;\end{aligned}$$

rearranging terms,

$$\widehat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\widehat{y}_{t|t-1},$$

Exponential smoothing

which gives

$$\begin{aligned}\widehat{y}_{t+1|t} &= \alpha y_t + (1-\alpha)\widehat{y}_{t|t-1} = \widehat{y}_{t|t-1} + \alpha(y_t - \widehat{y}_{t|t-1}) \\ &= \widehat{y}_{t|t-1} + \alpha e_t,\end{aligned}$$

where $e_t = y_t - \widehat{y}_{t|t-1}$ is the forecast error.

The value α is estimated optimizing the errors squared.

The work by Holt & Winters allowed the inclusion of seasonal, s_t, and trending, b_t, terms, beside the level term, ℓ_t.

Exponential smoothing

Holt-Winters Modeling (Additive model)

$$
\begin{aligned}
y_{t+h|t} &= \ell_t + hb_t + s_{t-m+h_m^+} \\
\ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} - b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}
\end{aligned}
$$

where m denotes the seasonality period (per year)
The symbol

$$h_m^+ = \lfloor (h-1) \quad mod(m) \rfloor + 1$$

makes sure that the estimation of the seasonality is the last year of the sample.

Exponential smoothing

Estimation

```
logJJ.forecast <- HoltWinters(
  log(JohnsonJohnson), beta = TRUE, gamma = TRUE)
logJJ.forecast
```

Exponential smoothing

Estimation

```
logJJ.forecast$coefficients[1]
```

```
##    a
## 2.61
```

```
logJJ.forecast$SSE # measure of estimate error
```

```
## [1] 0.661
```

Exponential smoothing

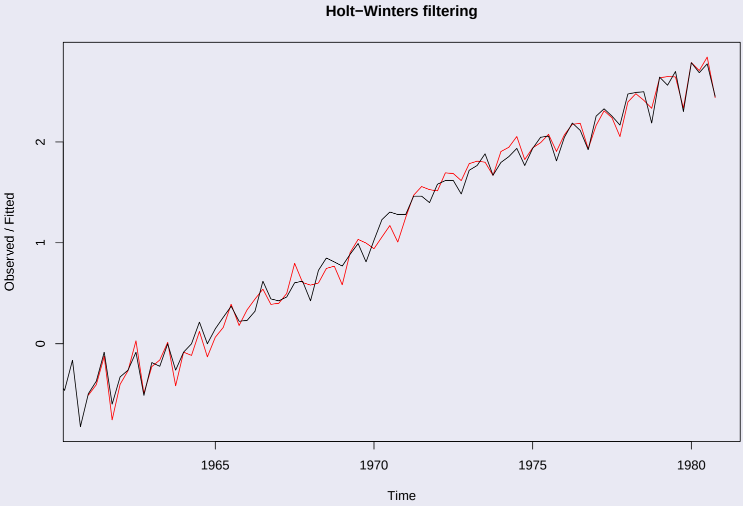
Plots

```
plot(logJJ.forecast)
```

The original series is plotted in black and the forecast is in red.

Exponential smoothing

Plots



Exponential smoothing

Estimate with an initial value

```
logJJ.forecast2 <- HoltWinters(
  log(JohnsonJohnson), beta = TRUE, gamma =
    TRUE, l.start = .91) # arbitrary initial value
logJJ.forecast2
```

Exponential smoothing

Estimate with an initial value

```
logJJ.forecast2$coefficients[1]
```

```
##    a
## 2.68
```

```
logJJ.forecast$coefficients[1]
```

```
##    a
## 2.61
```

```
logJJ.forecast$SSE # measure of estimate error
```

```
## [1] 0.661
```

```
logJJ.forecast2$SSE
```

```
## [1] 6.01
```

Exponential smoothing

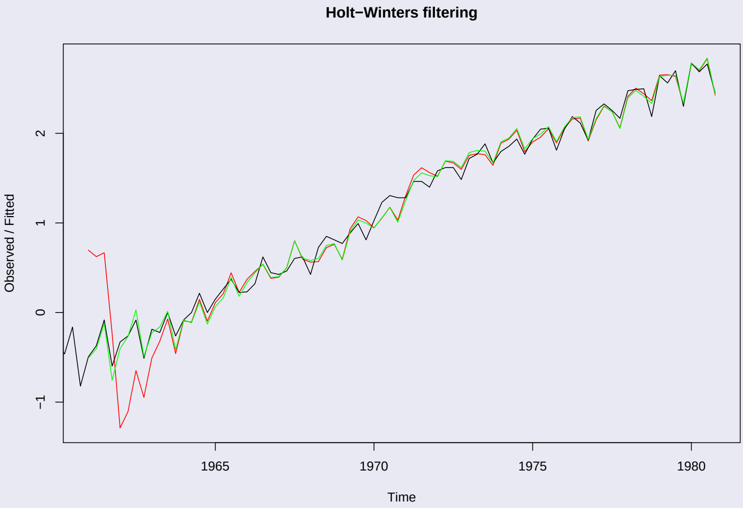
Comparing plots

```
plot(logJJ.forecast2)
lines(logJJ.forecast$fitted[,1], col = "green")
```

The object is a matrix whose columns are, respectively, the fitted time series; its level part; its trend part and the seasonality. We select above the fitted part.

Exponential smoothing

Comparing plots



Exponential smoothing

Forecast

Forecast can be performed using the library `forecast`

```
library(forecast)
```

Exponential smoothing

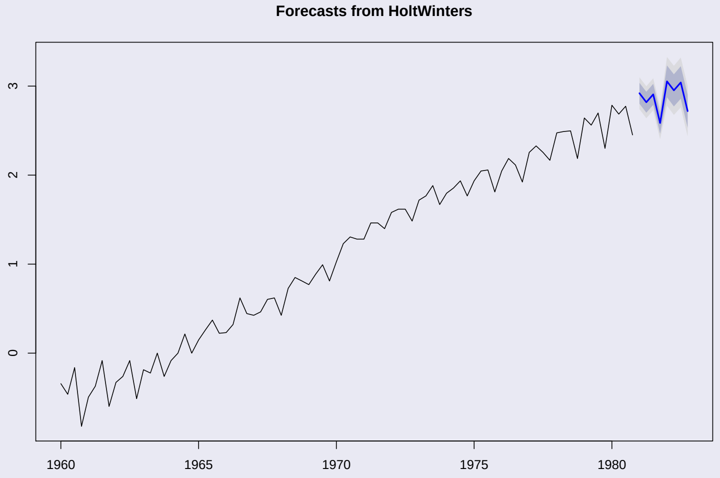
Forecast

```
logJJ.forecast8095 <-
  forecast.HoltWinters(logJJ.forecast, h=8)
```

- We use the first estimate, `logJJ.forecast`, as it is proven to be the better.
- The parameter `h=8` will forecast the estimate **eight** periods in the future (two years).
- The forecast also plots a confidence interval (80%; 95% is the default).

Exponential smoothing

Forecast: Plot



Exponential smoothing

Forecast: Plot

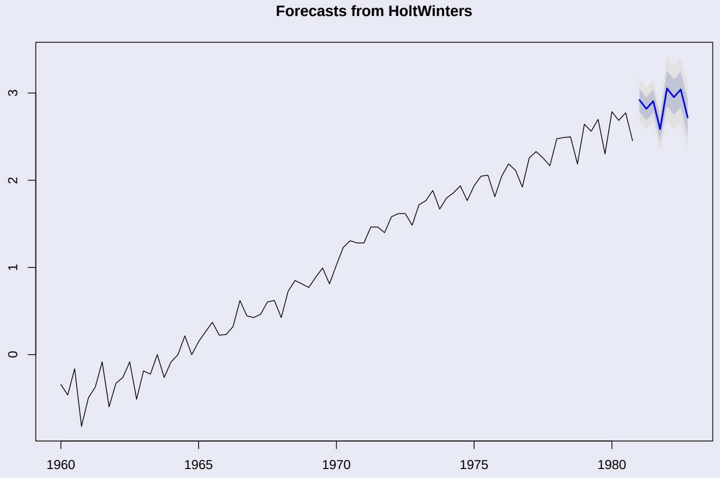
```
logJJ.forecast8599 <- forecast(
  logJJ.forecast, h=8, level = c(85,99))
plot(logJJ.forecast8599)
```

Exponential smoothing

Introduction to R

Lino AA Notarantonio (lino@tec.mx)

Forecast: Plot



Exponential smoothing

Diagnostics

```
library(tseries)
ts8095.fitted <- as.ts(logJJ.forecast8095$fitted)
ts8095.fitted <- na.omit(ts8095.fitted)
ts8095.residuals <-
  as.ts(logJJ.forecast8095$residuals)
ts8095.residuals <-
  na.omit(as.ts(logJJ.forecast8095$residuals))
```

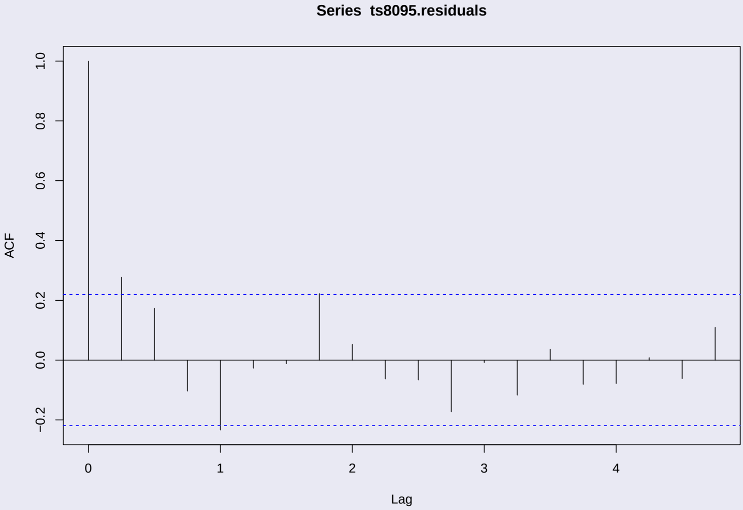
Exponential smoothing

Diagnostics: ACF

```
acf(ts8095.residuals)
```

Exponential smoothing

Diagnostics: ACF



Exponential smoothing

Diagnostics: Ljung-Box Test

```
Box.test(ts8095.residuals, lag = 20)
```

```
##
##  Box-Pierce test
##
## data:  ts8095.residuals
## X-squared = 25, df = 20, p-value = 0.2
```

Conclusion

We do not reject the null hypothesis of no autocorrelation (for the first 20 lags).

Exponential smoothing

Diagnostics: Augmented Dickey-Fuller Test

```
adf.test(ts8095.fitted)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts8095.fitted
## Dickey-Fuller = -1, Lag order = 4, p-value = 0.8
## alternative hypothesis: stationary
```

Conclusion

There is evidence that the errors are not white noise.

Box-Jenkins approach

See the script `BoxJenkins.R`