# Introduction to R

Lino AA Notarantonio (lino@tec.mx)

30/07/2020

# Timetable and material



| Day | hour |
|---|---|
| Thursday | 16:00-19:00 hours |
| Friday | 10:00-13:00 hours |
| Saturday | 11:00-14:00 hours |

There will a 30-minute break during each session.

The material of this workshop is here

# Introduction to R(Studio)

# R(Studio)



- You should have defined a project (working directory) somewhere in your computer (*e.g.*, Desktop).
- 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 $= 4$) $= 2$

# Display Information



- Getting help: in the console, put a question mark before the function name; RStudio will display the documentation: `?cos`
- 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`
- To type it, you can use the shortcut `ALT+"-"` (ALT key and minus sign); it works with Windows, MacOS.
- 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()` (concatenate) 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
## [26] 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
## [26]  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
- Arithmetically, `TRUE` is considered 1 and `FALSE` 0: `sum(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
  selects which entries satisfy the condition. Indexing of vectors starts from 1.
- `y[which(y <= 4)]`: 1, -2, 4, 2, 1
  returns the values of the entries satisfying the condition.
- `which(y == max(y))`: 5
  entry of the vector with the maximum value
- `y[which(y == max(y))]`: 9
  maximum value of the vector

# 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 subtraction 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



```
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



```
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
```

- The dot product can also be implemented as

```
sum(a*b)
```

```
## [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



```r
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.
  - Download the file, change its name to `cleve.csv` and move it to the project (working directory).
  - The names of the variables will be assigned later.
- The dataset will be called `cleve` hereinafter in the presentation.
- The dataset format is `csv` (comma separated variables).

# Import the dataset with RStudio



1. Open an R script and save it in the project.
2. Write in the script
   `cleve <- read.csv("cleve.csv", header = FALSE)`
   and hit **CTRL + Enter** (Windows); **CMD + Enter** (Mac OSX) to execute.
   a. If you get an error in the console, you may be not in the project (working directory)
3. Type `head(cleve)` in the script and execute the instruction. It will show in the console the first six observations.

# Working with the dataset



- Set the names of the variables. Copy the instructions and paste it in the script:

```
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
- The gender variable `sex`: 0 for females, 1 for males.

# Working with the dataset



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

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

- On the other hand, if we work with several variables in a dataset, it is better to use `attach(namedataset)` at the beginning; once done, use `detach(namedataset)`.

# 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
```

## Remark

R works in memory (RAM), so only the dataset `cleve` loaded in memory is changed. The file cleve.csv with the original observations is not changed (reproducibility in research).

# 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:

```
with(cleve, xtabs(~ sex + diagnostic))
```

```
##     diagnostic
## 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:

```
with(cleve, xtabs(~ sex + diagnostic)/nrow(cleve))
```

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

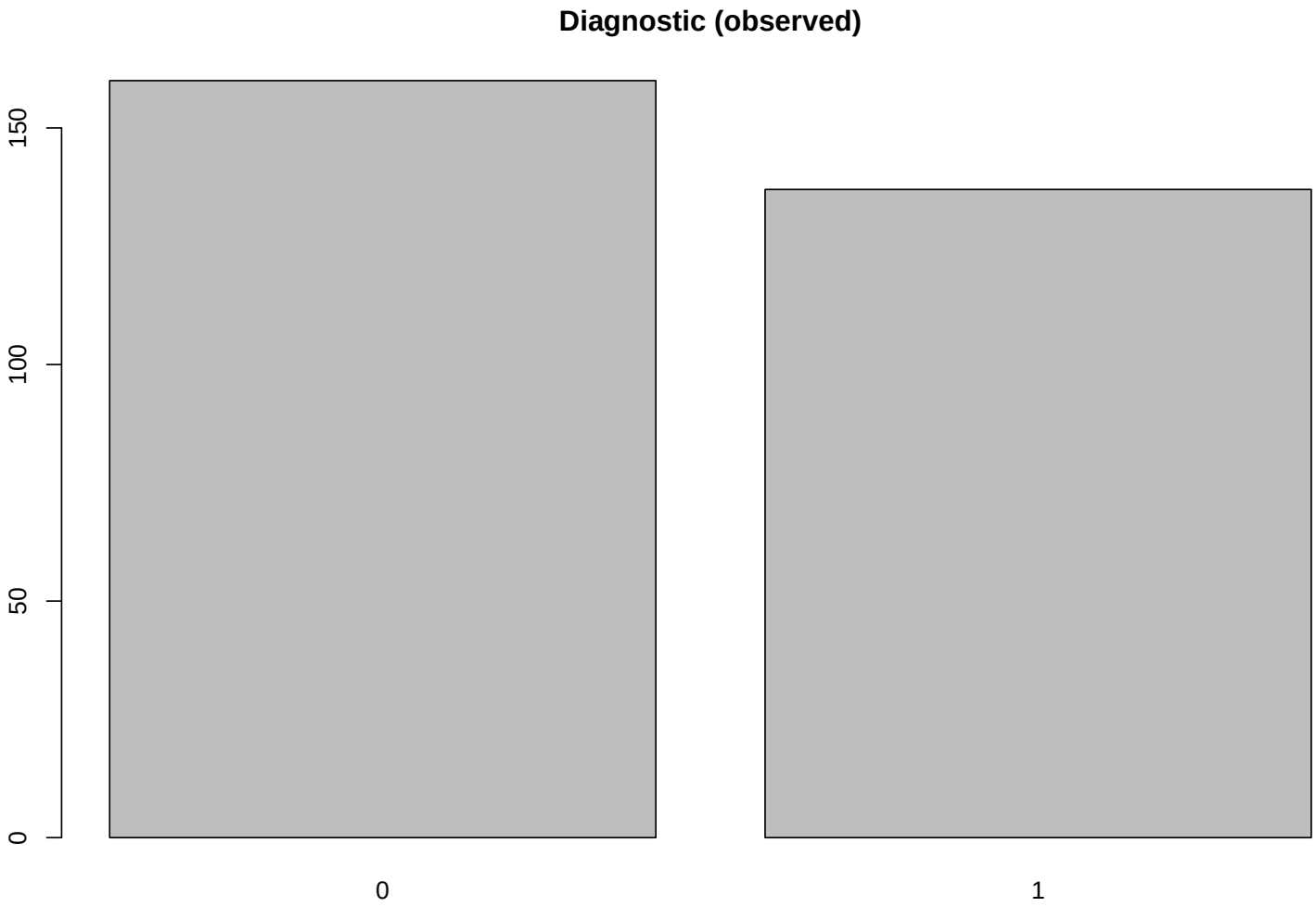
# Visualization: Barplot



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

# Visualization: Barplot





# Visualization: Barplot



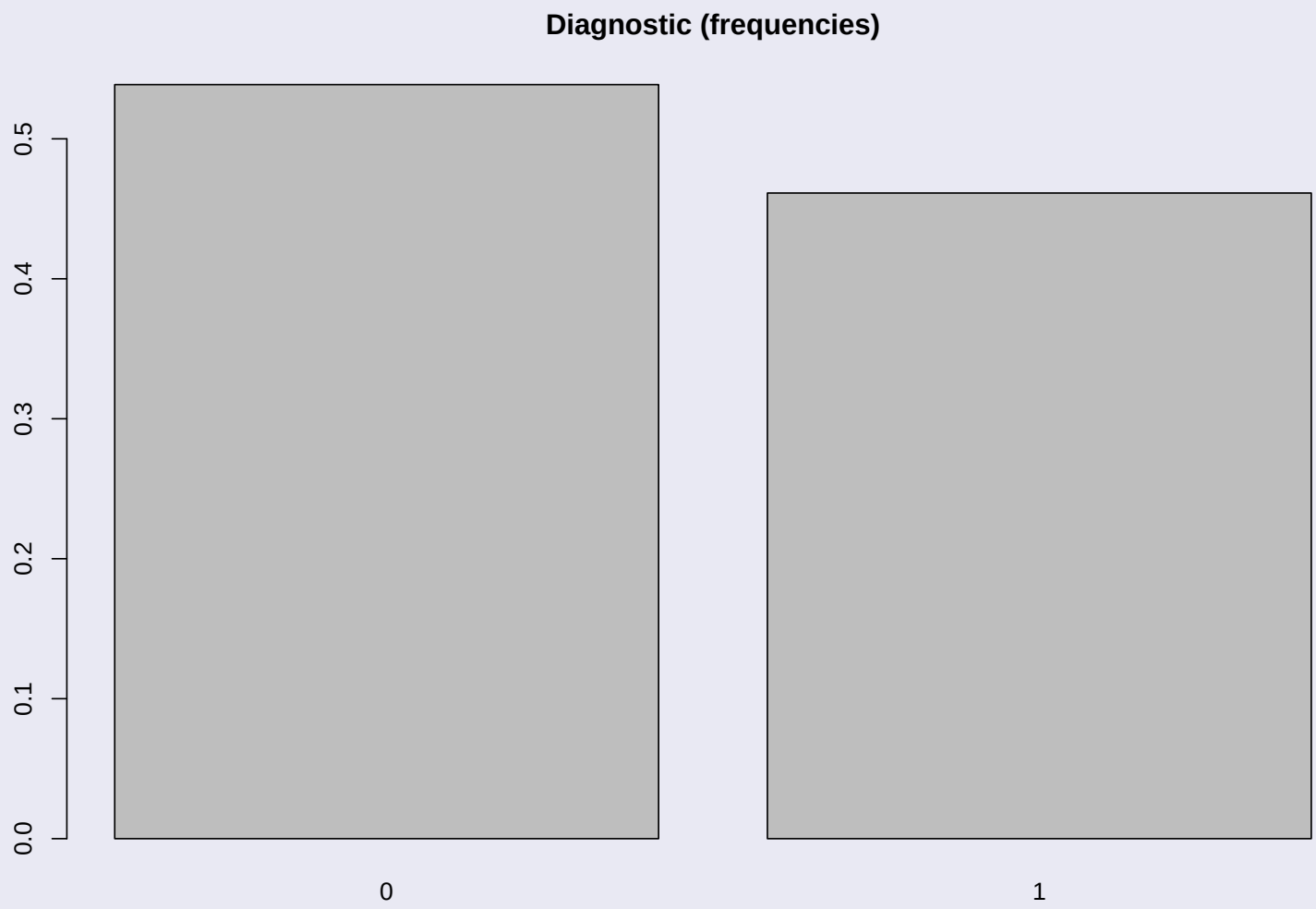
## Barplot with observed frequencies

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

# Visualization: Barplot



## Barplot with observed frequencies



# Visualization: Barplot



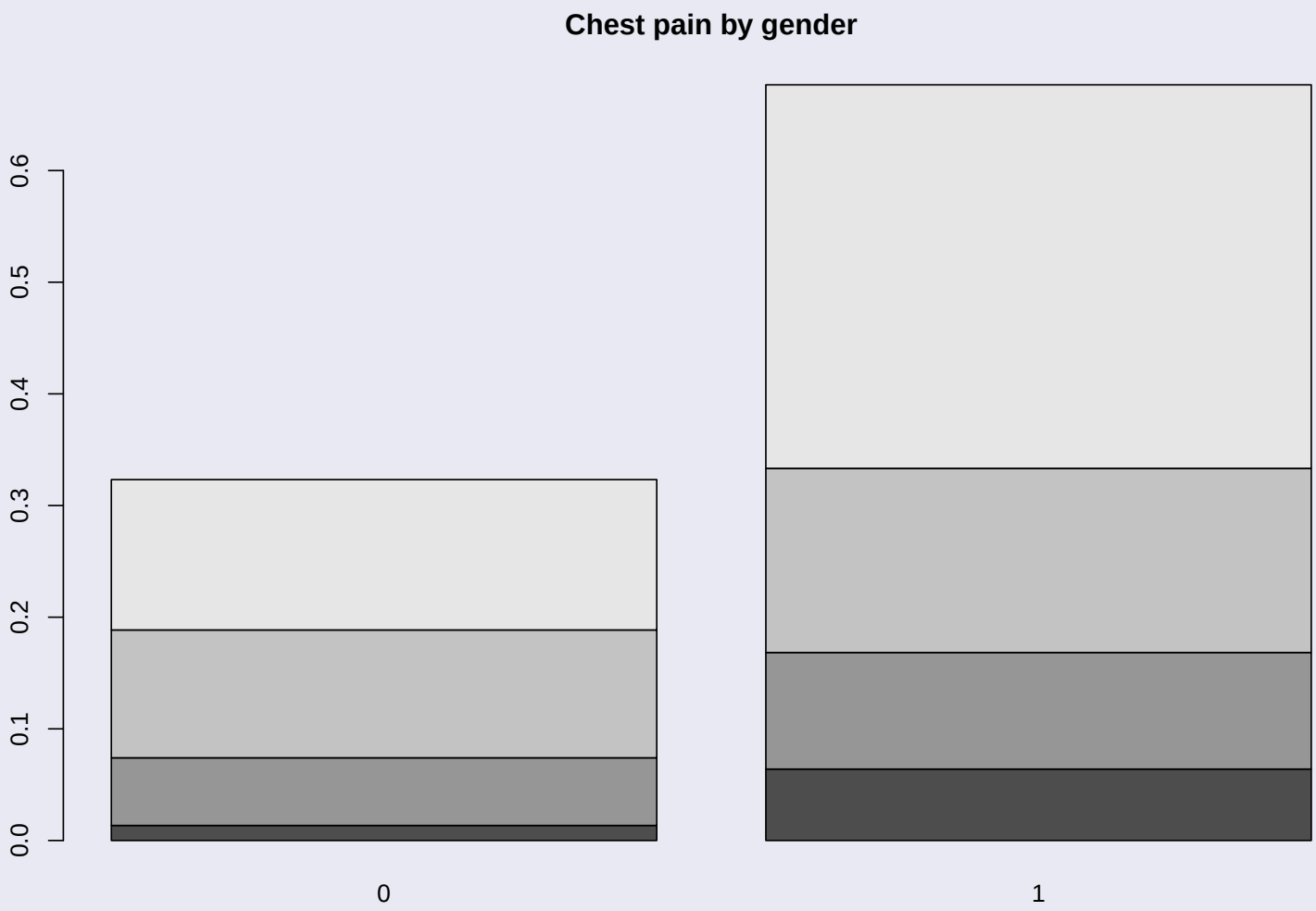
## Barplot with a contingency table

```
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



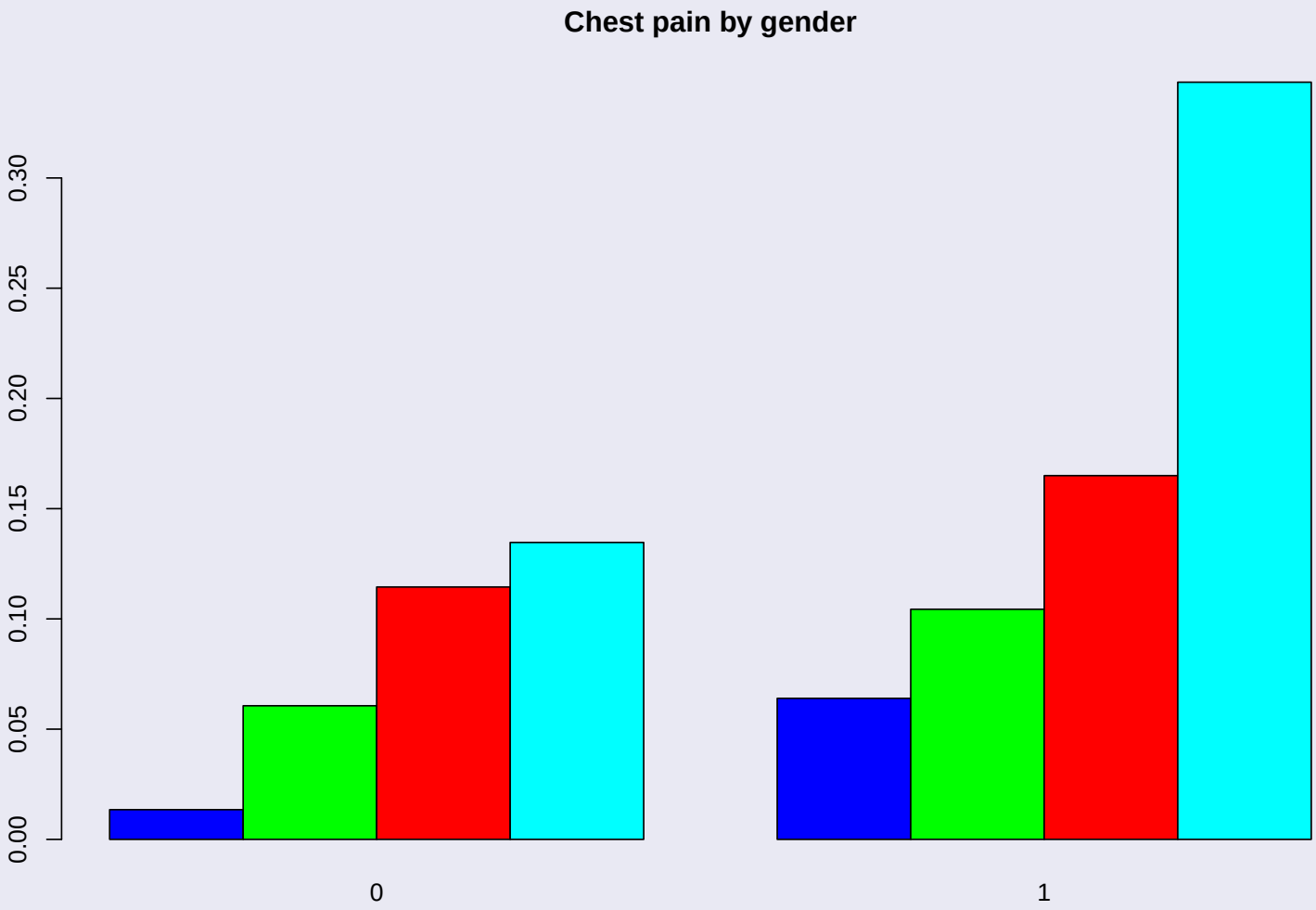
## Grouped barplots with colors

```
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



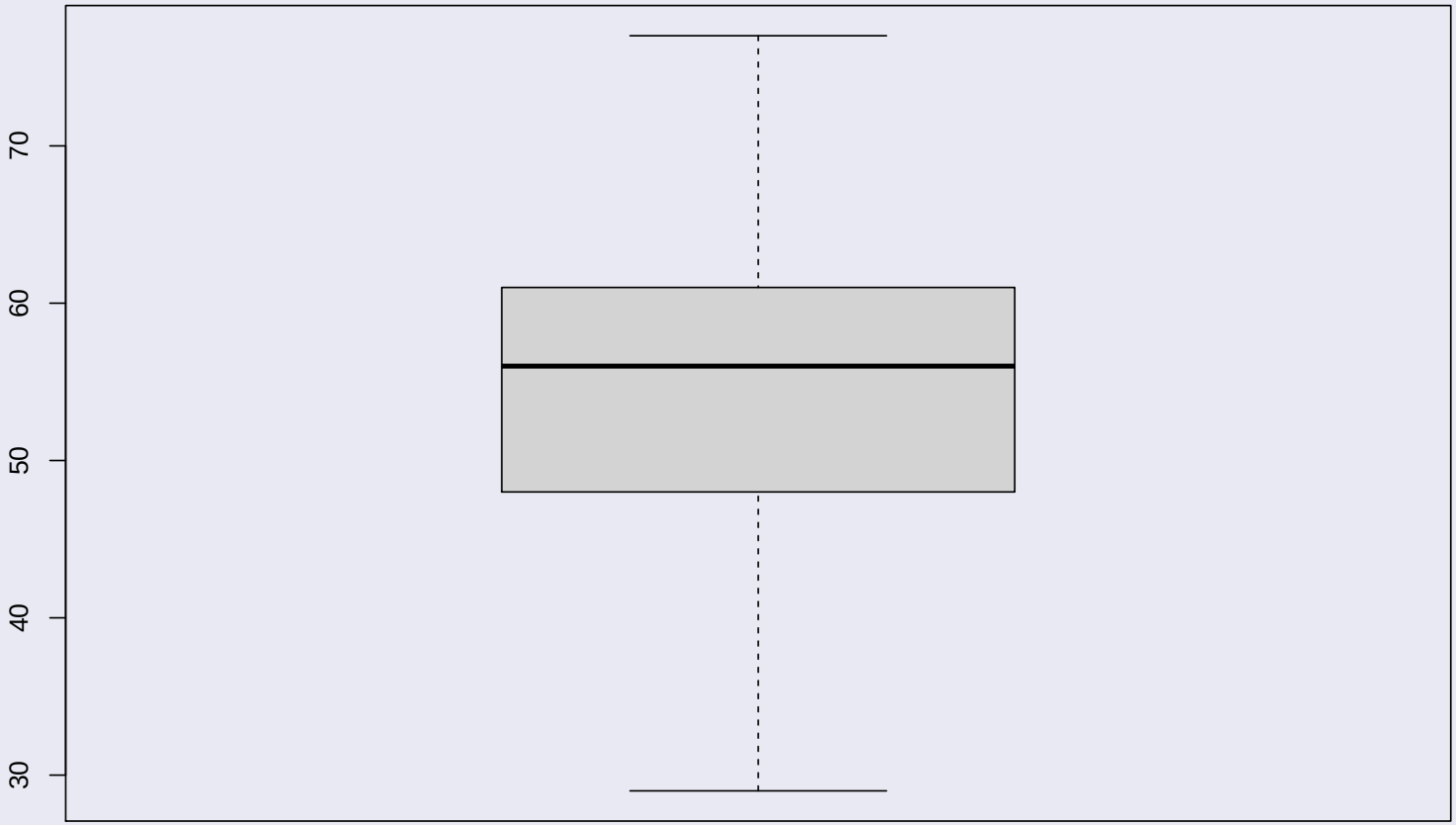
## Variable "age"

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

# Visualization: Boxplot



## Variable "age"



# Visualization: Boxplot



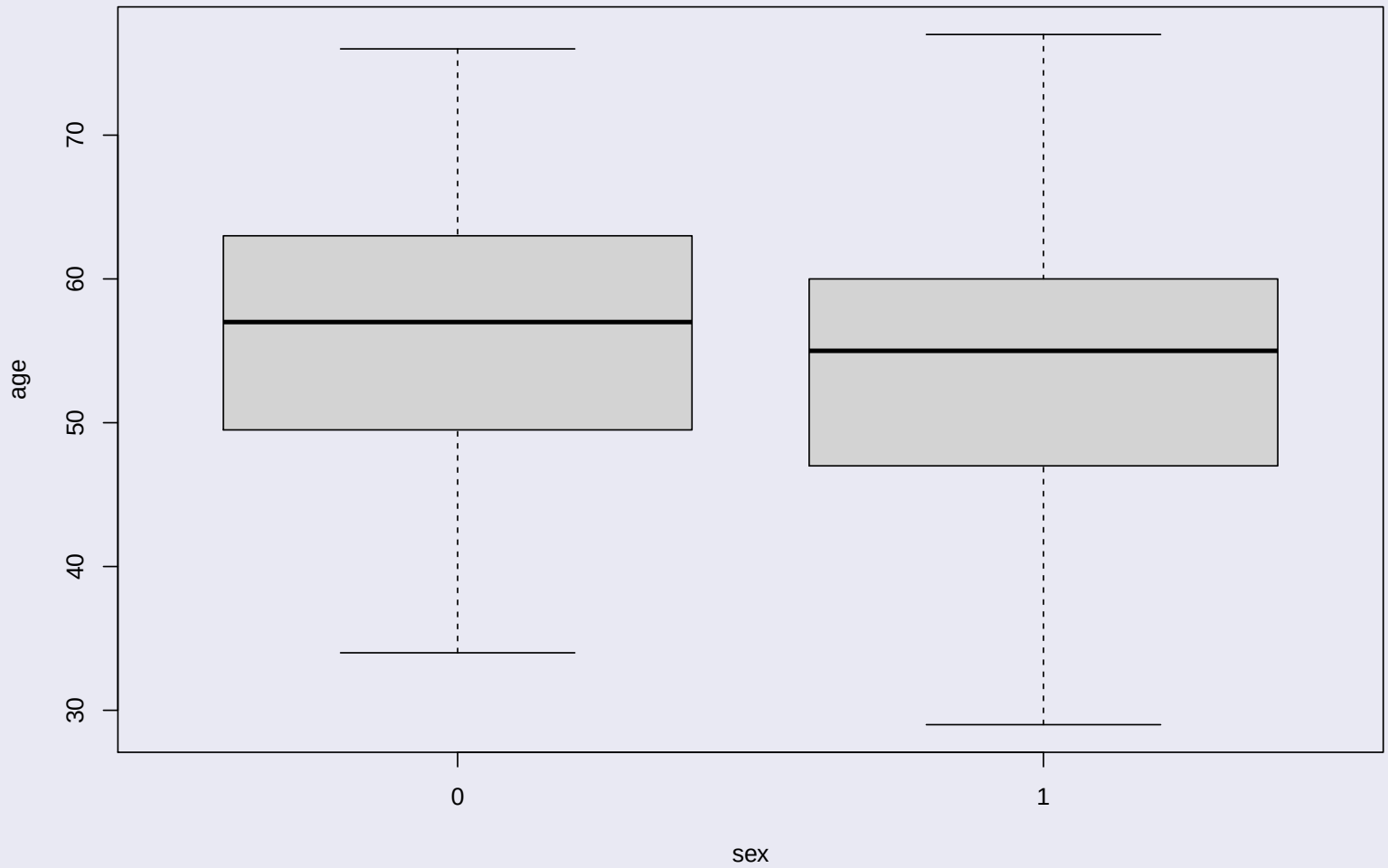
## Age as a function of gender

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

# Visualization: Boxplot



## Age as a function of gender



# Visualization: Boxplot



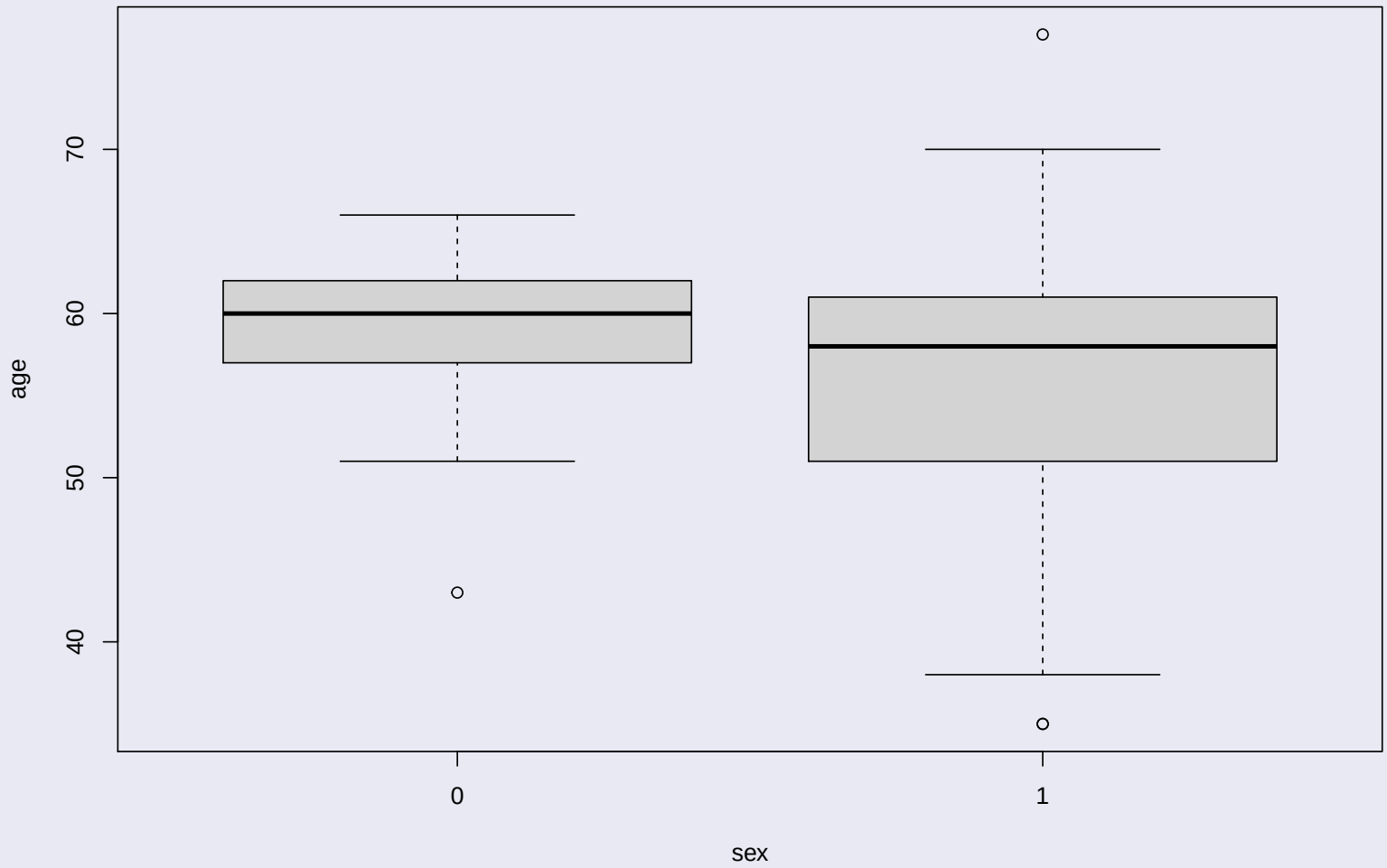
## Age as a function of gender, with positive diagnosis

```
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)
```

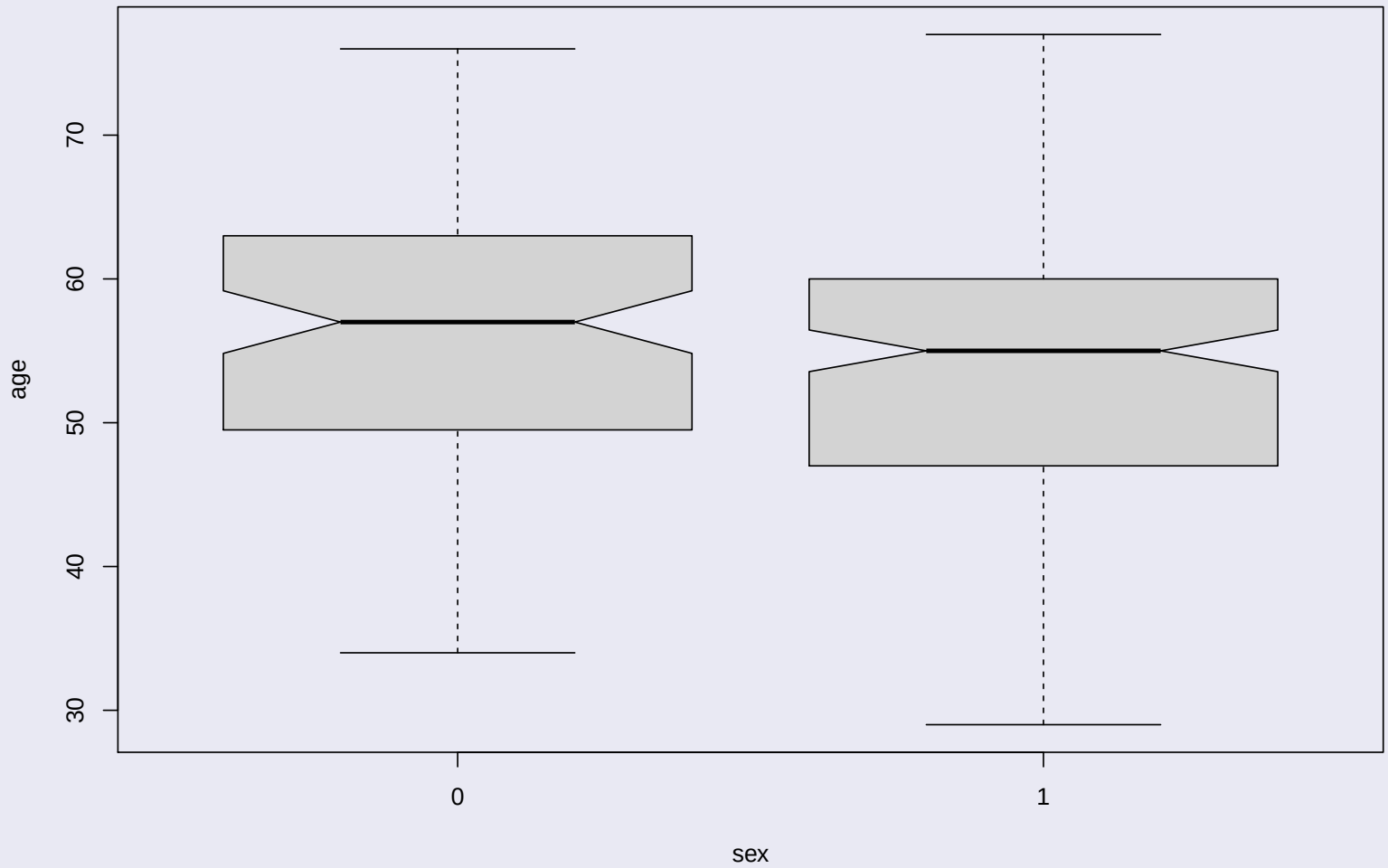
## Justification

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



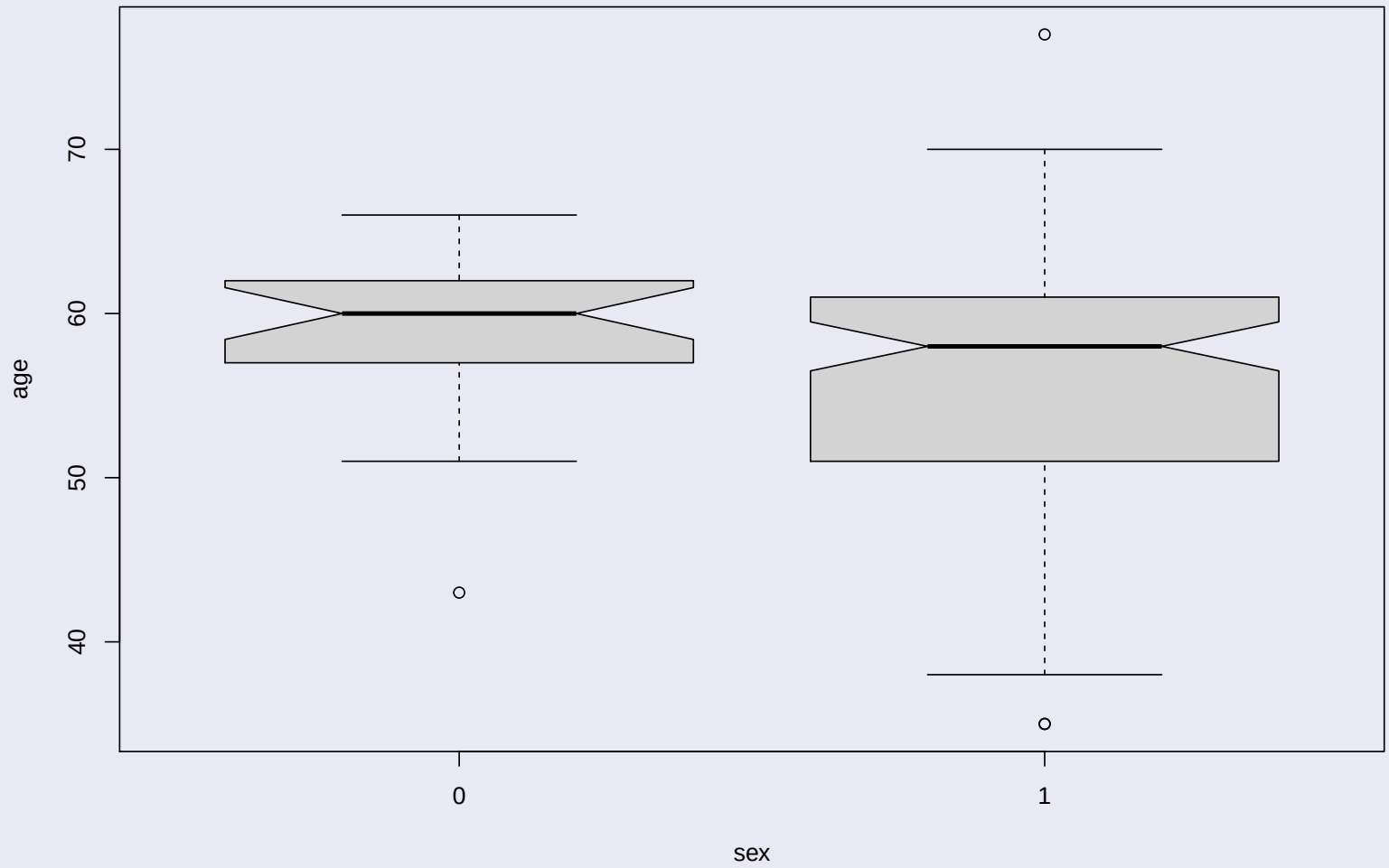
## Age as a function of gender with positive diagnosis

```
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



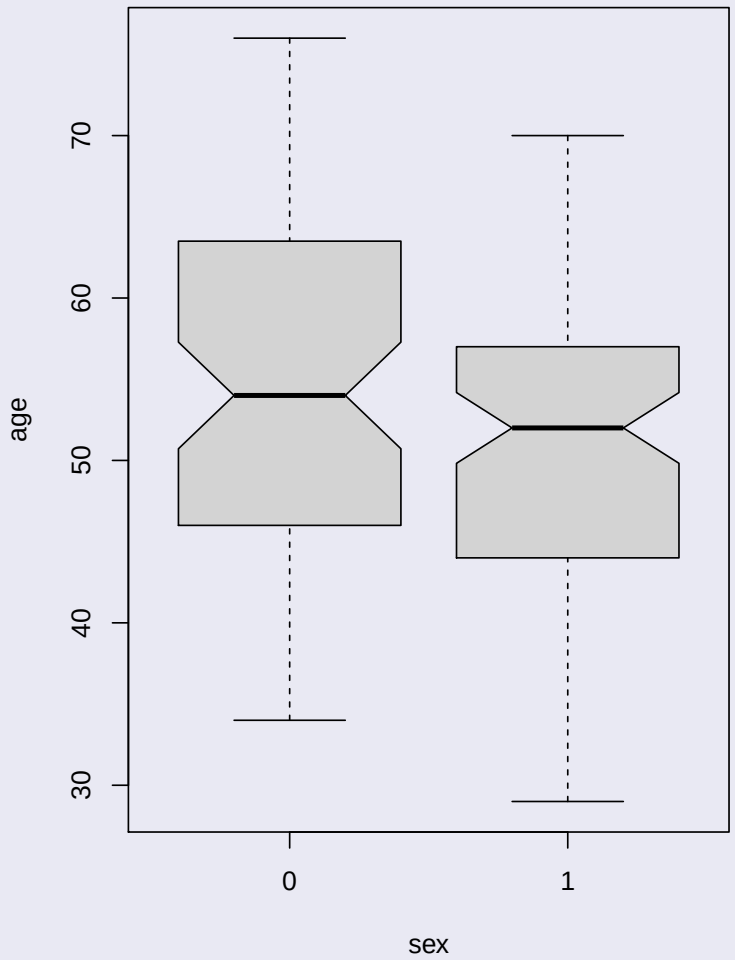
## Two graphs side by side

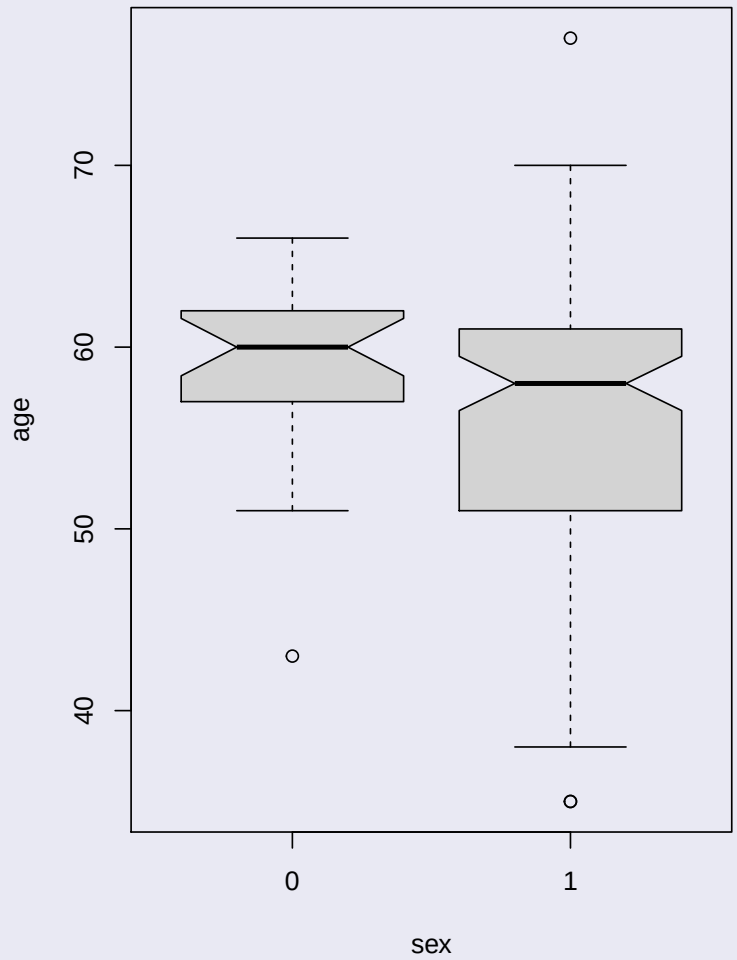
```
par(mfrow=c(1,2))
boxplot(age ~ sex, data = cleve,
        notch = TRUE,
        subset = diagnostic == 0)
boxplot(age ~ sex, data = cleve,
        notch = TRUE,
        subset = diagnostic == 1)
par(mfrow = c(1,1))
```

# Visualization: Boxplot



## Two graphs side by side



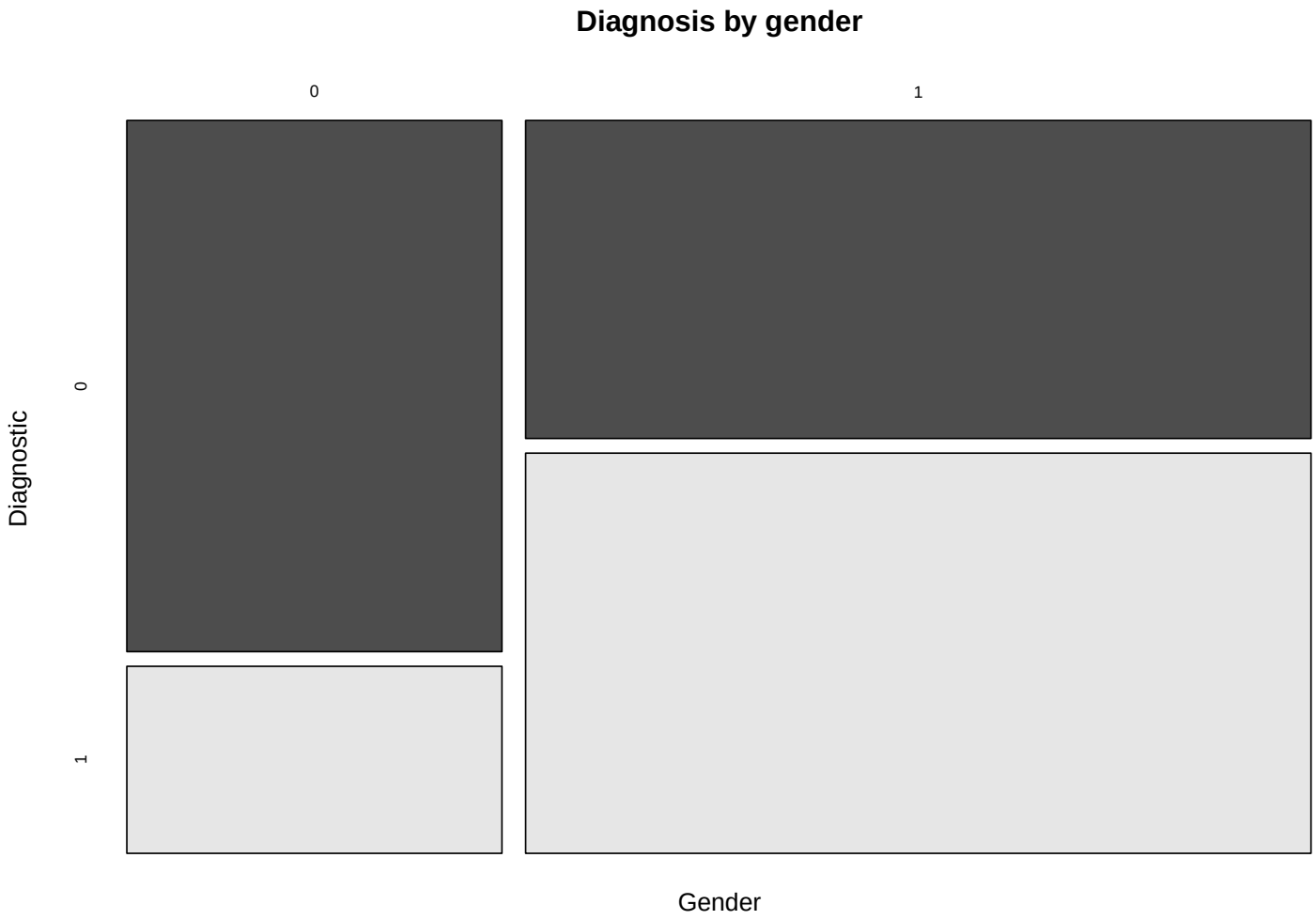
# Visualization: Mosaicplot



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

# Visualization: Mosaicplot





# Visualization: Mosaicplot

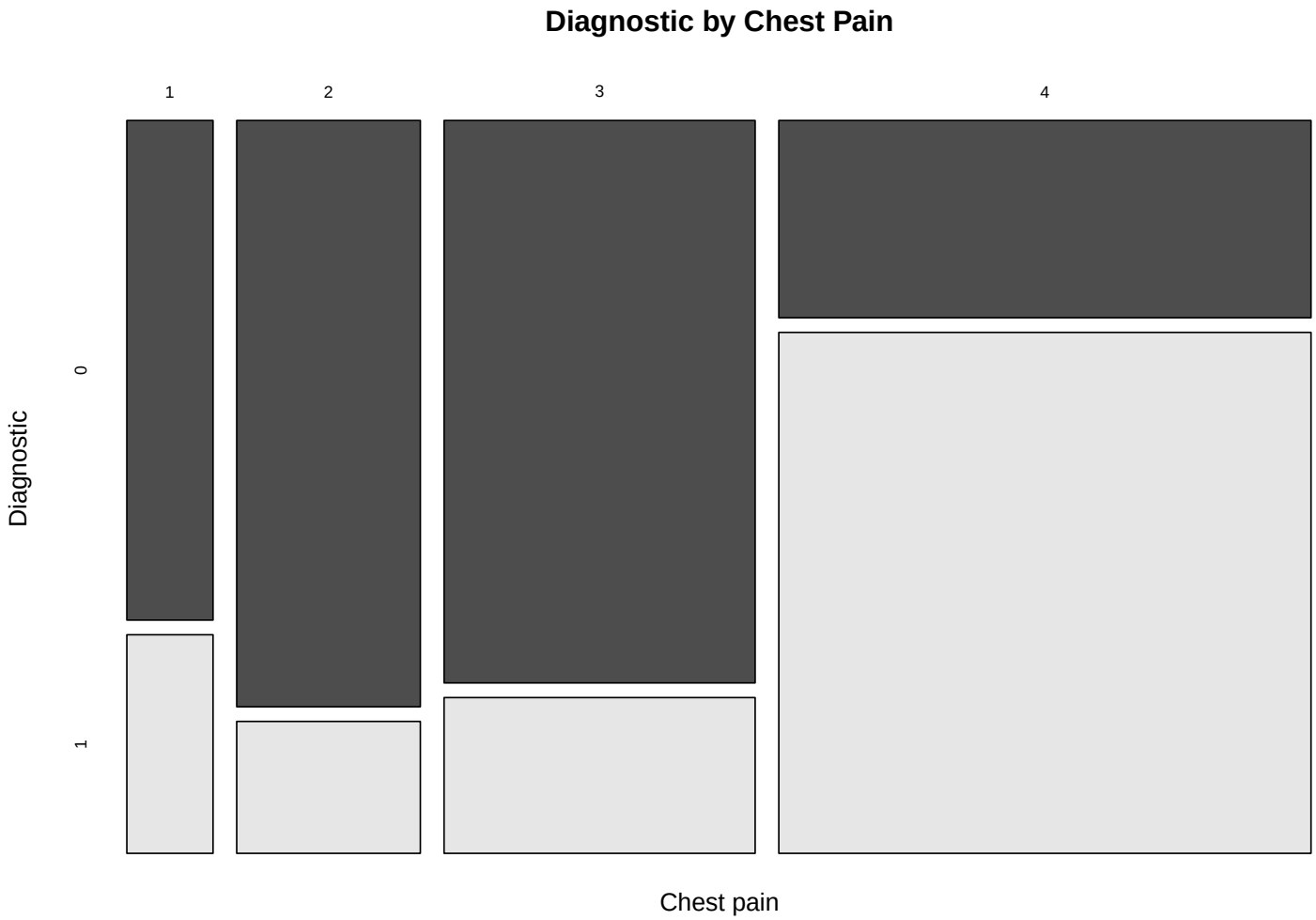


```
mosaicplot(cleve$cp ~ cleve$diagnostic,
           shade=F, color=T,
  xlab = "Chest pain", ylab = "Diagnostic",
  main = "Diagnostic by Chest Pain")
```

The variable cp has four levels.

# Visualization: Mosaicplot





# Visualization: Mosaicplot

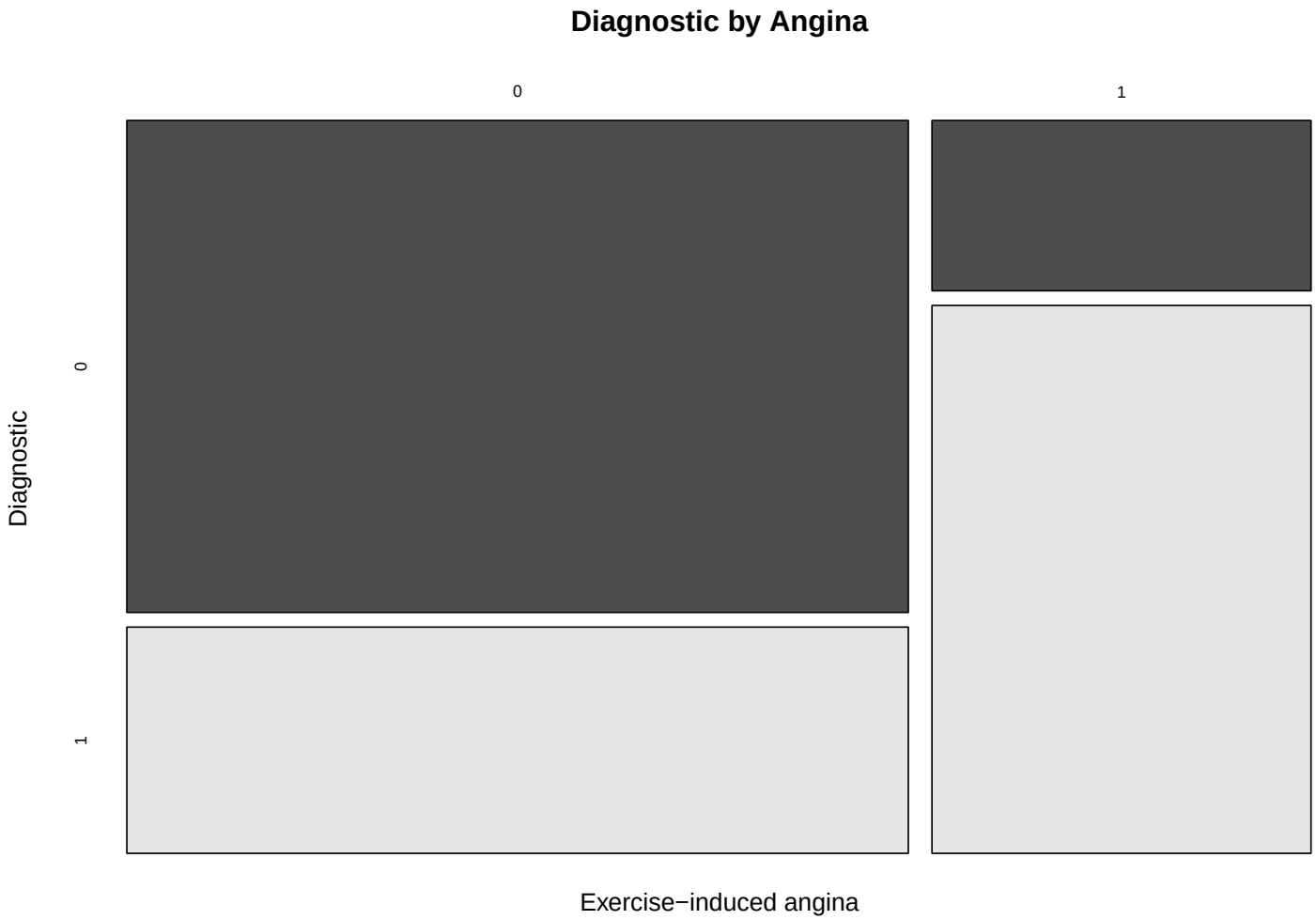


```
mosaicplot(cleve$exang ~ cleve$diagnostic,
           shade=F, color=T,
  xlab = "Exercise-induced angina",
  ylab = "Diagnostic",
  main = "Diagnostic by Angina")
```

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

# Visualization: Mosaicplot





# Visualization of continuous variables



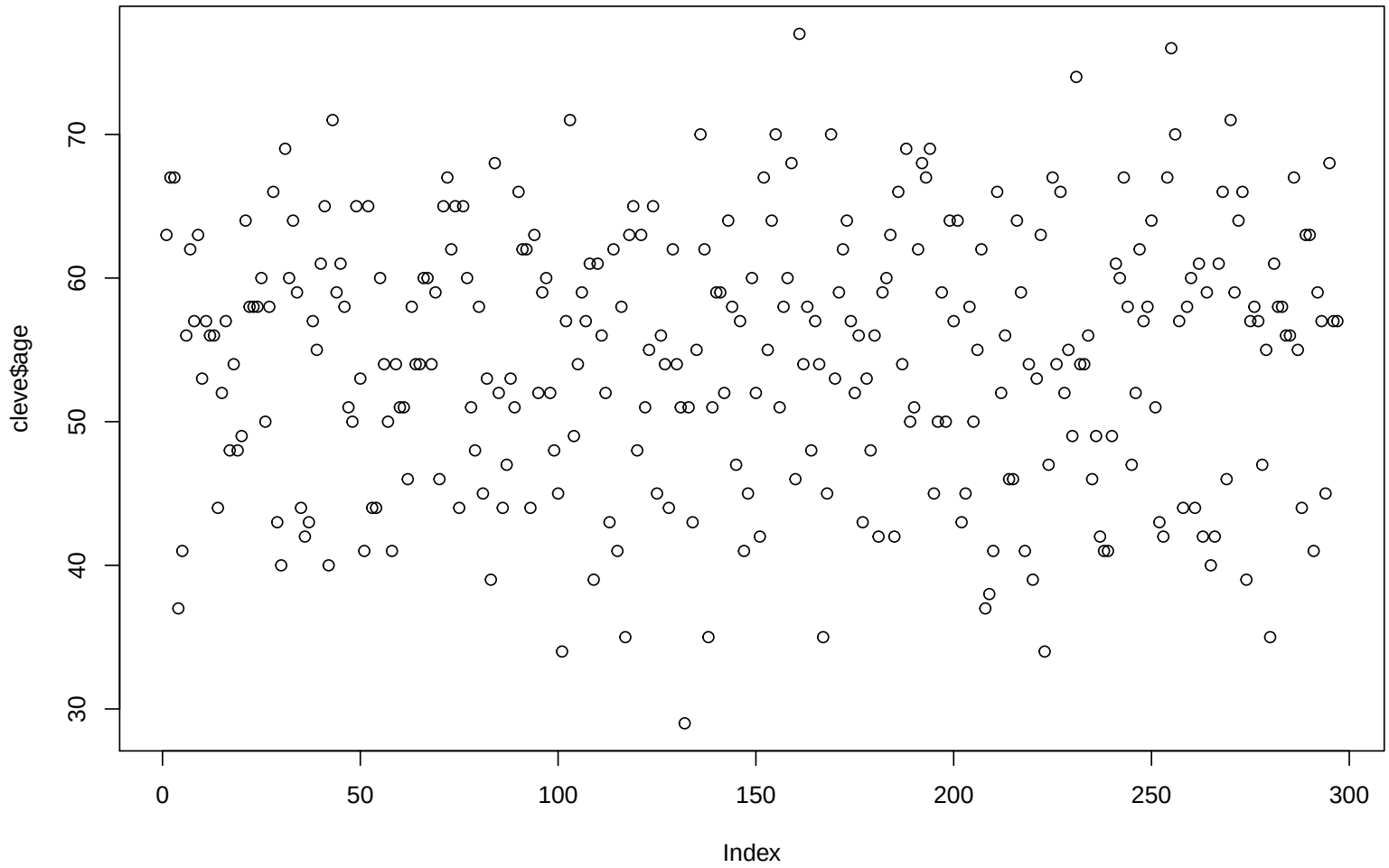
Consider the variable *age*.

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

gives the scatterplot of the variable.

# Visualization of continuous variables





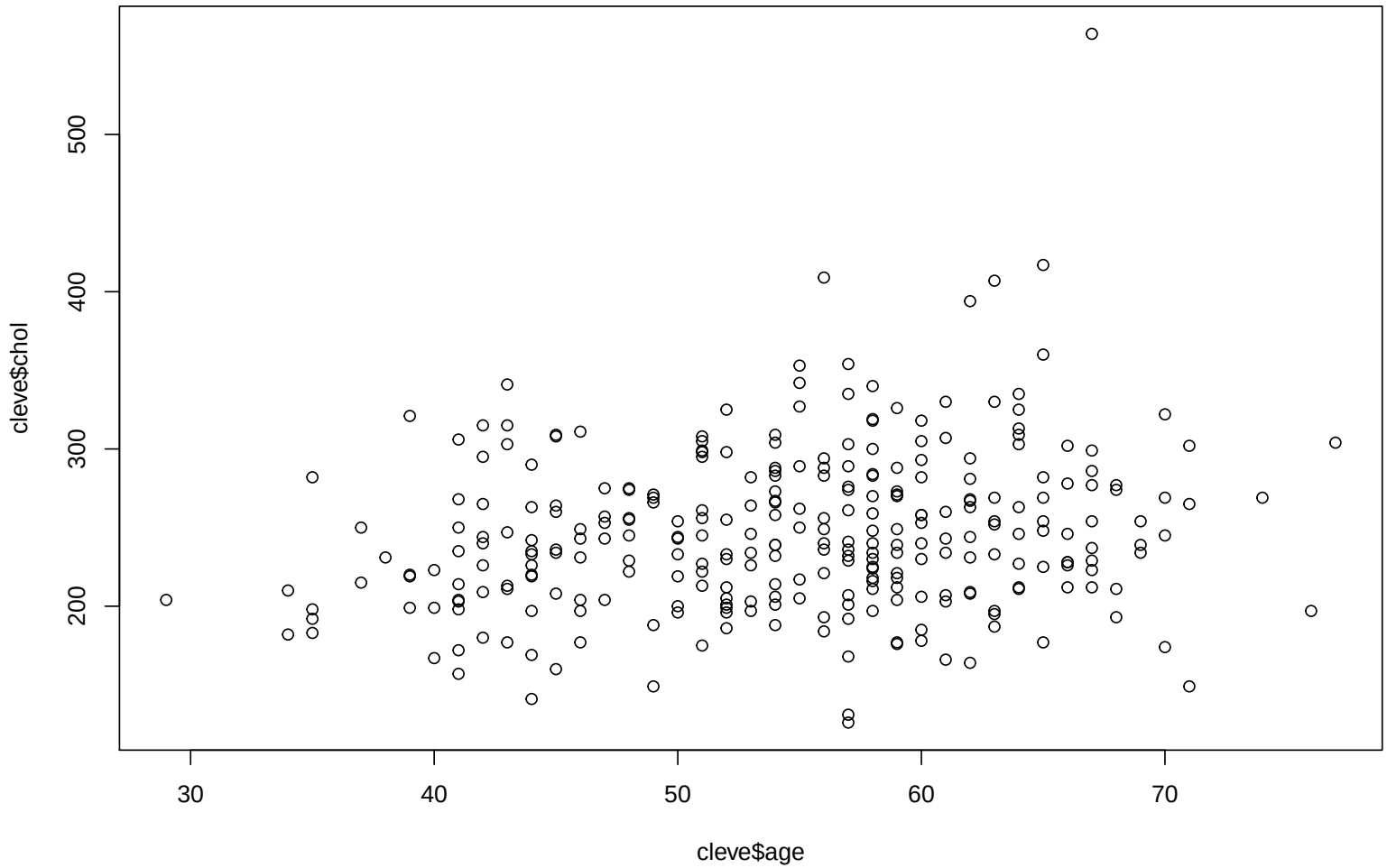
# Visualization of continuous variables



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

# Visualization of continuous variables





# Visualization of continuous variables

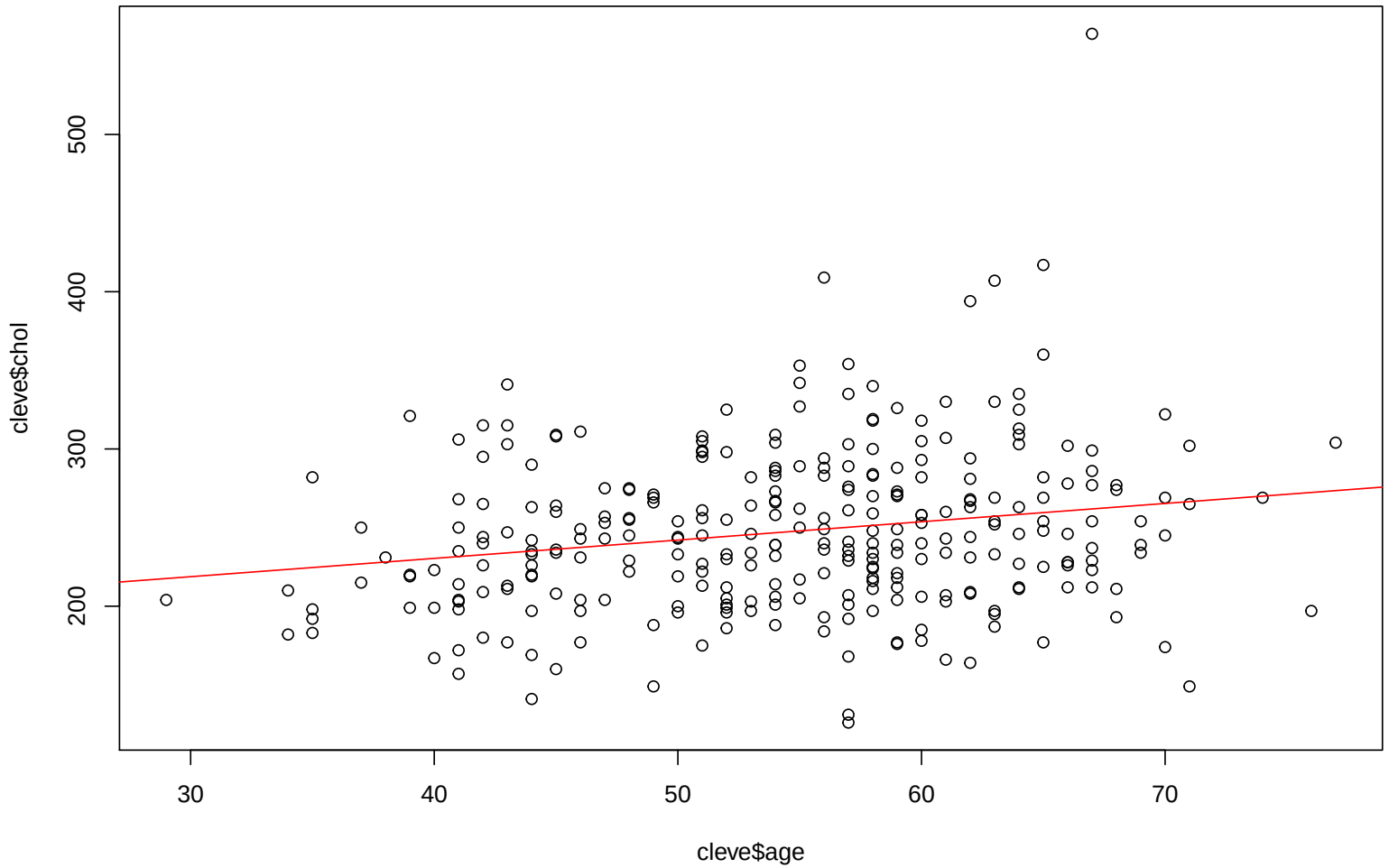


With regression line:

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

# Visualization of continuous variables





# 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[1].

```
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: $\alpha = .05$)

[1] We set 'diagnostic=1' if some heart problem are present, using 'cleve$diagnostic[cleve$diagnostic > 0 ] <- 1' in slide 65.

# 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, $\alpha$, $sd$, $n$ (sample size), $\delta$ (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
```

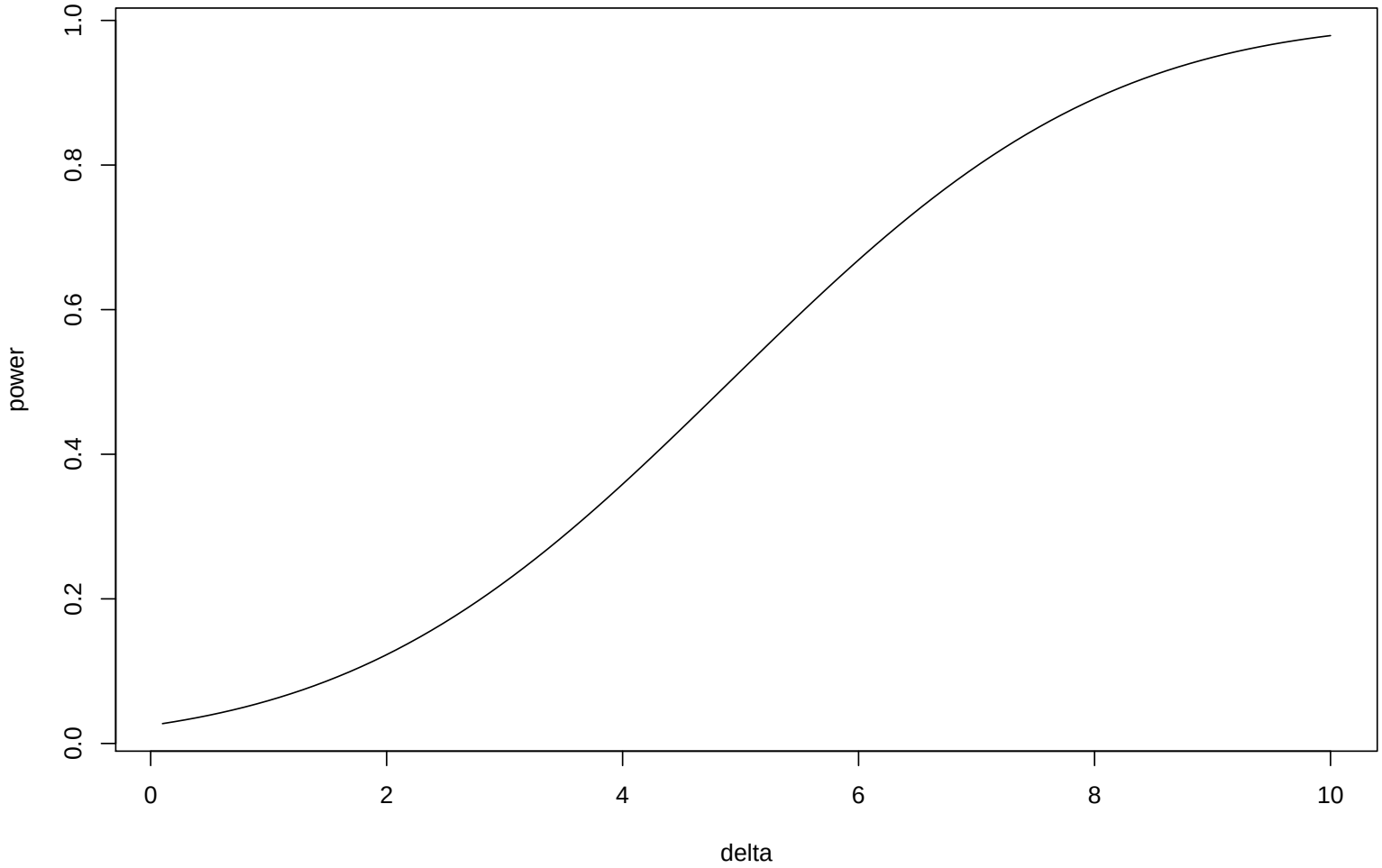
# Power curve



```
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



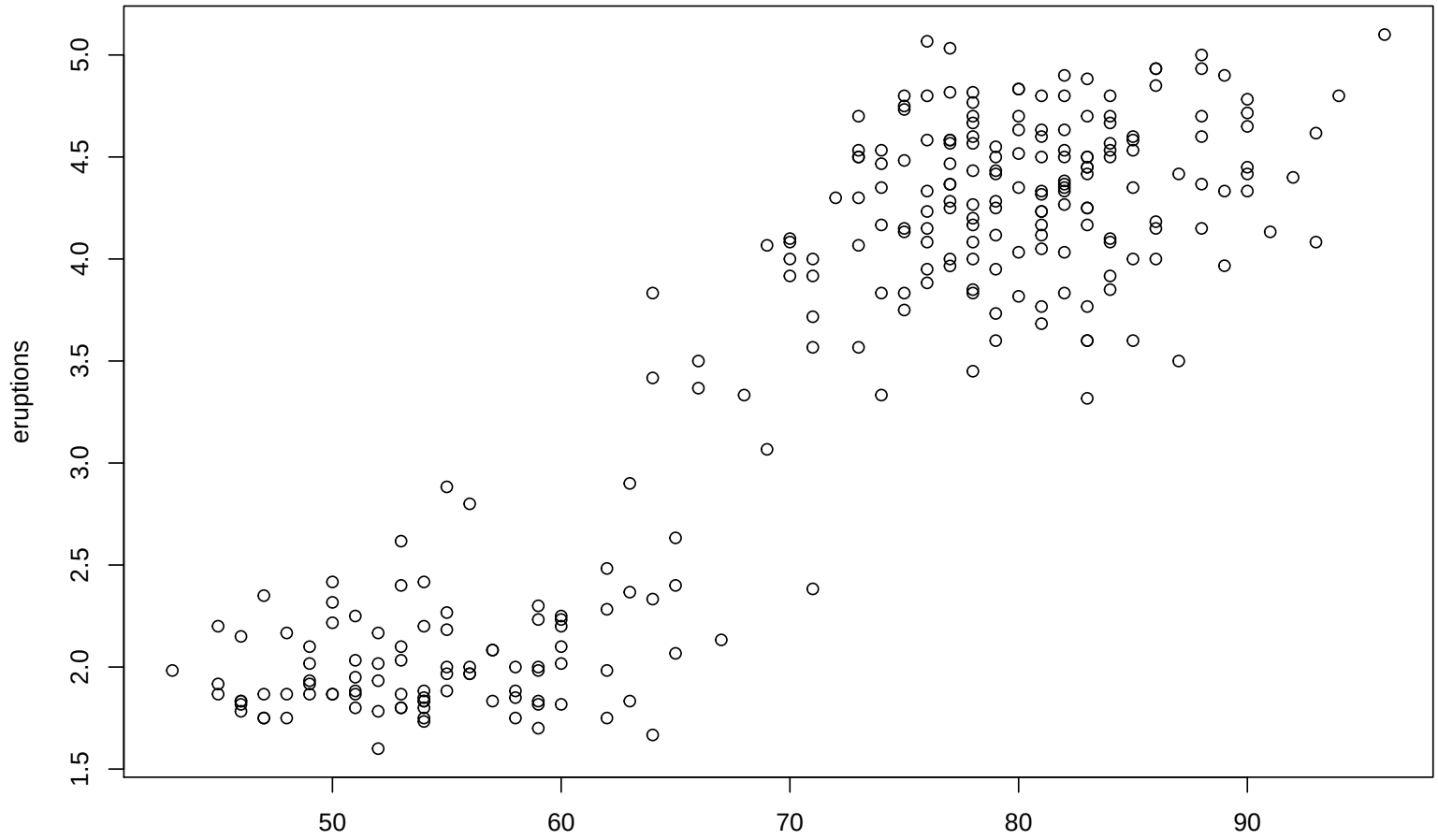


# Linear regression models

# Simple linear regression model



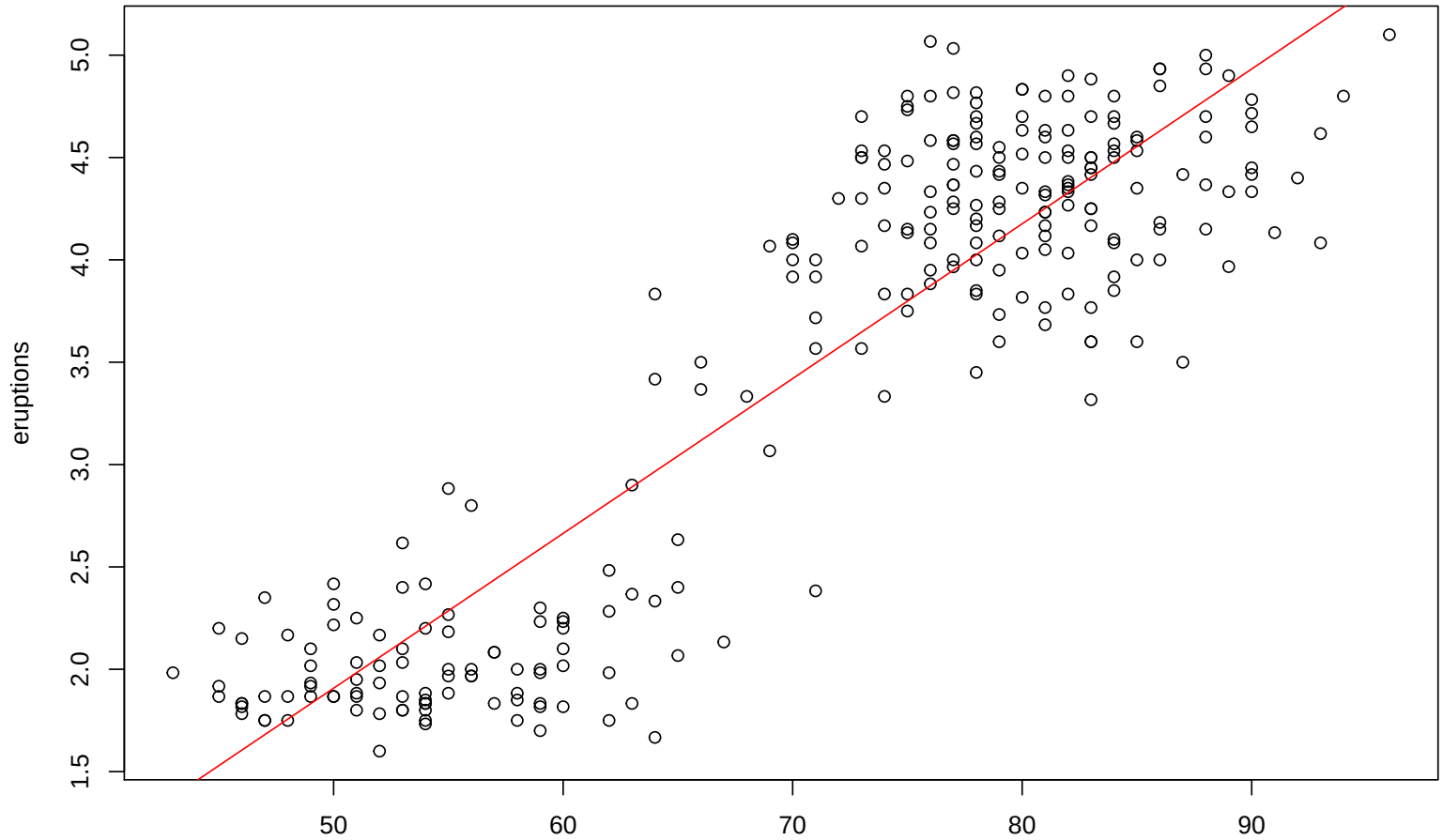
```
attach(faithful)
plot(waiting, eruptions)
```



# Simple linear regression model



```
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)
```

# Simple linear regression model



```
slrm.summary <- summary(slrm)
slrm.summary$coefficients
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  -1.8740    0.16014   -11.7  7.36e-26
## waiting       0.0756    0.00222    34.1 8.13e-100
```

# Simple linear regression model



## Estimation

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

- Using the function `coefficients` to extract the coefficients from the estimated model:

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

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

- Using the function `predict`:

```
predict(slrm, data.frame(waiting = 90))
```

```
##    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
```

## Remark

After using a dataset, detach it:

```
detach(faithful)
```

# Multiple linear regression



Donwload and then load the library openintro

```
install.packages("openintro") # donwload the library
library(openintro)
    # load the library in the current session
```

Load the dataset hsb2:

```
data(hsb2)
attach(hsb2)
```

The description of the variables is obtained typing

```
?hsb2
```

or can be found here.

# Multiple linear regression



## Example 1

Estimate the standardized math score by race:

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

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

# Multiple linear regression



## Example 1

## Model estimation

```
hsb2 <- read.csv("hsb2.csv")
hsb2$race <- factor(c("black",
                      "asian", "hisp", "white"))
attach(hsb2)
m1 <- lm(math ~ factor(race))
summary.m1 <- summary(m1)
```

# Multiple linear regression



## Example 1

### Model estimation: Extraction of the estimated coefficients

```
summary.m1$coefficients
```

```
##                      Estimate Std. Error t value Pr(>
## (Intercept)             54.40       1.32  41.082 2.93
## factor(race)black       -1.34       1.87  -0.716 4.75
## factor(race)hisp        -2.98       1.87  -1.591 1.13
## factor(race)white       -2.70       1.87  -1.442 1.51
```

# Multiple linear regression



## Example 1

## Use of vectorization

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

```
##   Estimate Std. Error    t value   Pr(>|t|)
##     -1.340      1.873     -0.716      0.475
```

# Multiple linear regression



## Example 2

## Factor interaction

Estimate the standardized math score by gender, race.

## R code

```
m2 <- lm(math ~ female*factor(race))
summary.m2 <- summary(m2)
```

# Multiple linear regression



## Example 2

## Estimated model

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

```
##                                Estimate Std. Error t val
## (Intercept)                       55.91       1.99  28.0
## female                            -2.69       2.66  -1.0
## factor(race)black                 -3.21       2.78  -1.1
## factor(race)hisp                  -2.39       2.78  -0.8
## factor(race)white                 -6.13       2.78  -2.2
## female:factor(race)black           3.37       3.75   0.8
## female:factor(race)hisp           -1.20       3.75  -0.3
## female:factor(race)white           6.25       3.75   1.6
```

# Multiple linear regression



## Example 3

Estimate the effect of the scores of reading skills, *read*; social studies, *socst*, y science, *science* on the score of mathematics, *math*.

## Model

$$math = \beta_0 + \beta_1 read + \beta_2 socst + \beta_3 science + u.$$

## R code

```
m3 <- lm(math ~ read + socst + science)
m3.estim <- summary(m3)
m3.estim$coefficients
```

# Multiple linear regression



## Example 3

## Estimated model

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    11.887     2.7937    4.25 3.24e-05
## read            0.313     0.0654    4.79 3.24e-06
## socst           0.154     0.0548    2.82 5.37e-03
## science         0.315     0.0599    5.25 3.96e-07
```

The model is significant, as the $F$-statistic of the model is equal to 74.576, with associated $p$-value $= 3.201 \times 10^{-32}$.

The model fit $R^2 = 0.533$.

# Multiple linear regression



## Remark

If we need to estimate models with interactions, or quadratic terms (continuous variables), we need to use `I(x1*x2)` in the model definition.
For example, to estimate the model

$$math = \beta_0 + \beta_1 read + \beta_2 socst + \beta_3 socst^2 + u,$$

we have to define the model es

```
m.inter <- lm(math ~ read + socst + I(socst^2))
```

# 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)`.

To show the necessary coding, we shall use the third model, `m3`.

To validate other models, we only need to suitably change the object `linmodel.Train`.

# In-sample and out-of-sample validation



## Example 3

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

# In-sample and out-of-sample validation



## Example 3

## Correlation matrix

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

```
##              actual predicted
## actual        1.000     0.829
## predicted     0.829     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] 0.915
```

# 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] 4.63
```

# 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, even 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] 0.0924
```

On the average, the error is 9.2%.

# In-sample and out-of-sample validation



## RMSE

$$RMSE = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(y_i - \hat{y}_i)^2}$$

- RMSE is the square root of the variance of the residuals.
- It can be interpreted as the standard deviation of the unexplained variance of the model.
- It has the same units as the response variable.

# In-sample and out-of-sample validation



## RMSE: R code

```
rmse <- sqrt(
  mean(((actual.predic$predicted -
                    actual.predic$actual))^2)
)
rmse
```

```
## [1] 5.66
```

# Diagnostics



The library `lmtest` permits to run tests to determine

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

The library `sandwich` permits to estimate HC, HAC[2] standard errors.

We shall run the diagnostics on the model `m3` (Example 3).

[2]HC: Heteroskedastic Consistent standard errors; HAC: Heteroskedastic and Autocorrelation Consistent errors.

# 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. For this reason, we also present the Jarque-Bera test.
- The normality of errors is less of a concern when the sample size is sufficiently large.

## Hyopthesis test

$$H_0 : \text{errors are normal}$$

$$H_1 : \text{errors are not normal}$$

$(\alpha = .05)$

# Diagnostics



## Normality: Shapiro-Wilk test

```
shapiro.test(m3$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  m3$residuals
## W = 1, p-value = 0.5
```

## Conclusion

There is no evidence of non-normality of errors.

# Diagnostics



## Jarque-Bera test

The Jarque-Bera test is available in the library `tseries`:

```
library(tseries)
jarque.bera.test(m3$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  m3$residuals
## X-squared = 3, df = 2, p-value = 0.2
```

## Conclusion

There is no evidence of non-normality of errors.

# Diagnostics



## QQ plot

The QQ plot compares the observed quantiles of the errors to the theoretical quantiles of the standard normal distribution.

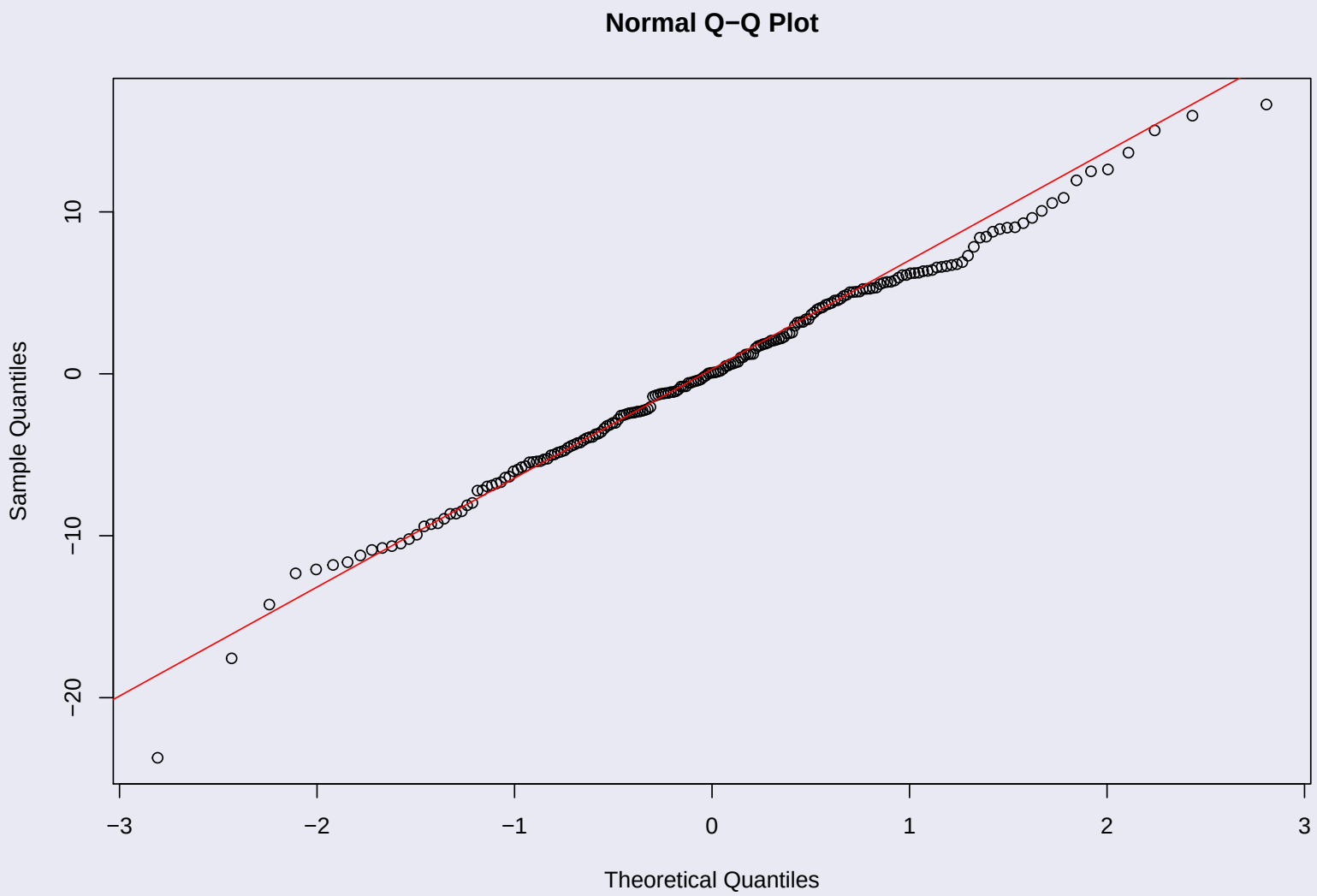
```
qqnorm(m3$residuals)
qqline(m3$residuals, col = "red")
```

We can see that the distribution of the errors follows pretty closely the standard normal distribution.

# Diagnostics



## QQ plot



# 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(m3)
```

# Diagnostics



## Heteroskedasticity

```
bptest(m3)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m3
## BP = 9, df = 3, p-value = 0.03
```

## Conclusion

- There is evidence of heteroskedasticity.
- If there is evidence of autocorrelation of errors, then neither OLS nor HC standard errors are correct.

# 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(m3, alternative = "two.sided")
```

```
##
##  Durbin-Watson test
##
## data:  m3
## DW = 2, p-value = 0.3
## alternative hypothesis: true autocorrelation is no
```

## Result

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

# Diagnostics



## Serial autocorrelation

```
bgtest(m3, order = 10)
```

```
##
##  Breusch-Godfrey test for serial correlation of or
##
## data:  m3
## LM test = 12, df = 10, p-value = 0.3
```

## Result

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

# Diagnostics



## Heteroskedasticity

We have detected heteroskedasticity, but not autocorrelation of the errors, so HC standard errors are appropriate.
HC standard errors are obtained using the function `coeftest()` passing the parameter `vcov. = vcovHC`[a]

```
library(sandwich)
coeftest(m3, vcov. = vcovHC)
```

All controls are still significant.

[a] There are several variants of HC standard errors.

# Diagnostics



## Heteroskedasticity

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.8865     2.4585    4.83  2.7e-06 *
## read          0.3134     0.0600    5.22  4.5e-07 *
## socst         0.1542     0.0572    2.70   0.0076 *
## science       0.3145     0.0567    5.54  9.5e-08 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '
```

**Remark** The estimated slopes are not affected by heteroskedasticity.

# Diagnostics



## RESET: Hypothesis test

$H_0$ : the model does not need nonlinear combination of controls

$H_1$ : the model needs nonlinear combination of controls

- Typically, cuadratic and cubic powers are considered.
- The test statistics may need heteroskedastic (HC; HAC) corrections.

## R code

```
resettest(m3, power = 2:3, vcov = vcovHC)
```

# Diagnostics



## RESET

```
##
##  RESET test
##
## data:  m3
## RESET = 1, df1 = 2, df2 = 196, p-value = 0.3
```

## Conclusion

There is no need of nonlinear combinations of regressors.

# 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(m3)
```

## Plots: documentation

It can be found here (check "Details").

# Diagnostics



## Plots: Leverage and influential observations

The concepts of leverage and influential observations can be used as a proxy for finding whether a given observation is an outlier.

- Leverage is a measure of how far the independent variables of an observation are from those of other observations.
- An influential observation is one whose deletion will affect greatly the estimate.

# Diagnostics



## Plots: Leverage and influential observations

- The leverage in a linear regression can be computed using the projection matrix (F. Hayashi, 2000, *Econometrics*, Princeton University Press, pp. 21-23).
- The Cook's distance defined in the next slide can be often used as a measure of observaions with high leverage; Cook's distance can also be used to detect highly influential observations.
- In R, the `residual-leverage` plot draws a red, dashed line identifying observations with Cook's distance greater than .5.
- In R, observations with leverage greater than 1 are omitted with a warning; check Residual-Leverage plot in "Details" here,

# Diagnostics

## Plots

## Cook's distance

The Cook's distance is a tool that is used to check whether a single observation has a large influence on the estimate of the linear regression:

$$D_i = \frac{\sum_{j=1}^{T} \left( \widehat{y}_j - \widehat{y}_{j(i)} \right)^2}{(k+1)TSS}$$

where $\widehat{y}_{j(i)}$ is the fitted response value when excluding $i$; $TSS$ is the total sum of square (mean square error): $TSS = (1/T) \sum_{i=1}^{T} (y_i - \widehat{y}_i)^2$.

```
detach(hsb2)
```

# Logit models

# Logit model



Estimate the probability of a positive diagnosis of heart disease, controlling for cholesterol levels and age:

$$\text{logit}(\textit{diagnostic}) = \beta_0 + \beta_1 \textit{chol} + \beta_2 \textit{age},$$

where

$$\text{logit}(\textit{diagnostic}) = \log\left(\frac{\textit{diagnostic}}{1 - \textit{diagnostic}}\right)$$

is the log-odds of a positive diagnosis.

Thus, *e.g.*, $\beta_1$ is the change of the log-odds of a positive diagnosis with respect to a unit increase of *chol*.

# Logit model



## Estimation

```
modelo.logit <- glm(diagnostic ~ chol + age,
                    data = cleve,
                    family = "binomial")
summary(modelo.logit)
```

# Logit model



## Estimation

```
##
## Call:
## glm(formula = diagnostic ~ chol + age, family = "b
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.593  -1.085  -0.815   1.158   1.729
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.32713    0.89145   -3.73  0.00019 *
## chol         0.00147    0.00237    0.62  0.53411
## age          0.05129    0.01405    3.65  0.00026 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '
##
```

# Logit model



## Goodness of fit

A measure of goodness of fit is a comparison of the estimated logit model to the null modelo

$$\text{logit}(\textit{diagnostic}) = \beta_0$$

The comparison is done by the hypothesis test

$$H_0 : \text{The null model fits better the data}$$
$$H_1 : \text{The logit model fits better the data}$$

($\alpha = .05$.)

# Logit model



## Goodness of fit

The test statistics is the difference of deviations, which has $\chi^2$ distribution, with `df. null - df.residual` degrees of freedom:

```
with(modelo.logit,
     pchisq(null.deviance - deviance,
                  df.null - df.residual,
                  lower.tail = FALSE))
```

```
## [1] 0.000322
```

## Conclusion

The logit model better fit the data.

# Logit model



Diagnosis of heart disease (*diagnostic* = 1), if the estimated value of *diagnostic* in the logit model is greater than `mean(diagnostic)`.

## Validation

```
probs.predicted <- predict(modelo.logit,
                           type = "response")
mu  <- mean(cleve$diagnostic)
probs.predicted2 <-
  ifelse(probs.predicted > mu, 1, 0)
table(cleve$diagnostic,
                    probs.predicted2)/nrow(cleve)
```

# Logit model



Off-diagonal terms are false positives and false negatives.

## Validation

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

There is a 21.5% of false positives and a 14.8% of false negatives.

# Logit model



Diagnosis of heart disease (*diagnostic* = 1), if the estimated value of *diagnostic* in the logit model is greater than .5.

## Validation

```
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,

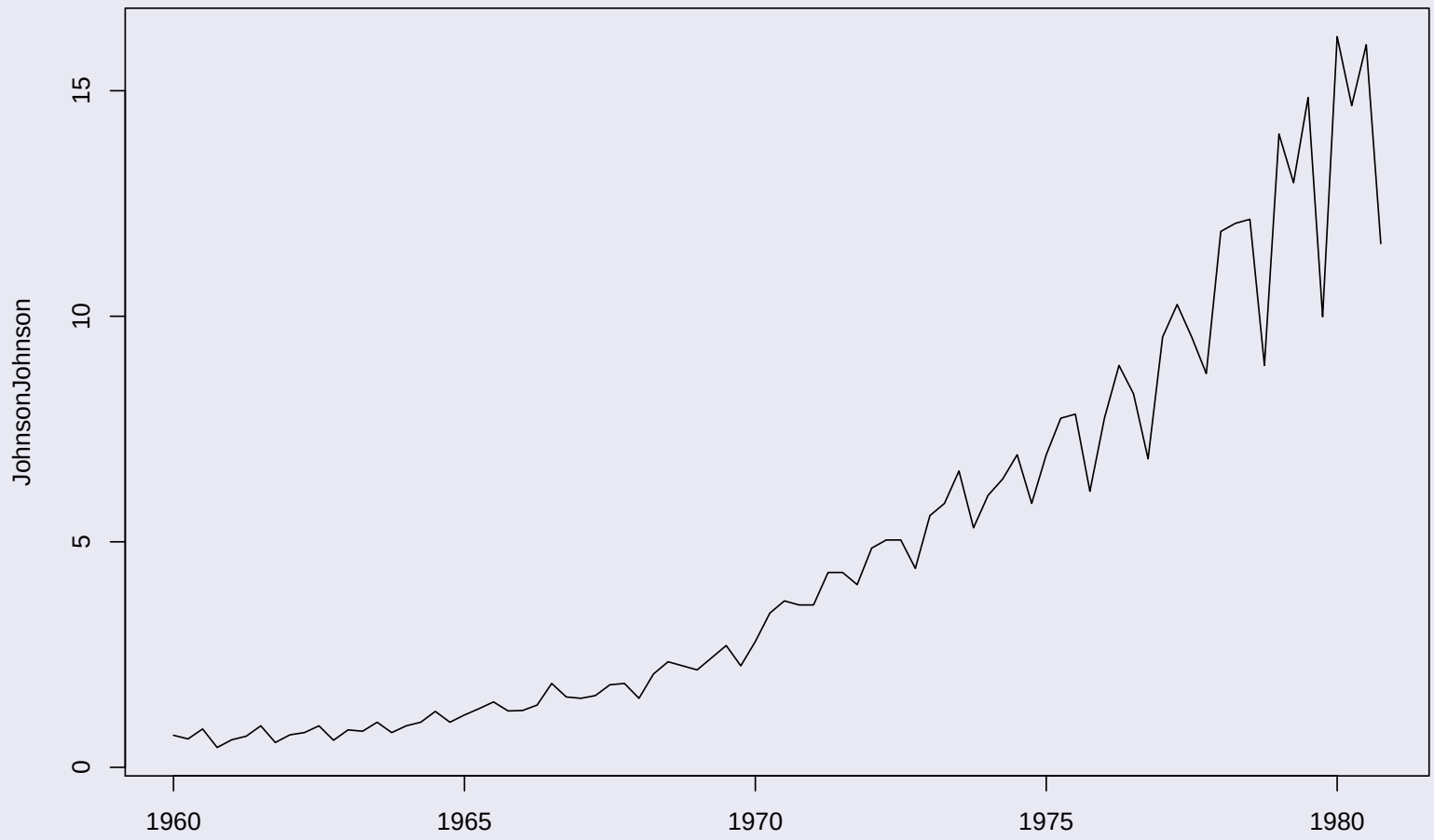
```
install.packages("forecast", "tseries")
```

and then load then in your session

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

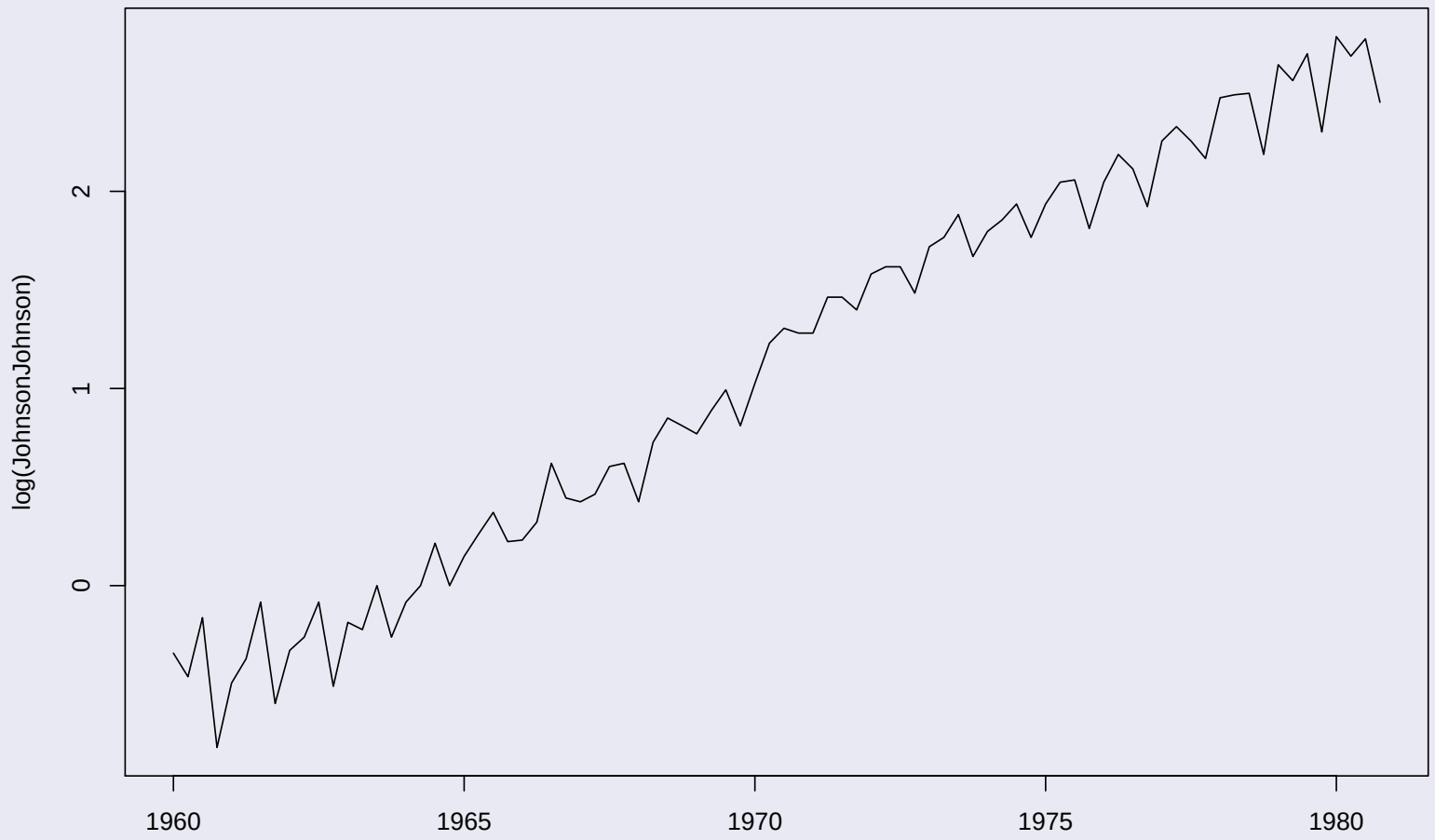
## Plots (Multiplicative model)

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



## 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:

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

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 $\alpha$ 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, $\ell_t$.

# 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

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

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

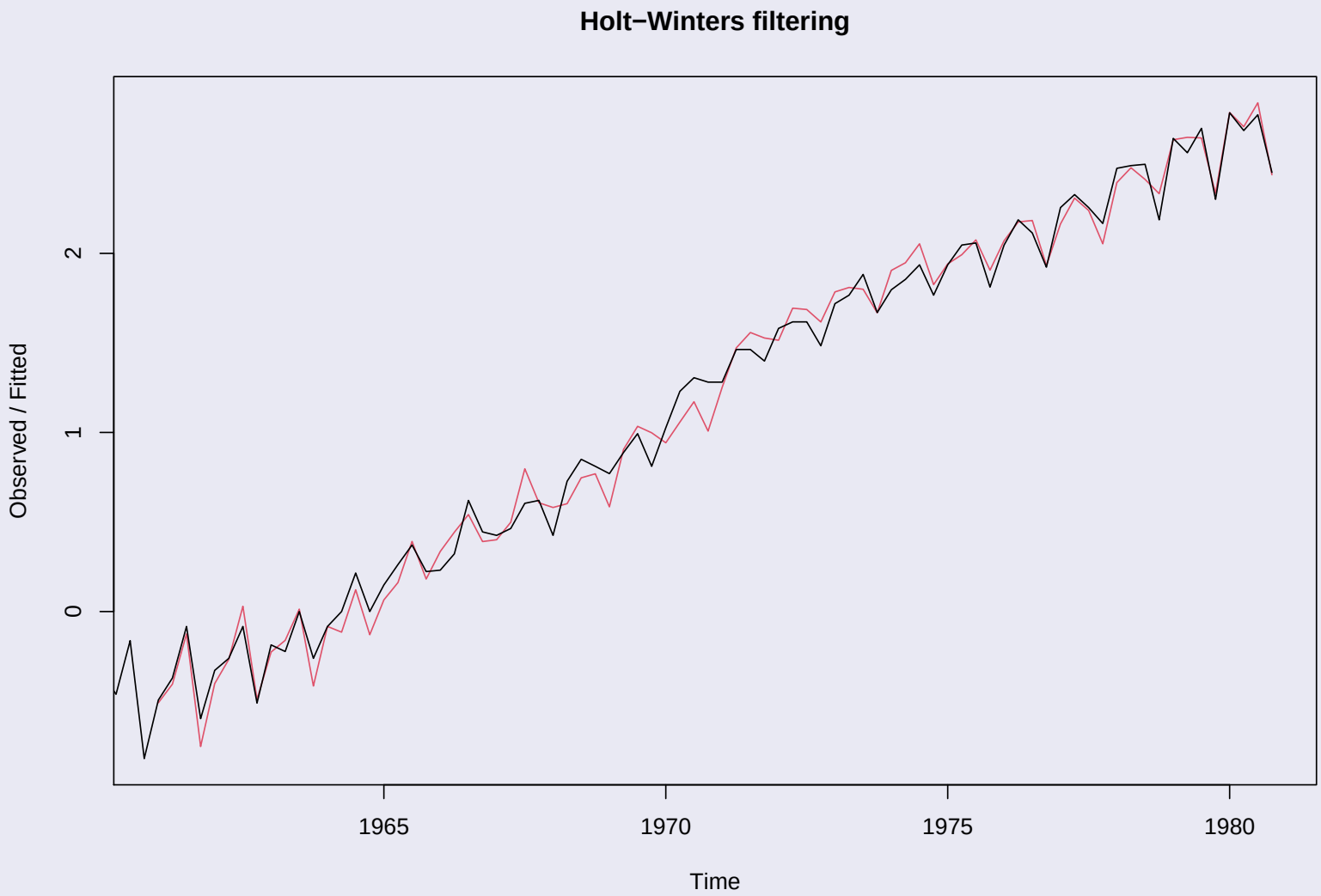
```
plot(logJJ.forecast)
```

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

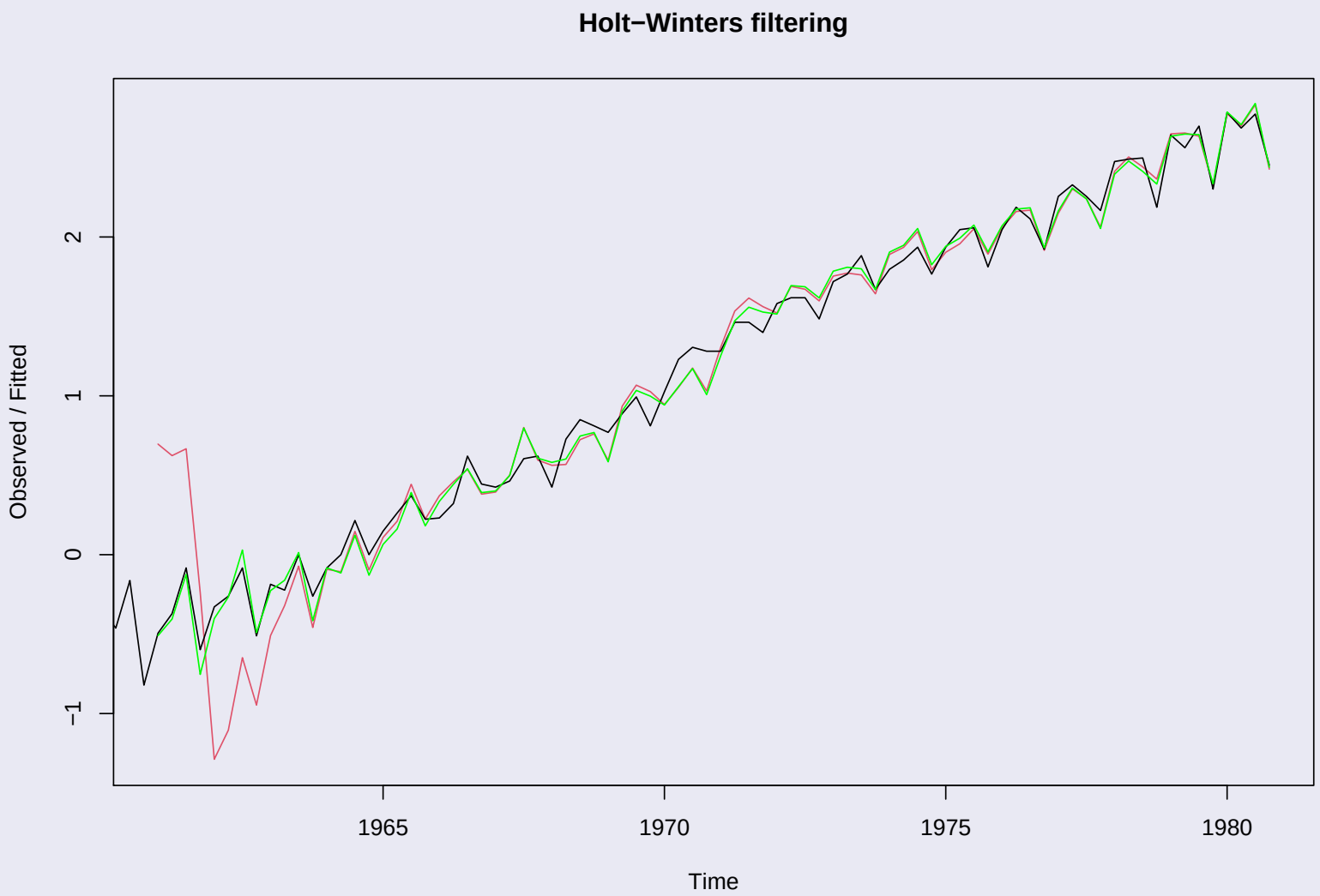
```
plot(logJJ.forecast2)
lines(logJJ.forecast$fitted[,1], col = "green")
```

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

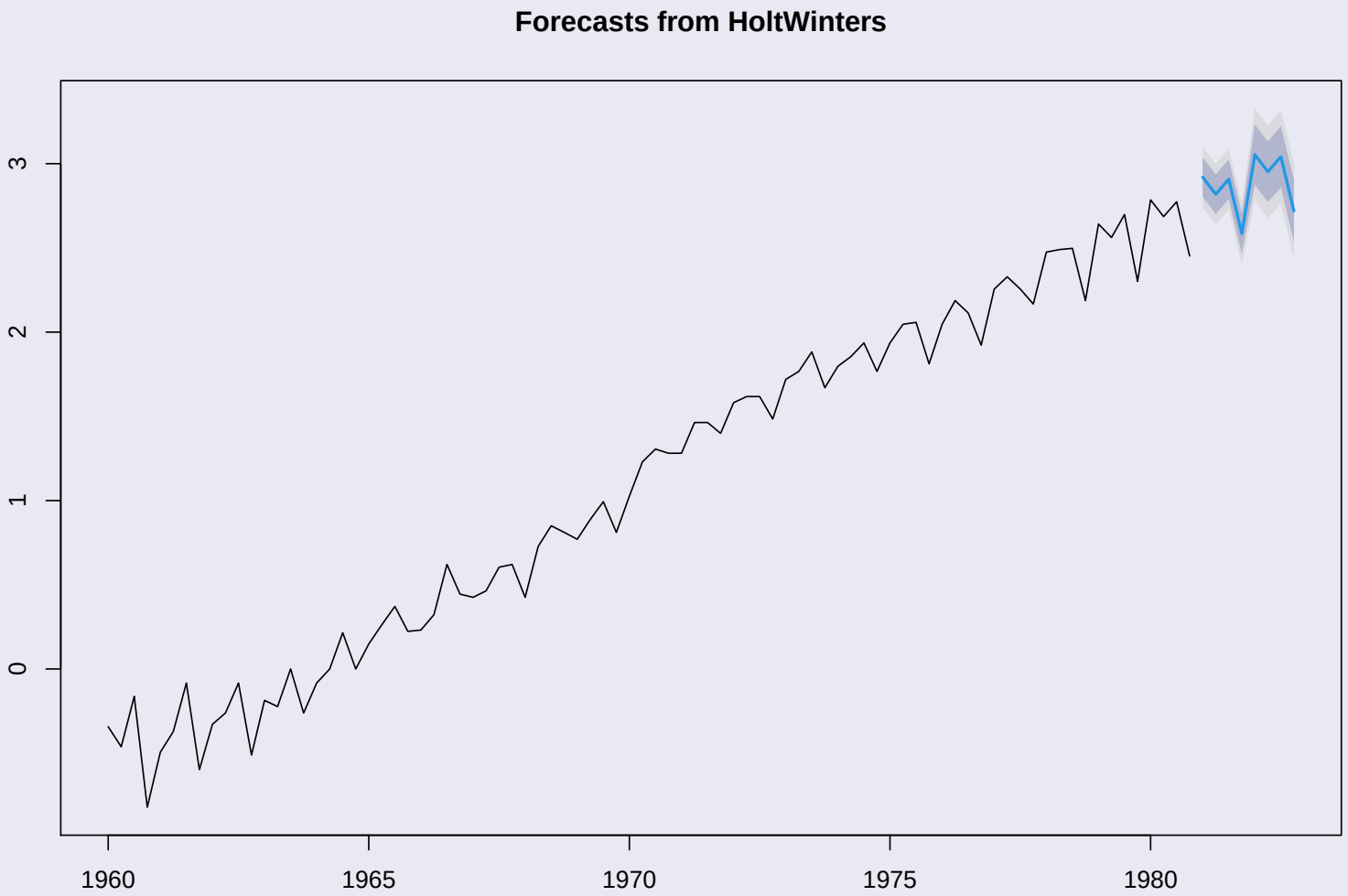
```
logJJ.forecast8095 <-
  forecast.HoltWinters(logJJ.forecast, h=8)
```

- 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



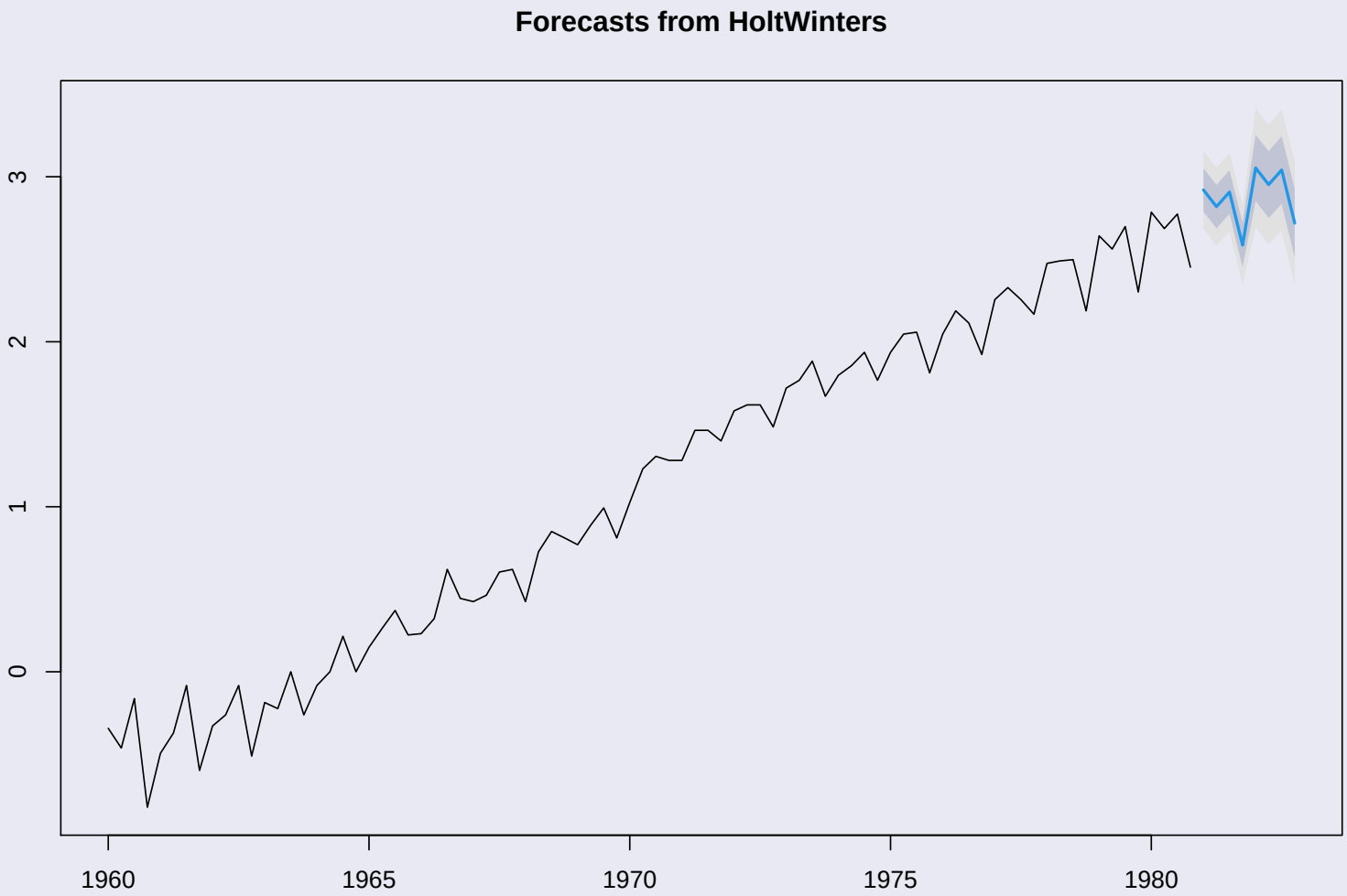
## Forecast: Plot

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

# Exponential smoothing



## 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



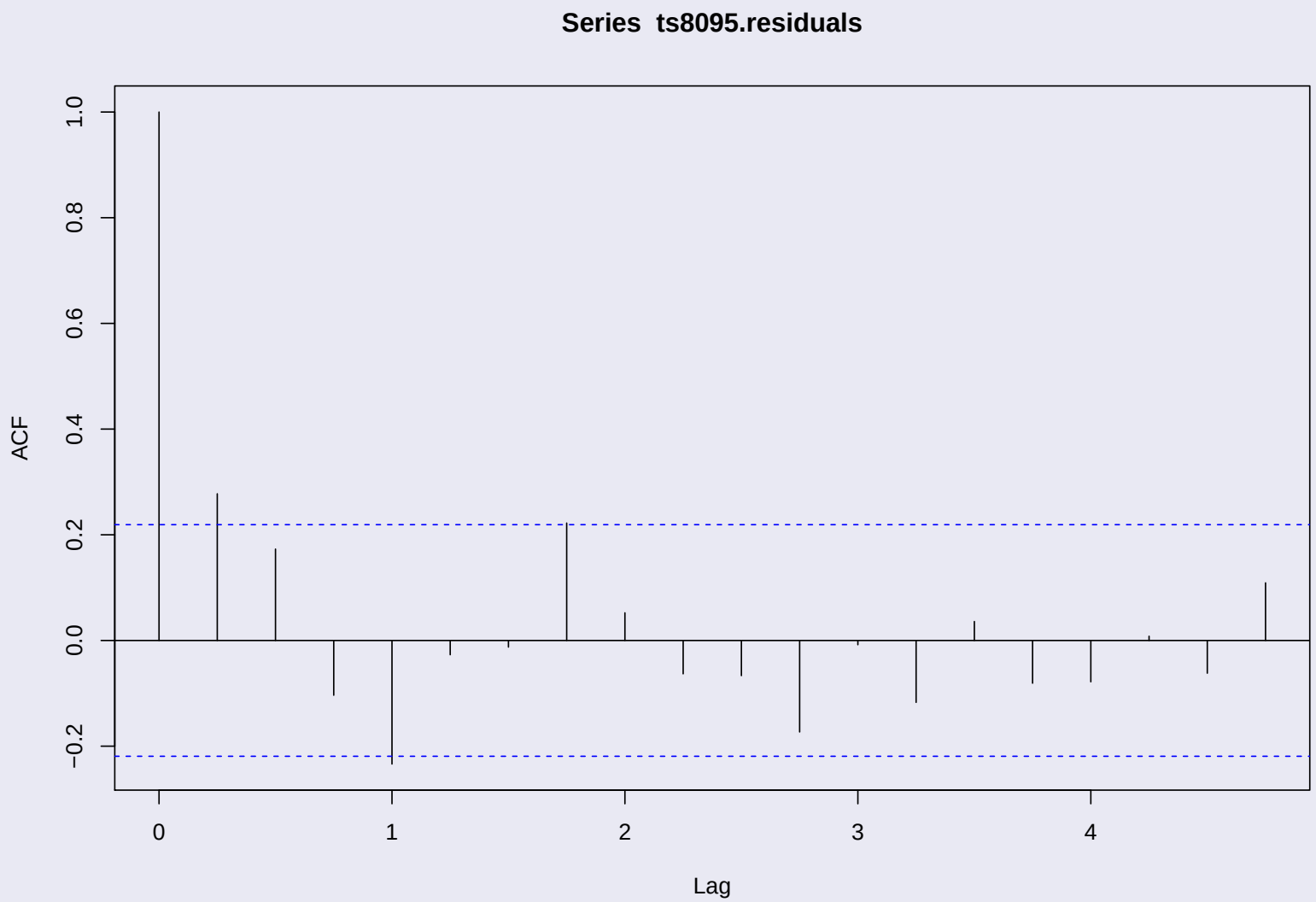
## Diagnostics: ACF

```
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`