Labs

FMor

May 26, 2023

Contents

0 Setup **7**
- 0.1 Colors . . . 7
- 0.2 Muliple plots . . . 7
- 0.3 Plot size . . . 8
- 0.4 Points charachters . . . 8
- 0.5 Line styles . . . 9
- 0.6 Legend . . . 10
- 0.7 Content summary . . . 11

1 Lab 01 - Introduction to R **12**
- 1.1 R basics . . . 12
  - 1.1.1 Categorical data . . . 15
  - 1.1.2 Lists . . . 15
  - 1.1.3 Dataframes . . . 16
  - 1.1.4 Data Import/Export . . . 17
- 1.2 Quantitative data . . . 19
  - 1.2.1 t-test for the mean . . . 21
  - 1.2.2 Linear regression . . . 23
  - 1.2.3 Conf and Pred intervals . . . 26
  - 1.2.4 Residuals . . . 28
- 1.3 Multivariate data . . . 29
- 1.4 Categorical data . . . 41
- 1.5 3d plots: bivariate gaussian . . . 46
- 1.6 Saving plots . . . 49

2 Lab 02 - First steps into statistics **49**
- 2.1 Probability density functions . . . 49

3 Lab 03 - PCA **51**
- 3.1 PCA of simulated normal data . . . 51
- 3.2 PCA on dataset tourists . . . 55
  - 3.2.1 PCA on original data . . . 57
  - 3.2.2 Loadings . . . 58
  - 3.2.3 Explained variance . . . 60
  - 3.2.4 PCA on std variables . . . 61
  - 3.2.5 Explained variance . . . 62
  - 3.2.6 Loadings . . . 63
  - 3.2.7 Scores . . . 64
  - 3.2.8 Scores in categories . . . 74
- 3.3 PCA on dataset food . . . 77
  - 3.3.1 PCA on std variables . . . 80
  - 3.3.2 Explained variance . . . 81
  - 3.3.3 Scores . . . 82
  - 3.3.4 Loadings . . . 84
  - 3.3.5 Projection on PCs space . . . 85
  - 3.3.6 Biplot . . . 87
- 3.4 R^p geom view of the PCs . . . 89

3.5 Exam exercise . . . 94

4 Lab 04 - About normality assumption **106**
4.1 On simulated data . . . 106
4.2 On dataset stiff . . . 109
4.2.1 Univariate normality . . . 110
4.2.2 Multivariate normality . . . 111
4.2.3 Recovering gaussianity . . . 112
4.2.3.1 Remove outliers . . . 112
4.2.3.2 Box-Cox transformations . . . 116
4.2.3.3 Univariate transformation . . . 117
4.2.3.4 Multivariate transformation . . . 118
4.2.4 Pvalue explanation . . . 123
4.2.5 Tranforming the data . . . 124
4.3 On a multivariate Gaussian . . . 127
4.3.1 Standard test for the mean . . . 128
4.3.2 Asymptotic test for the mean . . . 130
4.3.3 Ellipsoidal region . . . 132
4.3.4 Simultaneous and Bonf CIs . . . 133
4.4 On dataset stiff . . . 134
4.4.1 Test for the mean . . . 135
4.4.2 CR for the mean . . . 136
4.4.3 Simultaneous CIs . . . 137
4.4.4 Bonferroni CIs . . . 138

5 Lab 05 - Tests for multivariate gaussians **139**
5.1 Paired multivariate Gaussians . . . 140
5.1.1 Test for the mean on D . . . 142
5.1.2 CR for the mean of D . . . 143
5.1.3 SCIs for lin comb of components . . . 144
5.1.4 SCIs on all directions . . . 146
5.1.5 Bonferroni CIs . . . 147
5.1.6 Comparison SCI vs BSCI . . . 149
5.2 Test for repeated measures . . . 152
5.2.1 Qa . . . 153
5.2.2 Qb . . . 154
5.2.3 Different test . . . 156
5.3 Test for two indep Gaussians . . . 157
5.4 Exam exercise . . . 158
5.4.1 Qa . . . 165
5.4.2 Qb . . . 166
5.4.3 Qc . . . 166
5.5 Exam exercise . . . 167
5.5.1 Qa . . . 169
5.5.2 Qb . . . 171
5.5.3 Qc . . . 172

6 Lab 06 - Anova and Manova **173**
6.1 One-way ANOVA . . . 173
6.1.1 Model . . . 174
6.1.2 Verify the assumptions . . . 175
6.1.3 Actual fit . . . 176
6.1.4 CI for all the differences . . . 176
6.1.5 Multiple testing . . . 178
6.2 One-way MANOVA . . . 180
6.2.1 Model . . . 183
6.2.2 Verify the assumptions . . . 183
6.2.3 Actual fit . . . 185
6.3 Two-ways ANOVA . . . 187
6.3.1 Model . . . 189

6.3.1.1 With interaction . . . 189
6.3.1.2 Without interaction . . . 190
6.3.2 CIs for the differences . . . 191
6.4 Two-ways MANOVA . . . 192
6.4.1 Model . . . 195
6.4.1.0.1 With interaction . . . 196
6.4.1.1 Without interaction . . . 196
6.4.2 CIs for the differences . . . 197
6.5 Exam exercise . . . 199
6.5.1 Qa . . . 200
6.5.2 Qb . . . 201

7 Lab 07 - Classification (LDA, QDA, FDA, KNN, SVM) 202
7.1 On 2 classes, univariate . . . 202
7.1.1 Verify assumptions . . . 203
7.1.2 LDA . . . 206
7.1.3 KNN . . . 209
7.2 On 3 classes, bivariate . . . 211
7.2.1 Jittering . . . 212
7.2.2 MANOVA to check features difference . . . 213
7.2.3 LDA . . . 213
7.2.4 Computation of AER and APER . . . 214
7.2.4.1 APER . . . 214
7.2.4.2 AER . . . 216
7.2.4.3 With R . . . 216
7.2.5 Partition generated by LDA . . . 218
7.2.6 QDA . . . 219
7.2.7 Computation of AER and APER . . . 220
7.2.7.1 APER . . . 220
7.2.7.2 AER . . . 221
7.2.7.3 Partition generated by QDA . . . 223
7.2.8 KNN . . . 224
7.3 On 2 classes, bivariate . . . 226
7.3.1 Parameters definition . . . 228
7.3.2 QDA . . . 229
7.3.3 APER computation . . . 230
7.4 FDA . . . 231
7.4.1 Jittering . . . 231
7.4.2 Canonical reference system . . . 233
7.4.3 New obs classification . . . 234
7.4.4 Partition generated by FDA . . . 235
7.4.5 Comparison with LDA . . . 236
7.4.6 Projections on the canonical directions . . . 237
7.5 SVM . . . 239
7.5.1 Linear case . . . 239
7.5.2 Fit the model . . . 240
7.5.3 Changing the cost parameter . . . 242
7.5.4 Tune to find the best cost . . . 245
7.5.5 On iris dataset . . . 247
7.5.6 Non linear case . . . 250
7.5.7 Fit the model . . . 251
7.6 Trying polar transformation . . . 254
7.6.1 Example on multiclass classification . . . 256

8 Lab 08 - Clustering (HClust, KMeans, DBScan) 257
8.1 On iris dataset . . . 258
8.1.1 Dissimilarity matrix . . . 258
8.1.2 Hierarchical clustering . . . 260
8.1.3 Dendrogram plots . . . 263
8.1.4 Get the clusters by cutting the dendrogram . . . 266
8.1.5 Cophenetic matrix . . . 269
8.1.6 Copehenetic coefficients . . . 269
8.2 On simulated data . . . 270
8.2.1 Univariate case . . . 270
8.2.2 Balanced vs unbalanced trees . . . 271
8.2.3 Chaining effect, bad for single linkage . . . 272
8.2.4 Ellipsoidal clusters, bad for complete linkage . . . 275
8.3 On earthquake dataset . . . 279
8.3.1 Dissimiliraty matrix . . . 280
8.3.2 Dendrogram plots . . . 281
8.3.3 Ward linkage . . . 282
8.4 K-means method . . . 283
8.4.1 Algorithm simulation . . . 283
8.4.2 K-means on earthquake dataset . . . 285
8.4.3 How to choose k . . . 287
8.5 DBscan . . . 288
8.5.1 On simulated data . . . 288
8.5.2 Choose the parameters . . . 289
8.5.3 Choose with silhouette score . . . 291
8.6 On dataset moon . . . 296
8.7 Exam exercise . . . 302
8.7.1 Qa . . . 302
8.7.2 Qb . . . 305
8.7.3 Qc . . . 307
8.8 Exam exercise . . . 307
8.8.1 Qa . . . 308
8.8.2 Qb . . . 310
8.8.3 Qc . . . 310
8.9 Exam exercise . . . 311
8.9.1 Qa . . . 312
8.9.2 Qb . . . 313
8.9.3 Qc . . . 313
8.10 Exam exercise . . . 315
8.10.1 Qa . . . 316
8.10.2 Qc . . . 319
8.10.3 Qd . . . 319
8.11 Multidimensional scaling . . . 320

9 Lab 09 - Linear models 321
9.1 Multiple linear regression, on dataset cars . . . 321
9.1.1 Estimate of the parameters . . . 322
9.1.1.1 Lm summary explanation . . . 323
9.1.2 Inference on the parameters . . . 325
9.1.2.1 Confidence region . . . 326
9.1.2.2 Bonferroni CIs . . . 327
9.1.3 Test on lin comb of the beta_i . . . 328
9.1.4 CIs and PIs on a new obs . . . 328
9.1.5 Verify assumptions . . . 330
9.2 Importance of diagnostic of residuals . . . 332
9.3 Tuning the model, on dataset brain wheight . . . 334
9.3.1 Standard default model . . . 337
9.3.2 Remove outliers . . . 338
9.3.3 Transform data to ln . . . 340
9.3.4 CIs and PIs, ln data . . . 342

9.3.5 CIs and PIs, original data . . . 342
9.3.6 Model with clusters, on dataset earthquake . . . 343
9.3.7 Standard default model . . . 343
9.3.8 Model with a dummy variable . . . 345
9.3.9 Test for the difference in the two models . . . 347
9.3.10 Model reduction . . . 347
9.4 Bias/Variance trade-off, on simulated data . . . 349
9.5 Exam exercise . . . 354
9.5.1 Qa . . . 355
9.5.2 Qb . . . 357
9.5.3 Qc . . . 359
9.6 Exam exercise . . . 359
9.6.1 Qa . . . 361
9.6.2 Qb . . . 362
9.6.3 Qc . . . 364
9.6.4 Qd . . . 364
9.6.5 Qe . . . 365
9.6.6 Qf . . . 366
9.7 Linear regression with categorical data . . . 367
9.7.1 With one qualitative predictor . . . 370
9.7.2 With two qualitative predictor . . . 372

10 Lab 10 - Linear models **376**
10.1 Problem of collinearity . . . 376
10.1.1 Slt through PCA . . . 379
10.1.1.1 Model reduction . . . 381
10.1.2 Slt through ridge regression . . . 386
10.1.2.1 Grid of lambda . . . 387
10.1.2.2 Choose lambda with CV . . . 388
10.1.3 Slt through lasso/elasticnet regression . . . 390
10.2 Multiple linear regression . . . 392
10.2.1 PCA regression . . . 394
10.2.2 Ridge regression . . . 398
10.2.3 Lasso regression . . . 402
10.2.4 Comparison . . . 406
10.3 Model/Variable Selection . . . 409
10.3.1 Exhaustive search . . . 410
10.3.2 Forward and backward selection . . . 415
10.3.3 Comparison . . . 420
10.3.4 Ridge and lasso regression . . . 425
10.3.5 Comparison . . . 430
10.4 Logistic regression . . . 431
10.5 CARTs . . . 433
10.5.1 Carseats dataset . . . 433
10.5.2 Boston housing dataset . . . 438
10.5.3 Exam exercise . . . 440

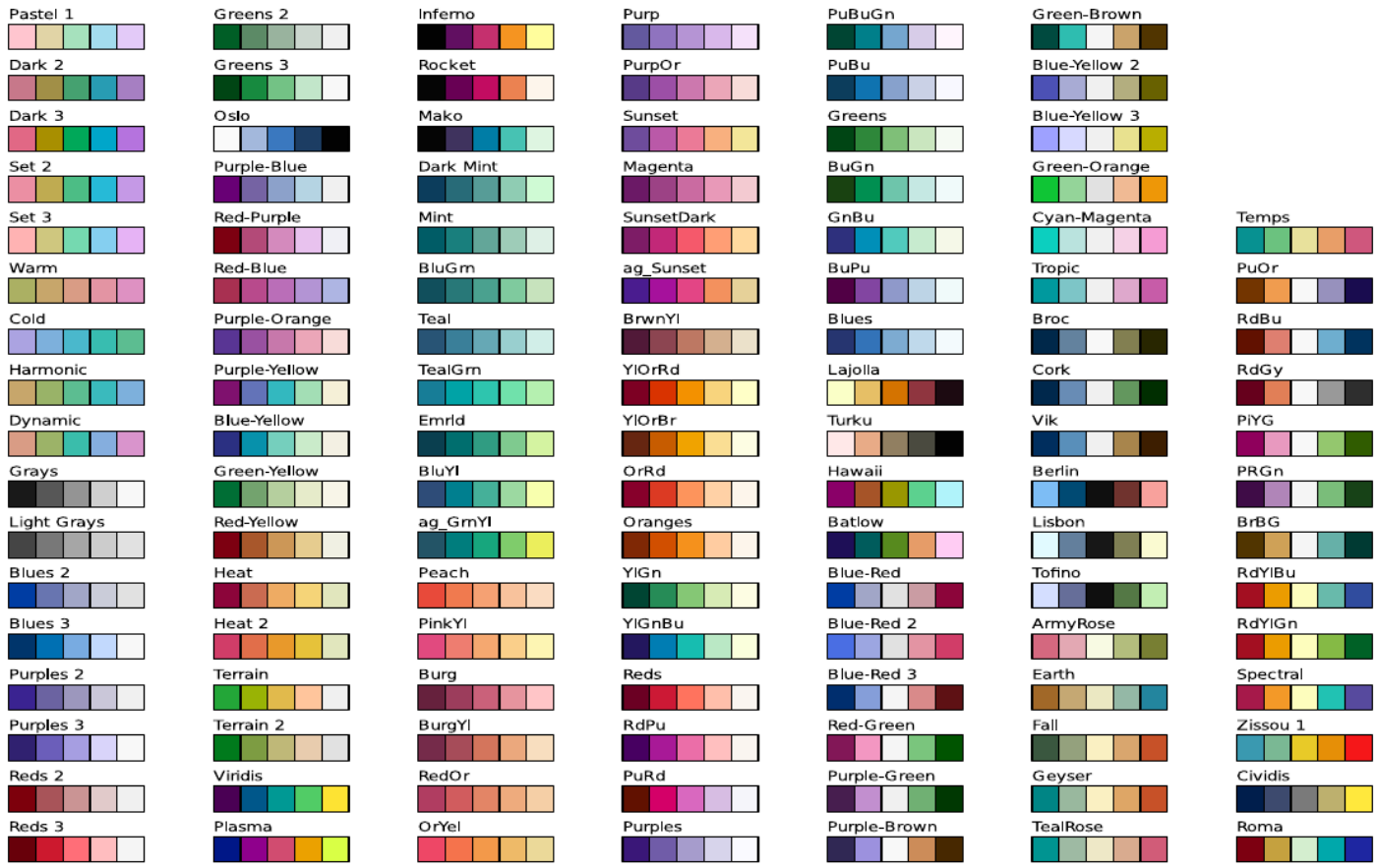
11 Lab 11 - LMMs **443**
11.1 From LM to LMM . . . 443
11.1.1 1. Linear Models with homoscedastic and independent errors . . . 445
11.1.2 2. Linear Models with heteroscedastic and independent errors . . . 451
11.1.2.1 2.1 VarIdent() . . . 451
11.1.2.2 2.2 VarPower() . . . 455
11.1.2.3 Residual analysis . . . 456
11.1.3 3. Linear Models with heteroscedastic and dependent errors . . . 460
11.1.3.1 3.1 CorCompSym() equal correlation . . . 462
11.1.3.2 3.2 AR(1) decreasing correlation . . . 465
11.1.3.3 3.3 general no contraints on the correlation . . . 467
11.1.3.4 Model-Fit Diagnostics . . . 468
11.2 LMM 1 - Age-Related Macular Degeneration . . . 472

11.2.0.0.1 PVRE . . . 476
11.2.0.0.2 Prediction . . . 477
11.2.0.0.3 Diagnostic plots . . . 479
11.2.1 2. Linear Models with random intercept + slope (q=1) . . . 481
11.2.1.1 2.1 general structure of D . . . 481
11.2.1.1.1 PVRE . . . 484
11.2.1.1.2 Comparing models . . . 486
11.2.1.1.3 Diagnostic plots . . . 486
11.2.1.2 2.2 diagonal D . . . 489
11.2.1.2.1 PVRE . . . 492
11.2.1.2.2 Comparing models . . . 493
11.2.2 With heteroschedastic rediduals . . . 493
11.3 LMM 2 - Students and Schools . . . 505
11.3.1 Linear model . . . 507
11.3.2 LMM with random intercept . . . 509
11.3.2.1 Fixed Effects and 95% CIs . . . 510
11.3.2.2 Variance components . . . 510
11.3.2.3 Random effects: b_0i . . . 511
11.3.2.4 All the regression lines . . . 514
11.3.2.5 Diagnostic plots . . . 515
11.3.2.6 Prediction . . . 517
11.3.2.7 Scenario Analysis . . . 518
11.3.3 LMM with random intercept and slope . . . 518
11.3.3.1 Variance components . . . 519
11.3.3.2 Estimates of fixed and random effects . . . 520
11.3.3.3 Visualization of random effects . . . 523
11.3.3.4 Lines Visualization . . . 524
11.3.3.5 Diagnostic plots . . . 524

0 Setup

0.1 Colors

```
# per tutte le palette:
x=hcl.pals()

fun_colori = function(len=2, seed=33){
    hcols_ = hcl.pals()
    col.ramp_ = hcl.colors(len,palette=hcols_[seed%%115+1])
    return(col.ramp_)
}
colori_fun = colorami = colora = fun_colori
```




```
# LABEL is a vector
col = hcl.colors(length(levels(LABEL)), palette = "viridis")
col = hcl.colors(length(unique(LABEL)), palette = "Cividis")

# And for the filled color legend
legend('topright', levels(factor(LABEL)), fill=col.ramp, bty='n', cex = 0.7)
# bty for not drawing a countour around the legend text
# cex to scale the dimension of the legend text
```

0.2 Muliple plots

```
# Create a multi-paneled plotting window. The par(mfrow) function is handy for
# creating a simple multi-paneled plot, while layout should be used for
# customized panel plots of varying sizes

par(mfrow = c(rows,cols))
#   mfrow - A vector of length 2, where the first argument specifies the number
#   of rows and the second the number of columns of plots
```

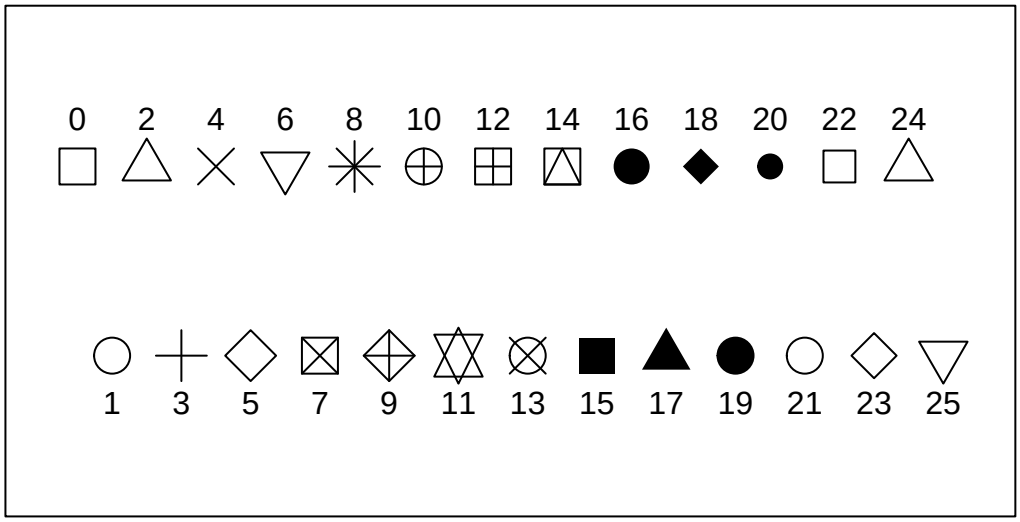
```r
layout(matrix, widths=rep(1, ncol(mat)), heights=rep(1, ncol(mat)))
#   matrix - A matrix describing the panel layout, where the numbers describe the order
#   in which to add the plots. A zero entry is interpreted as don't plot anything here
#   widths - The widths of the panel columns
#   heights - The heights of the panel rows

# Some examples
layout(matrix(c(1,2), ncol=1)) # to create a 1-col 2-row spaced plot
layout(matrix(c(1,2), ncol=2)) # to create a 2-col 1-row spaced plot

layout(matrix(c(1,1,2,3), nrow=2)) # to create a 2-rows 2-cols spaced plot
# first 2 panels will be for chart1 (11..), the third for chart2 (..2.) and the last for chart3 (...3)

layout(matrix(...),widths=c(3,1)) # to say col1 takes 3/4 of the plot space, col2 will take 1/4
layout(matrix(...),widths=c(2,1,2)) # col1 takes 2/5 of the plot space, col2 1/5 and col3 2/5
```

0.3 Plot size

```r
# all these alternatives should work
windows.options(width=5, height=6)
x11(width=4, height=6)
dev.new(width=5, height=4, unit="in")
dev.new(width = 1080, height = 720, unit = "px")

# when saving images
pdf("example.pdf", width=4, height=6)
```

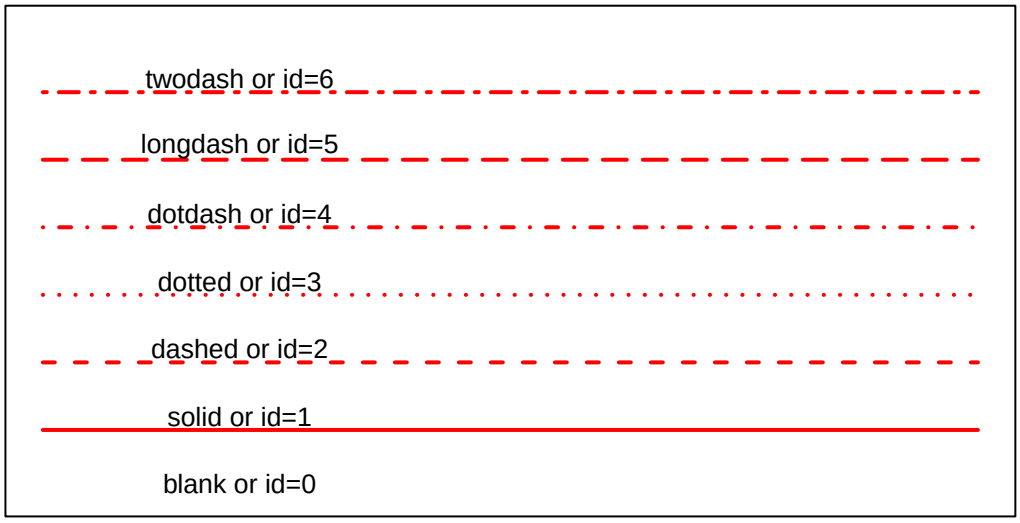
0.4 Points charachters

```r
x=0:25
y=ifelse(x%%2==0,+.1,-.1)
plot(x,y,xlim=c(-1,26), ylim=c(-.25,.25), pch=x,cex=2.5, main="PCH options",
     xlab = "",ylab="",axes=F)
box()
for( val in x){
    text(val,ifelse(val%%2==0,0.15,-0.15),val)
}
```



0.5 Line styles

```
len=20
x=1:len
y=rep(0,len)
#           0       1       2         3         4          5           6
styles = c("blank","solid","dashed", "dotted", "dotdash", "longdash", "twodash")

plot(x,y,type="l",ylim=c(1,8),col="red",lty="blank", #0
     axes=F, ylab="", xlab="",main="LTY styles")
box()
y=y+1
for(i in 1:7){
    lines(x,y,type="l",col="red",lwd=2, lty=styles[i]);
    y=y+1
}
for( h in 1:7){
    text(5,h+0.2,paste0(styles[h]," or id=",h-1),cex=0.8)
}
```



0.6 Legend

```
# https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/legend
# We think of LABEL as the vector containing the different cases for the plot.
# So something like
# - the months of the year (Jan, Feb, ecc, Nov, Dec)
# - the regions/states (Ita, Esp, Den, Nor, ecc)
# - the different treatments (Diet1, Diet2, ecc; Drug1, Drug2, ecc)
# - the different categories (Virginica, Versicolor, Setosa)

legend('topright', levels(LABEL), fill=rainbow(12), bty='n')
legend('topright', c(LABELS), fill=col.ramp, bty='n', cex = 0.7)
legend('topright', levels(unique(LABEL)), fill=col.ramp, bty='n', cex = 0.7)
legend('topright', levels(unique(LABEL)), fill=col.ramp, bty='n', cex = 0.7)
legend("topright", legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

legend('topleft',c('True','Estimated'),col=c('springgreen','red'),lty=c(1,1),lwd=2)
legend('topleft', c('Exact', 'Asymptotic'), col=c('blue', 'lightblue'),
legend(-10, 0.75, legend=c('LDA','knn'), lty=c(2,1), col=c('red','black'))

legend(-10, 0.03, legend=c(expression(paste('P(A)',f[A],'(x)')), col='blue'))
legend(-10, 0.9, legend=c('P(A|X=x)', 'P(B|X=x)'), fill=c('blue','red'), cex = 0.7)

# for letting next to the names a line type instead of filled squares
legend("topleft",c("lm","ridge"),lty=c(4,1),
       col=c("black","red"),lwd=2)
```

0.7 Content summary

```
# Lab 01
#   - R basics
#
# Lab 02
#    - Probability density/distribution functions, quantiles
#    - Random number generation
#    - QQplots
#
# Lab 03
#   - Principal Component Analysis
#
# Lab 04
#   - Multivariate normality tests
#   - Box-Cox transformations
#   - Tests and confidence regions for the mean of a multivariate Gaussian
#
# Lab 05
#   - Test for the mean of paired multivariate Gaussian observations
#   - Test for repeated measures
#   - Test for two independent Gaussian populations
#
# Lab 06
#   - One-way ANOVA and MANOVA
#   - Two-ways ANOVA and MANOVAs
#
# Lab 07
#   - Linear and Quadratic Discriminant Analysis
#   - Support Vector Machines
#
# Lab 08
#   - Hierarchical clustering
#   - K-means clustering
#   - DBSCAN
#   - Exercises
#   - Multidimensional Scaling
#
# Lab 09
#   - Linear models


### Cambiare i nomi delle variabili utilizzate nel codice
### Also by using ctrl+shift+alt+m which refactors name in current scope
```

1 Lab 01 - Introduction to R

1.1 R basics

```
# scalars
a = 1

# vectors
v = c(2, 3, 5, 4)
u = seq(2, 5, len=4)
u = seq(2, 5, by=1)
z = 2:5

# matrices
#! concatenate by row
W = rbind(c(11, 13, 15), c(12, 14, 16))
#! concatenate by col
W = cbind(c(11, 12), c(13, 14), c(15, 16))
W = matrix(data = c(11, 12, 13, 14, 15, 16), nrow = 2, ncol = 3, byrow = F)
W = matrix(c(11, 12, 13, 14, 15, 16), 2, 3)

# help(COMMANDNAME)
# e.g.
# help(matrix)

# Extraction of elements from a vector or a matrix
v
```

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

```
v[2]
```

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

```
v[c(2, 3)]
```

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

```
#! tutti tranne (meno) il terzo valore
v[-3]
```

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

```
#! tutti tranne (meno) quelli indicati
v[-c(1, 4)]
```

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

```
W
```

```
##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
```

```
W[2, 3]
```

```
## [1] 16
```

```
W[2, c(2, 3)]
```

```
## [1] 14 16
```

```
W[2, ]
```

```
## [1] 12 14 16
```

```r
W[, c(2, 3)]
```

```
##      [,1] [,2]
## [1,]   13   15
## [2,]   14   16
```

```r
# Remark: in R vectors "are not" matrices n*1 o 1*n:
# vectors have only one index, whereas matrices have
# two indices (for rows and columns)

v
```

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

```r
#! creates a matrix using c as rows
rbind(v)
```

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

```r
#! creates a matrix using c as cols
cbind(v)
```

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

```r
# Algebraic operations in R
# Remark: by default, operations are done component-wise

a = 1
b = 2
c = c(2, 3, 4)
d = c(10, 10, 10)
Z = matrix(c(1, 10, 3, 10, 5, 10), nrow = 2, ncol = 3, byrow = F)

# Sum and multiplication (component-wise).
# (this default is different from that of matlab!)

a + b # scalar + scalar
```

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

```r
c + d # vector + vector
```

```
## [1] 12 13 14
```

```r
a * b # scalar * scalar
```

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

```r
c * d # vector * vector (component-wise)
```

```
## [1] 20 30 40
```

```r
c + a # vector + scalar
```

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

```r
c^2 # attention: operations are always component-wise!
```

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

```r
exp(c)
```

```
## [1]  7.389056 20.085537 54.598150
```

```r
sum(c) # sums the components of c
```

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

```r
prod(c) # returns the product of the components of c
```

```
## [1] 24
```

```r
# Operations on matrices
V = t(W) # transpose of a matrix
Z + W # matrix + matrix (component-wise)
```

```
##      [,1] [,2] [,3]
## [1,]   12   16   20
## [2,]   22   24   26
```

```r
Z * W # matrix * matrix (component-wise)
```

```
##      [,1] [,2] [,3]
## [1,]   11   39   75
## [2,]  120  140  160
```

```r
V * W # matrix * matrix (component-wise) (error!)
```

```
## Error in V * W: array incompatibili
```

```r
V %*% W # Matrix multiplication
```

```
##      [,1] [,2] [,3]
## [1,]  265  311  357
## [2,]  311  365  419
## [3,]  357  419  481
```

```r
W %*% V
```

```
##      [,1] [,2]
## [1,]  515  554
## [2,]  554  596
```

```r
W + a # matrix + scalar
```

```
##      [,1] [,2] [,3]
## [1,]   12   14   16
## [2,]   13   15   17
```

```r
W + c # matrix + vector
```

```
##      [,1] [,2] [,3]
## [1,]   13   17   18
## [2,]   15   16   20
```

```r
W + 2:5
```

```
##      [,1] [,2] [,3]
## [1,]   13   17   17
## [2,]   15   19   19
```

```r
# Remark: R uses the "recycling", ie, it tries to make the
# terms dimensions compatible by recycling data if missing
W + 2:4 # recycling without warning!
```

```
##      [,1] [,2] [,3]
## [1,]   13   17   18
## [2,]   15   16   20
```

```r
# Inverse of a matrix (square and invertible)
A = matrix(c(11, 13, 12, 14), ncol=2, nrow=2, byrow=TRUE)
det(A)
```

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

```r
solve(A)
```

```
##      [,1] [,2]
## [1,]   -7  6.5
## [2,]    6 -5.5
```

```r
# Solution of a linear system Ax=b
b = c(1, 1)
solve(A, b)
```

```
## [1] -0.5  0.5
```

1.1.1 Categorical data

```r
# The command 'factor' converts the argument (vector of numbers or strings)
# in a vector of realizations of a categorical random variable, whose possible
# values are collected in 'Levels'

#! these are the data
district = c('MI', 'MI', 'VA', 'BG', 'LO', 'LO', 'CR', 'Alt', 'CR', 'MI',
             'Alt', 'CR', 'LO', 'VA', 'MI', 'Alt', 'LO', 'MI')
#! in this way we "group" our populations
district = factor(district, levels=c('MI', 'LO', 'BG', 'CR', 'VA', 'Alt'))
#! is useful to write by ourself the levels, so we can also count some categories
#! that we were interested in, even if they didnt show up in the data
#! use factor when dealing with categorical data
district
```

```
##  [1] MI  MI  VA  BG  LO  LO  CR  Alt CR  MI  Alt CR  LO  VA  MI  Alt LO  MI
## Levels: MI LO BG CR VA Alt
```

```r
resass = table(district) # table of absolute frequencies
resass
```

```
## district
##  MI  LO  BG  CR  VA Alt
##   5   4   1   3   2   3
```

```r
resrel = table(district) / length(district) # table of relative frequencies
resrel
```

```
## district
##         MI         LO         BG         CR         VA        Alt
## 0.27777778 0.22222222 0.05555556 0.16666667 0.11111111 0.16666667
```

1.1.2 Lists

```r
##### lists: objects made of objects (objects can be of different type)

#! are objects, like in classes of c++
exam = list (course = 'Applied Statistics',
            date = '27/09/2022',
            enrolled = 7,
            corrected = 6,
            student_id = as.character(c(45020, 45679, 46789,
                                        43126, 42345, 47568, 45674)),
            evaluation = c(30, 29, 30, NA, 25, 26, 27)
)
exam
```

```
## $course
## [1] "Applied Statistics"
##
## $date
## [1] "27/09/2022"
##
## $enrolled
## [1] 7
##
## $corrected
## [1] 6
##
## $student_id
## [1] "45020" "45679" "46789" "43126" "42345" "47568" "45674"
##
## $evaluation
## [1] 30 29 30 NA 25 26 27
```

```r
#! to access a certain field
exam$evaluation
```

```
## [1] 30 29 30 NA 25 26 27
```

```r
#! here we get the list (title and data)
exam[6]
```

```
## $evaluation
## [1] 30 29 30 NA 25 26 27
```

```r
#! here we get the values inside
exam[[6]]
```

```
## [1] 30 29 30 NA 25 26 27
```

1.1.3 Dataframes

```r
##### data.frame: objects made of vectors of the same lengths,
##### possibly of different types.
##### (Remark: they look like matrices by they aren't!)

#! data.frames are a subclass of list type

#! basically a matrix with column names, and possibly different types of data
#! rows are units, cols are for different variables
exam = data.frame(
    #! as character as we in the future wont use that, almost surely, as a regressor
  student_id = factor(as.character(c(45020, 45679, 46789, 43126, 42345, 47568,45674))),
  evaluation_W = c(30, 29, 30, NA, 25, 26, 17),
  evaluation_O = c(28, 30, 30, NA, 28, 27, NA),
  evaluation_P = c(30, 30, 30, 30, 28, 28, 28),
  outcome  = factor(c('Passed', 'Passed', 'Passed', 'To be repeated', 'Passed',
                      'Passed', 'To be repeated')))
exam
```

```
##   student_id evaluation_W evaluation_O evaluation_P        outcome
## 1      45020           30           28           30         Passed
## 2      45679           29           30           30         Passed
## 3      46789           30           30           30         Passed
## 4      43126           NA           NA           30 To be repeated
## 5      42345           25           28           28         Passed
## 6      47568           26           27           28         Passed
## 7      45674           17           NA           28 To be repeated
```

```r
exam$evaluation_W # a data.frame is a particular kind of list!
```

```
## [1] 30 29 30 NA 25 26 17
```

```r
exam[[2]]
```

```
## [1] 30 29 30 NA 25 26 17
```

```r
#! this to get the second row (row 2, all his cols)
exam[2, ]
```

```
##   student_id evaluation_W evaluation_O evaluation_P outcome
## 2      45679           29           30           30  Passed
```

```r
attach(exam)

#! with attach, changes done on the cols wont propagate to the dataframe itself
#! col_1+=42 wont change df$col_1
evaluation_W
```

```
## [1] 30 29 30 NA 25 26 17
```

```r
detach(exam)
```

1.1.4 Data Import/Export

```r
#! or also from Import Dataset, in Enviroment space
record = read.table('Lab01/record.txt', header=T, sep=' ')
record
```

```
##           m100  m200  m400 m800 m1500 m3000 Marathon
## argentin 11.61 22.94 54.50 2.15  4.43  9.79   178.52
## australi 11.20 22.35 51.08 1.98  4.13  9.08   152.37
## austria  11.43 23.09 50.62 1.99  4.22  9.34   159.37
## belgium  11.41 23.04 52.00 2.00  4.14  8.88   157.85
## bermuda  11.46 23.05 53.30 2.16  4.58  9.81   169.98
## brazil   11.31 23.17 52.80 2.10  4.49  9.77   168.75
## burma    12.14 24.47 55.00 2.18  4.45  9.51   191.02
## canada   11.00 22.25 50.06 2.00  4.06  8.81   149.45
## chile    12.00 24.52 54.90 2.05  4.23  9.37   171.38
## china    11.95 24.41 54.97 2.08  4.33  9.31   168.48
## columbia 11.60 24.00 53.26 2.11  4.35  9.46   165.42
## cookis   12.90 27.10 60.40 2.30  4.84 11.10   233.22
## costa    11.96 24.60 58.25 2.21  4.68 10.43   171.80
## czech    11.09 21.97 47.99 1.89  4.14  8.92   158.85
## denmark  11.42 23.52 53.60 2.03  4.18  8.71   151.75
## domrep   11.79 24.05 56.05 2.24  4.74  9.89   203.88
## finland  11.13 22.39 50.14 2.03  4.10  8.92   154.23
## france   11.15 22.59 51.73 2.00  4.14  8.98   155.27
## gdr      10.81 21.71 48.16 1.93  3.96  8.75   157.68
## frg      11.01 22.39 49.75 1.95  4.03  8.59   148.53
## gbni     11.00 22.13 50.46 1.98  4.03  8.62   149.72
## greece   11.79 24.08 54.93 2.07  4.35  9.87   182.20
## guatemal 11.84 24.54 56.09 2.28  4.86 10.54   215.08
## hungary  11.45 23.06 51.50 2.01  4.14  8.98   156.37
## india    11.95 24.28 53.60 2.10  4.32  9.98   188.03
## indonesi 11.85 24.24 55.34 2.22  4.61 10.02   201.28
## ireland  11.43 23.51 53.24 2.05  4.11  8.89   149.38
## israel   11.45 23.57 54.90 2.10  4.25  9.37   160.48
## italy    11.29 23.00 52.01 1.96  3.98  8.63   151.82
## japan    11.73 24.00 53.73 2.09  4.35  9.20   150.50
## kenya    11.73 23.88 52.70 2.00  4.15  9.20   181.05
## korea    11.96 24.49 55.70 2.15  4.42  9.62   164.65
## dprkorea 12.25 25.78 51.20 1.97  4.25  9.35   179.17
## luxembou 12.03 24.96 56.10 2.07  4.38  9.64   174.68
## malaysia 12.23 24.21 55.09 2.19  4.69 10.46   182.17
## mauritiu 11.76 25.08 58.10 2.27  4.79 10.90   261.13
## mexico   11.89 23.62 53.76 2.04  4.25  9.59   158.53
## netherla 11.25 22.81 52.38 1.99  4.06  9.01   152.48
## nz       11.55 23.13 51.60 2.02  4.18  8.76   145.48
## norway   11.58 23.31 53.12 2.03  4.01  8.53   145.48
## png      12.25 25.07 56.96 2.24  4.84 10.69   233.00
## philippi 11.76 23.54 54.60 2.19  4.60 10.16   200.37
## poland   11.13 22.21 49.29 1.95  3.99  8.97   160.82
## portugal 11.81 24.22 54.30 2.09  4.16  8.84   151.20
## rumania  11.44 23.46 51.20 1.92  3.96  8.53   165.45
## singapor 12.30 25.00 55.08 2.12  4.52  9.94   182.77
## spain    11.80 23.98 53.59 2.05  4.14  9.02   162.60
## sweden   11.16 22.82 51.79 2.02  4.12  8.84   154.48
```

```
## switzerl 11.45 23.31 53.11 2.02  4.07  8.77   153.42
## taipei   11.22 22.62 52.50 2.10  4.38  9.63   177.87
## thailand 11.75 24.46 55.80 2.20  4.72 10.28   168.45
## turkey   11.98 24.44 56.45 2.15  4.37  9.38   201.08
## usa      10.79 21.83 50.62 1.96  3.95  8.50   142.72
## ussr     11.06 22.19 49.19 1.89  3.87  8.45   151.22
## wsamoa   12.74 25.85 58.73 2.33  5.81 13.04   306.00
```

```r
str(record)
```

```
## 'data.frame':    55 obs. of  7 variables:
##  $ m100    : num  11.6 11.2 11.4 11.4 11.5 ...
##  $ m200    : num  22.9 22.4 23.1 23 23.1 ...
##  $ m400    : num  54.5 51.1 50.6 52 53.3 ...
##  $ m800    : num  2.15 1.98 1.99 2 2.16 2.1 2.18 2 2.05 2.08 ...
##  $ m1500   : num  4.43 4.13 4.22 4.14 4.58 4.49 4.45 4.06 4.23 4.33 ...
##  $ m3000   : num  9.79 9.08 9.34 8.88 9.81 9.77 9.51 8.81 9.37 9.31 ...
##  $ Marathon: num  179 152 159 158 170 ...
```

```r
head(record)
```

```
##           m100  m200  m400 m800 m1500 m3000 Marathon
## argentin 11.61 22.94 54.50 2.15  4.43  9.79   178.52
## australi 11.20 22.35 51.08 1.98  4.13  9.08   152.37
## austria  11.43 23.09 50.62 1.99  4.22  9.34   159.37
## belgium  11.41 23.04 52.00 2.00  4.14  8.88   157.85
## bermuda  11.46 23.05 53.30 2.16  4.58  9.81   169.98
## brazil   11.31 23.17 52.80 2.10  4.49  9.77   168.75
```

```r
dim(record)
```

```
## [1] 55  7
```

```r
dimnames(record)
```

```
## [[1]]
##  [1] "argentin" "australi" "austria"  "belgium"  "bermuda"  "brazil"
##  [7] "burma"    "canada"   "chile"    "china"    "columbia" "cookis"
## [13] "costa"    "czech"    "denmark"  "domrep"   "finland"  "france"
## [19] "gdr"      "frg"      "gbni"     "greece"   "guatemal" "hungary"
## [25] "india"    "indonesi" "ireland"  "israel"   "italy"    "japan"
## [31] "kenya"    "korea"    "dprkorea" "luxembou" "malaysia" "mauritiu"
## [37] "mexico"   "netherla" "nz"       "norway"   "png"      "philippi"
## [43] "poland"   "portugal" "rumania"  "singapor" "spain"    "sweden"
## [49] "switzerl" "taipei"   "thailand" "turkey"   "usa"      "ussr"
## [55] "wsamoa"
##
## [[2]]
## [1] "m100"     "m200"     "m400"     "m800"     "m1500"    "m3000"    "Marathon"
```

```r
# Transform times in seconds
record[, 4:7] = record[, 4:7] * 60
record
```

```
##           m100  m200  m400  m800 m1500 m3000 Marathon
## argentin 11.61 22.94 54.50 129.0 265.8 587.4  10711.2
## australi 11.20 22.35 51.08 118.8 247.8 544.8   9142.2
## austria  11.43 23.09 50.62 119.4 253.2 560.4   9562.2
## belgium  11.41 23.04 52.00 120.0 248.4 532.8   9471.0
## bermuda  11.46 23.05 53.30 129.6 274.8 588.6  10198.8
## brazil   11.31 23.17 52.80 126.0 269.4 586.2  10125.0
## burma    12.14 24.47 55.00 130.8 267.0 570.6  11461.2
## canada   11.00 22.25 50.06 120.0 243.6 528.6   8967.0
## chile    12.00 24.52 54.90 123.0 253.8 562.2  10282.8
## china    11.95 24.41 54.97 124.8 259.8 558.6  10108.8
## columbia 11.60 24.00 53.26 126.6 261.0 567.6   9925.2
## cookis   12.90 27.10 60.40 138.0 290.4 666.0  13993.2
## costa    11.96 24.60 58.25 132.6 280.8 625.8  10308.0
## czech    11.09 21.97 47.99 113.4 248.4 535.2   9531.0
## denmark  11.42 23.52 53.60 121.8 250.8 522.6   9105.0
## domrep   11.79 24.05 56.05 134.4 284.4 593.4  12232.8
```

```
## finland  11.13 22.39 50.14 121.8 246.0 535.2   9253.8
## france   11.15 22.59 51.73 120.0 248.4 538.8   9316.2
## gdr      10.81 21.71 48.16 115.8 237.6 525.0   9460.8
## frg      11.01 22.39 49.75 117.0 241.8 515.4   8911.8
## gbni     11.00 22.13 50.46 118.8 241.8 517.2   8983.2
## greece   11.79 24.08 54.93 124.2 261.0 592.2  10932.0
## guatemal 11.84 24.54 56.09 136.8 291.6 632.4  12904.8
## hungary  11.45 23.06 51.50 120.6 248.4 538.8   9382.2
## india    11.95 24.28 53.60 126.0 259.2 598.8  11281.8
## indonesi 11.85 24.24 55.34 133.2 276.6 601.2  12076.8
## ireland  11.43 23.51 53.24 123.0 246.6 533.4   8962.8
## israel   11.45 23.57 54.90 126.0 255.0 562.2   9628.8
## italy    11.29 23.00 52.01 117.6 238.8 517.8   9109.2
## japan    11.73 24.00 53.73 125.4 261.0 552.0   9030.0
## kenya    11.73 23.88 52.70 120.0 249.0 552.0  10863.0
## korea    11.96 24.49 55.70 129.0 265.2 577.2   9879.0
## dprkorea 12.25 25.78 51.20 118.2 255.0 561.0  10750.2
## luxembou 12.03 24.96 56.10 124.2 262.8 578.4  10480.8
## malaysia 12.23 24.21 55.09 131.4 281.4 627.6  10930.2
## mauritiu 11.76 25.08 58.10 136.2 287.4 654.0  15667.8
## mexico   11.89 23.62 53.76 122.4 255.0 575.4   9511.8
## netherla 11.25 22.81 52.38 119.4 243.6 540.6   9148.8
## nz       11.55 23.13 51.60 121.2 250.8 525.6   8728.8
## norway   11.58 23.31 53.12 121.8 240.6 511.8   8728.8
## png      12.25 25.07 56.96 134.4 290.4 641.4  13980.0
## philippi 11.76 23.54 54.60 131.4 276.0 609.6  12022.2
## poland   11.13 22.21 49.29 117.0 239.4 538.2   9649.2
## portugal 11.81 24.22 54.30 125.4 249.6 530.4   9072.0
## rumania  11.44 23.46 51.20 115.2 237.6 511.8   9927.0
## singapor 12.30 25.00 55.08 127.2 271.2 596.4  10966.2
## spain    11.80 23.98 53.59 123.0 248.4 541.2   9756.0
## sweden   11.16 22.82 51.79 121.2 247.2 530.4   9268.8
## switzerl 11.45 23.31 53.11 121.2 244.2 526.2   9205.2
## taipei   11.22 22.62 52.50 126.0 262.8 577.8  10672.2
## thailand 11.75 24.46 55.80 132.0 283.2 616.8  10107.0
## turkey   11.98 24.44 56.45 129.0 262.2 562.8  12064.8
## usa      10.79 21.83 50.62 117.6 237.0 510.0   8563.2
## ussr     11.06 22.19 49.19 113.4 232.2 507.0   9073.2
## wsamoa   12.74 25.85 58.73 139.8 348.6 782.4  18360.0
```

```r
# to save a data frame (or a matrix)
write.table(record, file = 'Lab01/record_mod.txt')
# Remark. The file containing 'record_mod.txt' will be saved in the working directory

# to save several objects in the workspace
W = matrix(data = c(11, 12, 13, 14, 15, 16), nrow = 2, ncol = 3, byrow = F)
V = t(W)
a = 1

#! for example to save heavily-computed stuff
#! and is more efficient
save(W, V, a, file = 'variousobjects.RData')

# to save the entire workspace: save.image('FILENAME.RData')
save.image("myworkspace.RData")

# this command remove all the variable of the workspace
rm(list=ls())

# to load a workspace (ie, .RData)
load("variousobjects.RData")
```

1.2 Quantitative data

```r
##### Example: analysis of quantitative data (with plots)

record = read.table('Lab01/record_mod.txt', header=T)
record
```

```
##           m100  m200  m400  m800 m1500 m3000 Marathon
## argentin 11.61 22.94 54.50 129.0 265.8 587.4  10711.2
## australi 11.20 22.35 51.08 118.8 247.8 544.8   9142.2
```

```
## austria  11.43 23.09 50.62 119.4 253.2 560.4   9562.2
## belgium  11.41 23.04 52.00 120.0 248.4 532.8   9471.0
## bermuda  11.46 23.05 53.30 129.6 274.8 588.6  10198.8
## brazil   11.31 23.17 52.80 126.0 269.4 586.2  10125.0
## burma    12.14 24.47 55.00 130.8 267.0 570.6  11461.2
## canada   11.00 22.25 50.06 120.0 243.6 528.6   8967.0
## chile    12.00 24.52 54.90 123.0 253.8 562.2  10282.8
## china    11.95 24.41 54.97 124.8 259.8 558.6  10108.8
## columbia 11.60 24.00 53.26 126.6 261.0 567.6   9925.2
## cookis   12.90 27.10 60.40 138.0 290.4 666.0  13993.2
## costa    11.96 24.60 58.25 132.6 280.8 625.8  10308.0
## czech    11.09 21.97 47.99 113.4 248.4 535.2   9531.0
## denmark  11.42 23.52 53.60 121.8 250.8 522.6   9105.0
## domrep   11.79 24.05 56.05 134.4 284.4 593.4  12232.8
## finland  11.13 22.39 50.14 121.8 246.0 535.2   9253.8
## france   11.15 22.59 51.73 120.0 248.4 538.8   9316.2
## gdr      10.81 21.71 48.16 115.8 237.6 525.0   9460.8
## frg      11.01 22.39 49.75 117.0 241.8 515.4   8911.8
## gbni     11.00 22.13 50.46 118.8 241.8 517.2   8983.2
## greece   11.79 24.08 54.93 124.2 261.0 592.2  10932.0
## guatemal 11.84 24.54 56.09 136.8 291.6 632.4  12904.8
## hungary  11.45 23.06 51.50 120.6 248.4 538.8   9382.2
## india    11.95 24.28 53.60 126.0 259.2 598.8  11281.8
## indonesi 11.85 24.24 55.34 133.2 276.6 601.2  12076.8
## ireland  11.43 23.51 53.24 123.0 246.6 533.4   8962.8
## israel   11.45 23.57 54.90 126.0 255.0 562.2   9628.8
## italy    11.29 23.00 52.01 117.6 238.8 517.8   9109.2
## japan    11.73 24.00 53.73 125.4 261.0 552.0   9030.0
## kenya    11.73 23.88 52.70 120.0 249.0 552.0  10863.0
## korea    11.96 24.49 55.70 129.0 265.2 577.2   9879.0
## dprkorea 12.25 25.78 51.20 118.2 255.0 561.0  10750.2
## luxembou 12.03 24.96 56.10 124.2 262.8 578.4  10480.8
## malaysia 12.23 24.21 55.09 131.4 281.4 627.6  10930.2
## mauritiu 11.76 25.08 58.10 136.2 287.4 654.0  15667.8
## mexico   11.89 23.62 53.76 122.4 255.0 575.4   9511.8
## netherla 11.25 22.81 52.38 119.4 243.6 540.6   9148.8
## nz       11.55 23.13 51.60 121.2 250.8 525.6   8728.8
## norway   11.58 23.31 53.12 121.8 240.6 511.8   8728.8
## png      12.25 25.07 56.96 134.4 290.4 641.4  13980.0
## philippi 11.76 23.54 54.60 131.4 276.0 609.6  12022.2
## poland   11.13 22.21 49.29 117.0 239.4 538.2   9649.2
## portugal 11.81 24.22 54.30 125.4 249.6 530.4   9072.0
## rumania  11.44 23.46 51.20 115.2 237.6 511.8   9927.0
## singapor 12.30 25.00 55.08 127.2 271.2 596.4  10966.2
## spain    11.80 23.98 53.59 123.0 248.4 541.2   9756.0
## sweden   11.16 22.82 51.79 121.2 247.2 530.4   9268.8
## switzerl 11.45 23.31 53.11 121.2 244.2 526.2   9205.2
## taipei   11.22 22.62 52.50 126.0 262.8 577.8  10672.2
## thailand 11.75 24.46 55.80 132.0 283.2 616.8  10107.0
## turkey   11.98 24.44 56.45 129.0 262.2 562.8  12064.8
## usa      10.79 21.83 50.62 117.6 237.0 510.0   8563.2
## ussr     11.06 22.19 49.19 113.4 232.2 507.0   9073.2
## wsamoa   12.74 25.85 58.73 139.8 348.6 782.4  18360.0
```

```r
# some synthetic indices
colMeans(record)
```

```
##        m100        m200        m400        m800       m1500       m3000
##    11.61855    23.64164    53.40582   124.58182   259.52727   566.85818
##    Marathon
## 10395.19636
```

```r
#! sapply apply a certain function column-wise on the df
sapply(record, mean)
```

```
##        m100        m200        m400        m800       m1500       m3000
##    11.61855    23.64164    53.40582   124.58182   259.52727   566.85818
##    Marathon
## 10395.19636
```

```r
sapply(record, sd)
```

```
##         m100         m200         m400         m800        m1500        m3000
##    0.4522103    1.1110602    2.6783367    6.4934466   19.9455319   49.4601474
##     Marathon
## 1825.7726951
```

```
sapply(record, var)
```

```
##         m100         m200         m400         m800        m1500        m3000
## 2.044941e-01 1.234455e+00 7.173488e+00 4.216485e+01 3.978242e+02 2.446306e+03
##     Marathon
## 3.333446e+06
```

```
cov(record)
```

```
##                 m100         m200        m400        m800        m1500
## m100       0.2044941    0.4787135    1.010955    2.136788     6.569596
## m200       0.4787135    1.2344547    2.550142    5.223808    15.476232
## m400       1.0109549    2.5501422    7.173488   15.624737    42.087172
## m800       2.1367879    5.2238081   15.624737   42.164848   116.772727
## m1500      6.5695960   15.4762323   42.087172  116.772727   397.824242
## m3000     16.5891232   39.0096808  103.014285  277.348485   956.093939
## Marathon 566.6616242 1390.7175616 3449.547725 9238.943636 31970.827879
##                 m3000     Marathon
## m100        16.58912     566.6616
## m200        39.00968    1390.7176
## m400       103.01428    3449.5477
## m800       277.34848    9238.9436
## m1500      956.09394   31970.8279
## m3000     2446.30618   81258.0017
## Marathon 81258.00170 3333445.9341
```

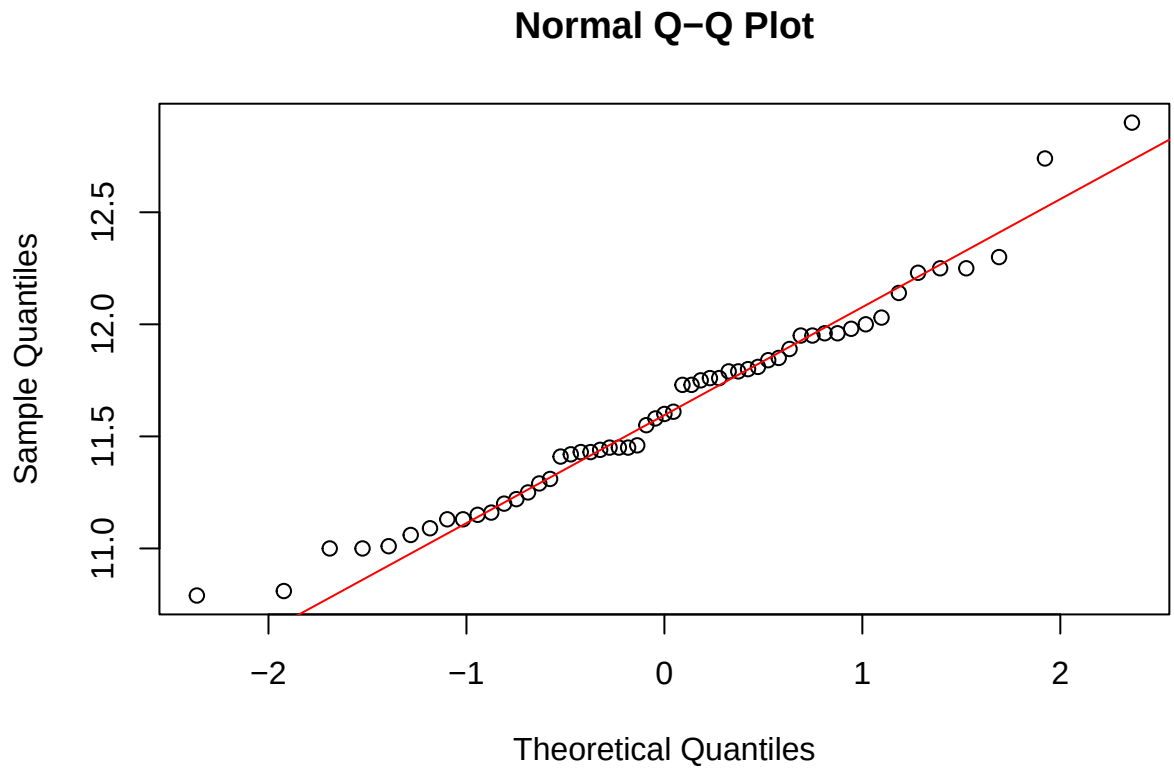
```
cor(record)
```

```
##               m100      m200      m400      m800     m1500     m3000  Marathon
## m100     1.0000000 0.9527911 0.8346918 0.7276888 0.7283709 0.7416988 0.6863358
## m200     0.9527911 1.0000000 0.8569621 0.7240597 0.6983643 0.7098710 0.6855745
## m400     0.8346918 0.8569621 1.0000000 0.8984052 0.7878417 0.7776369 0.7054241
## m800     0.7276888 0.7240597 0.8984052 1.0000000 0.9016138 0.8635652 0.7792922
## m1500    0.7283709 0.6983643 0.7878417 0.9016138 1.0000000 0.9691690 0.8779334
## m3000    0.7416988 0.7098710 0.7776369 0.8635652 0.9691690 1.0000000 0.8998374
## Marathon 0.6863358 0.6855745 0.7054241 0.7792922 0.8779334 0.8998374 1.0000000
```

```
##### Descriptive/inferential analysis on the variable m100 ('very basic'!)
attach(record)
```

1.2.1 t-test for the mean

```
### t-test for the mean value of the quantity
### H0: mean == 11.5 vs H1: mean != 11.5

#! Question: can the sample mean of this dataset confirm or disprove that the
#! mean time of running 100m is 11.5 or not? so comparing "what we observed with
#! an hypothesis about a general theory on the "100 meters time" ("global") data"

# Recall: qqplot to verify (qualitatively) the Gaussian assumption on the
# distribution generating sample
qqnorm(m100) # quantile-quantile plot
qqline(m100, col='red') # theoretical line
```




```
# Recall: Shapiro-Wilk test to verify (quantitatively) the Gaussian assumption on the
# distribution generating sample
shapiro.test(m100)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  m100
## W = 0.97326, p-value = 0.2569
```

```
#! shapiro works also on multivariate data
#! high pvalue -> we accept H0

alpha = .05
mean.H0 = 11.5

# automatically
esito=t.test(m100, mu = mean.H0, alternative = 'two.sided', conf.level = 1-alpha)
#! we can access his fields with esito$field

# manually
sample.mean = mean(m100)
sample.sd = sd(m100)
n = length(m100)
tstat = (sample.mean - mean.H0) / (sample.sd / sqrt(n))
cfr.t = qt(1 - alpha/2, n-1)
abs(tstat) < cfr.t
```

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

```
#! we accept H0, the tstat is inside the acceptance region

pval  = ifelse(tstat >= 0, (1 - pt(tstat, n-1))*2, pt(tstat, n-1)*2)
pval
```

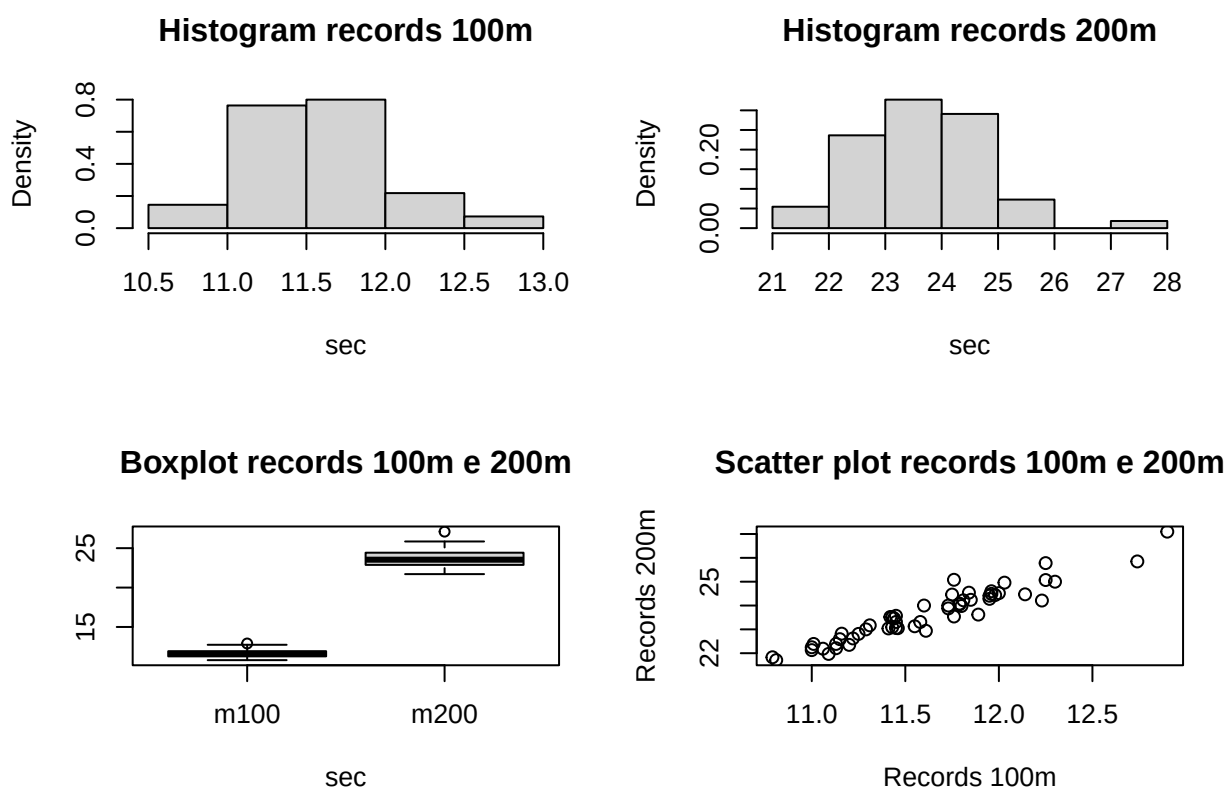
```
## [1] 0.05709702
```

```r
IC = c( inf     = sample.mean - sample.sd / sqrt(n) * qt(1 - alpha/2, n-1),
        center  = sample.mean,
        sup     = sample.mean + sample.sd / sqrt(n) * qt(1 - alpha/2, n-1))
IC
```

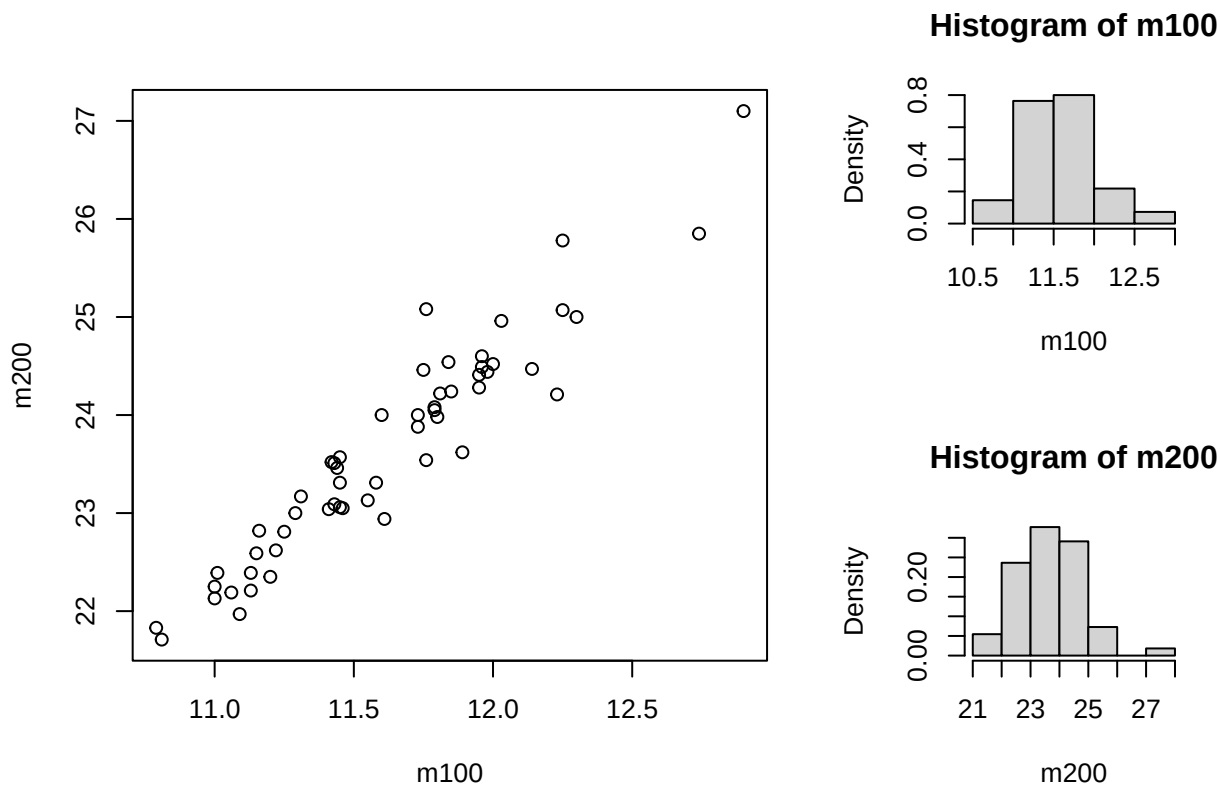
```
##      inf   center      sup
## 11.49630 11.61855 11.74080
```

1.2.2 Linear regression

```r
##### Simple linear regression (variable m200 vs m100)

# More than one plot in a unique
# (command par)

par(mfrow=c(2, 2))
hist(m100, prob=T, main="Histogram records 100m", xlab="sec")
hist(m200, prob=T, main="Histogram records 200m", xlab="sec")
boxplot(record[,1:2], main="Boxplot records 100m e 200m", xlab="sec")
plot(m100, m200, main='Scatter plot records 100m e 200m', xlab="Records 100m", ylab="Records 200m")
```




```r
#! funnel (? ciminiera) for higher m100 we see an higher variability

# command layout
layout(cbind(c(1, 1), c(2, 3)), widths=c(2, 1), heights=c(1, 1))
plot(m100, m200)
hist(m100, prob=T)
hist(m200, prob=T)
```




```r
# Fit of the linear model (command lm)
# Model: m200 = beta0 + beta1 * m100 + eps, eps ~ N(0, sigma^2)
#! Ie regress m200 having at our disposal m100
regression = lm(m200 ~ m100)
regression
```

```
##
## Call:
## lm(formula = m200 ~ m100)
##
## Coefficients:
## (Intercept)         m100
##      -3.557        2.341
```

```r
#! all'esame scrivi in natural language, il codice non conta troppo

summary(regression)
```

```
##
## Call:
## lm(formula = m200 ~ m100)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.86303 -0.16559 -0.00756  0.16599  1.10722
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.5570     1.1914  -2.985  0.00428 **
## m100          2.3410     0.1025  22.845  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3405 on 53 degrees of freedom
## Multiple R-squared:  0.9078, Adjusted R-squared:  0.9061
## F-statistic: 521.9 on 1 and 53 DF,  p-value: < 2.2e-16
```

```r
#! the std
#! the residual std error is the one of sigma^2 for the eps

coef(regression)
```

```
## (Intercept)        m100
##   -3.556967    2.340965
```

```r
vcov(regression)
```

```
##             (Intercept)        m100
## (Intercept)   1.4195378 -0.12199717
## m100         -0.1219972  0.01050021
```

```r
residuals(regression)
```

```
##            1            2            3            4            5            6
## -0.681631757 -0.311836293 -0.110258139 -0.113438848 -0.220487075  0.250657607
##            7            8            9           10           11           12
## -0.392342968  0.056356617 -0.014607931 -0.007559704  0.401777888  0.458523975
##           13           14           15           16           17           18
##  0.159030651 -0.434330192  0.343151507  0.006994624 -0.107968774  0.045211935
##           19           20           21           22           23           24
## -0.038860119  0.172946971 -0.063643383  0.036994624  0.379946396 -0.187077430
##           25           26           27           28           29           30
## -0.137559704  0.056536751  0.309741861  0.322922570  0.127476898  0.097452497
##           31           32           33           34           35           36
## -0.022547503  0.049030651  0.660150932  0.355163132 -0.863029777  1.107223560
##           37           38           39           40           41           42
## -0.657101831  0.031115480 -0.351173885 -0.241402821 -0.049849068 -0.432776440
##           43           44           45           46           47           48
## -0.287968774  0.130175333  0.236332216 -0.236897296 -0.086415022  0.251802289
##           49           50           51           52           53           54
##  0.062922570 -0.088655584  0.510633206 -0.047788640  0.127959172 -0.144101256
##           55
## -0.416921697
```

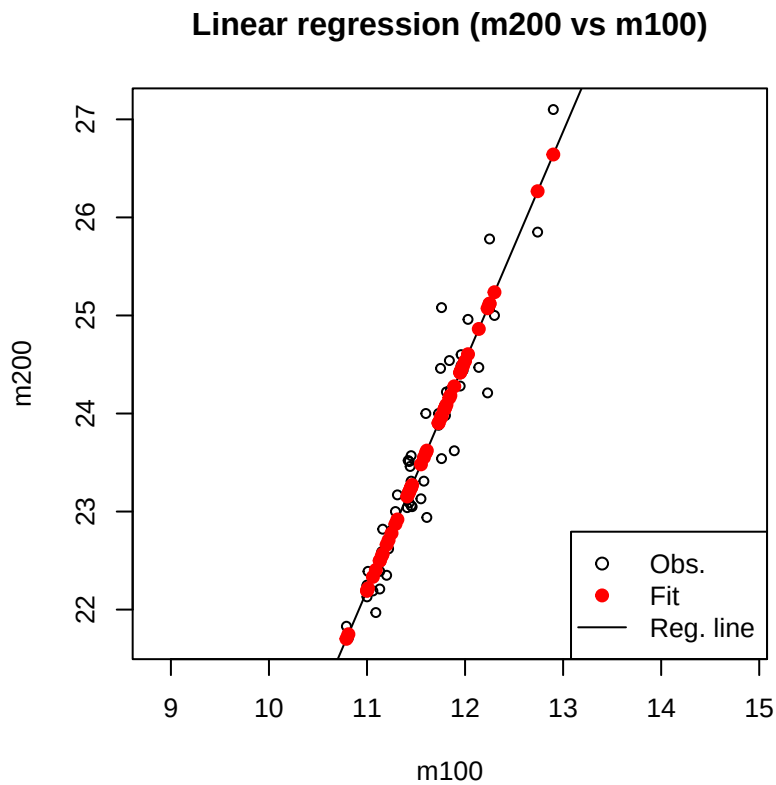
```r
fitted(regression)
```

```
##        1        2        3        4        5        6        7        8
## 23.62163 22.66184 23.20026 23.15344 23.27049 22.91934 24.86234 22.19364
##        9       10       11       12       13       14       15       16
## 24.53461 24.41756 23.59822 26.64148 24.44097 22.40433 23.17685 24.04301
##       17       18       19       20       21       22       23       24
## 22.49797 22.54479 21.74886 22.21705 22.19364 24.04301 24.16005 23.24708
##       25       26       27       28       29       30       31       32
## 24.41756 24.18346 23.20026 23.24708 22.87252 23.90255 23.90255 24.44097
##       33       34       35       36       37       38       39       40
## 25.11985 24.60484 25.07303 23.97278 24.27710 22.77888 23.48117 23.55140
##       41       42       43       44       45       46       47       48
## 25.11985 23.97278 22.49797 24.08982 23.22367 25.23690 24.06642 22.56820
##       49       50       51       52       53       54       55
## 23.24708 22.70866 23.94937 24.48779 21.70204 22.33410 26.26692
```

```r
plot(m100, m200, asp=1, cex=0.75)
abline(coef(regression))
points(m100, fitted(regression), col='red', pch=19)

legend(
  'bottomright',
  c('Obs.', 'Fit', 'Reg. line'),
  col = c('black', 'red', 'black'),
  lwd = c(1, 1, 1),
  lty = c(-1, -1, 1),
  pch = c(c(1, 19, -1))
)

title(main='Linear regression (m200 vs m100)')
```

```r
# Test F "by hand" (H0: beta1=0 vs H1: beta1!=0)
SSreg = sum((fitted(regression) - mean(m200))^2)
SSres = sum(residuals(regression)^2)
SStot = sum((m200 - mean(m200))^2)

n = length(m200)
Fstat = (SSreg/1) / (SSres/(n-2))
P = 1 - pf(Fstat, 1, n-2)
P # reject H0
```

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



1.2.3 Conf and Pred intervals

```r
# Confidence and prediction intervals (command predict)
newdata = data.frame(m100=c(10, 11, 12))
pred_nd = predict(regression, newdata)
pred_nd
```

```
##        1        2        3
## 19.85268 22.19364 24.53461
```

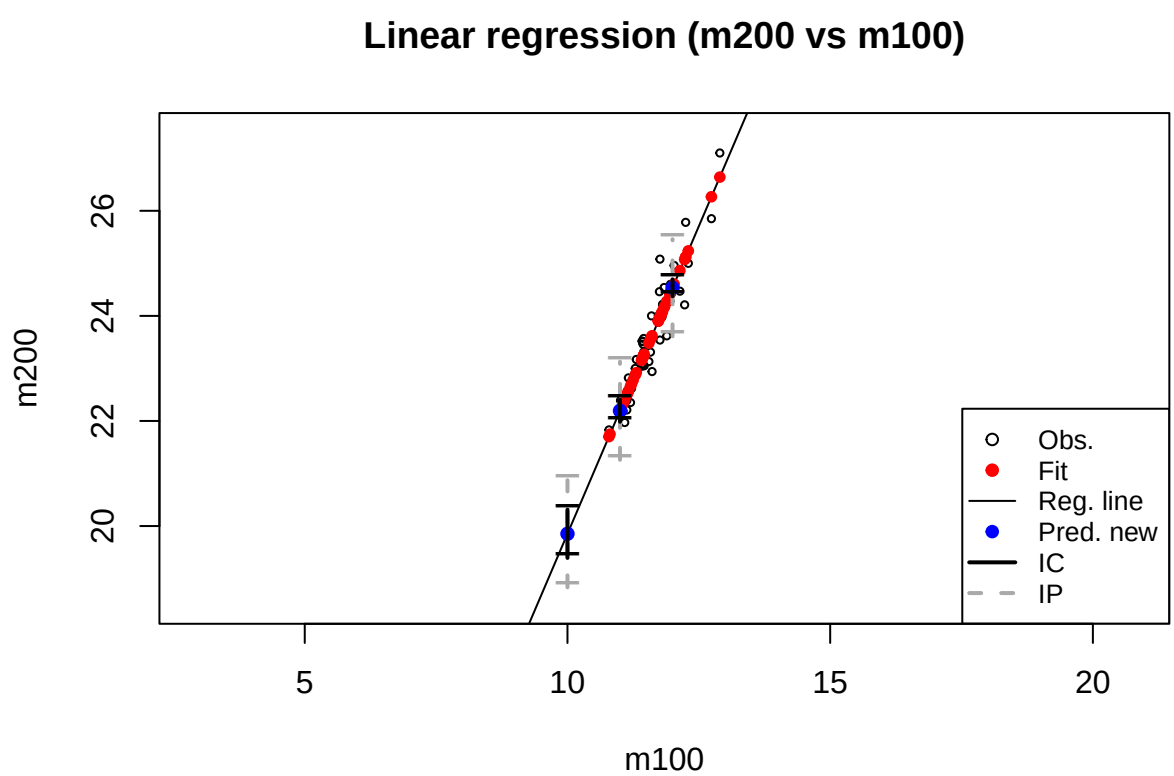
```r
IC_nd = predict(regression, newdata, interval='confidence', level=.99)
IC_nd
```

```
##        fit      lwr      upr
## 1 19.85268 19.39288 20.31248
## 2 22.19364 21.98453 22.40276
## 3 24.53461 24.37350 24.69572
```

```r
#! confidence: we are estimating the mean of Y given a certain X=x
#! interval to enclose the mean
IP_nd = predict(regression, newdata, interval='prediction', level=.99)
IP_nd
```

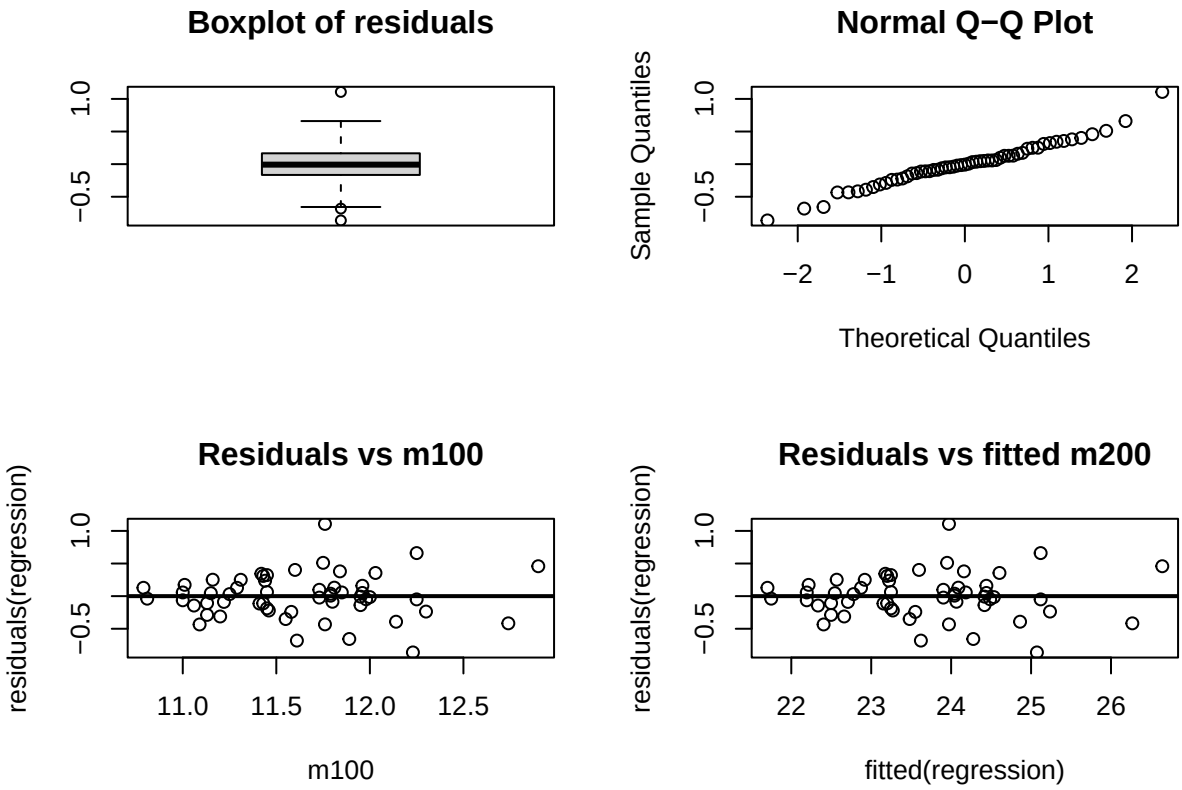
```
##        fit      lwr      upr
## 1 19.85268 18.83330 20.87206
## 2 22.19364 21.26013 23.12716
## 3 24.53461 23.61066 25.45856
```

```r
#! prediction: we have X=x, and we want an interval to cover 99% of the Y
#! given that X. So now we look at the distribution of Y given X=x
#! So an interval for where the values (no more the mean) of Y would be, given
#! the uncertainty we had in estimating the mean of Y through the regression

plot(m100, m200, asp=1, ylim=c(18.5, 27.5), cex=0.5)
abline(coef(regression))
points(m100, fitted(regression), col='red', pch=20)
points(c(10, 11, 12), pred_nd, col='blue', pch=16)

matlines(rbind(c(10, 11, 12), c(10, 11, 12)), t(IP_nd[, -1]), type="l", lty=2,
         col='dark grey', lwd=2)
matpoints(rbind(c(10, 11, 12), c(10, 11, 12)), t(IP_nd[, -1]), pch="-", lty=2,
          col='dark grey', lwd=2, cex=1.5)
matlines(rbind(c(10, 11, 12), c(10, 11, 12)), t(IC_nd[, -1]), type="l", lty=1,
         col='black', lwd=2)
matpoints(rbind(c(10, 11, 12), c(10, 11, 12)), t(IC_nd[, -1]), pch="-", lty=1,
          col='black', lwd=2, cex=1.5)

legend(
  'bottomright',
  c('Obs.', 'Fit', 'Reg. line', 'Pred. new', 'IC', 'IP'),
  col = c('black', 'red', 'black', 'blue', 'black', 'dark grey'),
  lwd = c(1, 1, 1, 1, 2, 2),
  lty = c(-1, -1, 1, -1, 1, 2),
  pch = c(c(1, 19, -1, 19, -1, -1)),
  cex=0.8
)

title(main='Linear regression (m200 vs m100)')
```

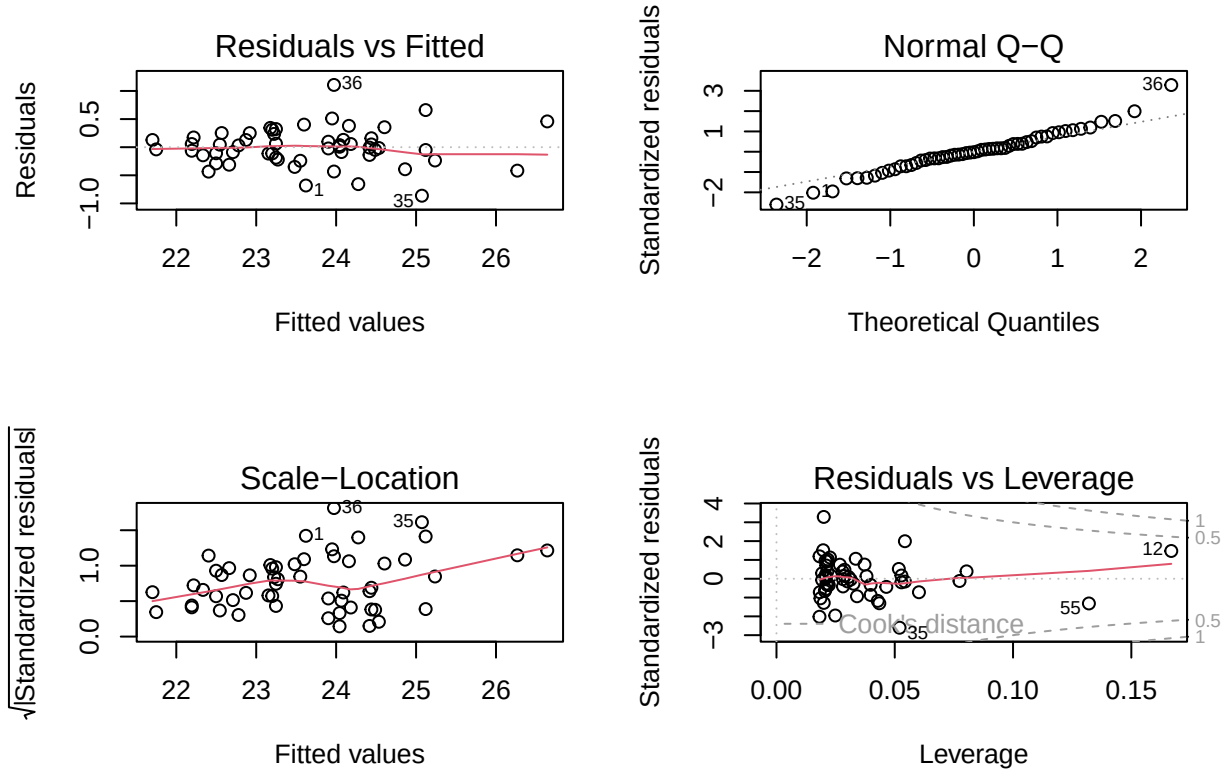


1.2.4 Residuals

```
# diagnostic of residuals
par(mfrow=c(2, 2))
boxplot(residuals(regression), main='Boxplot of residuals')
qqnorm(residuals(regression))
plot(m100, residuals(regression), main='Residuals vs m100')
abline(h=0, lwd=2)
plot(fitted(regression), residuals(regression), main='Residuals vs fitted m200')
abline(h=0, lwd=2)
```




```
par(mfrow=c(2, 2))
plot(regression)
```




```
detach(record)
```

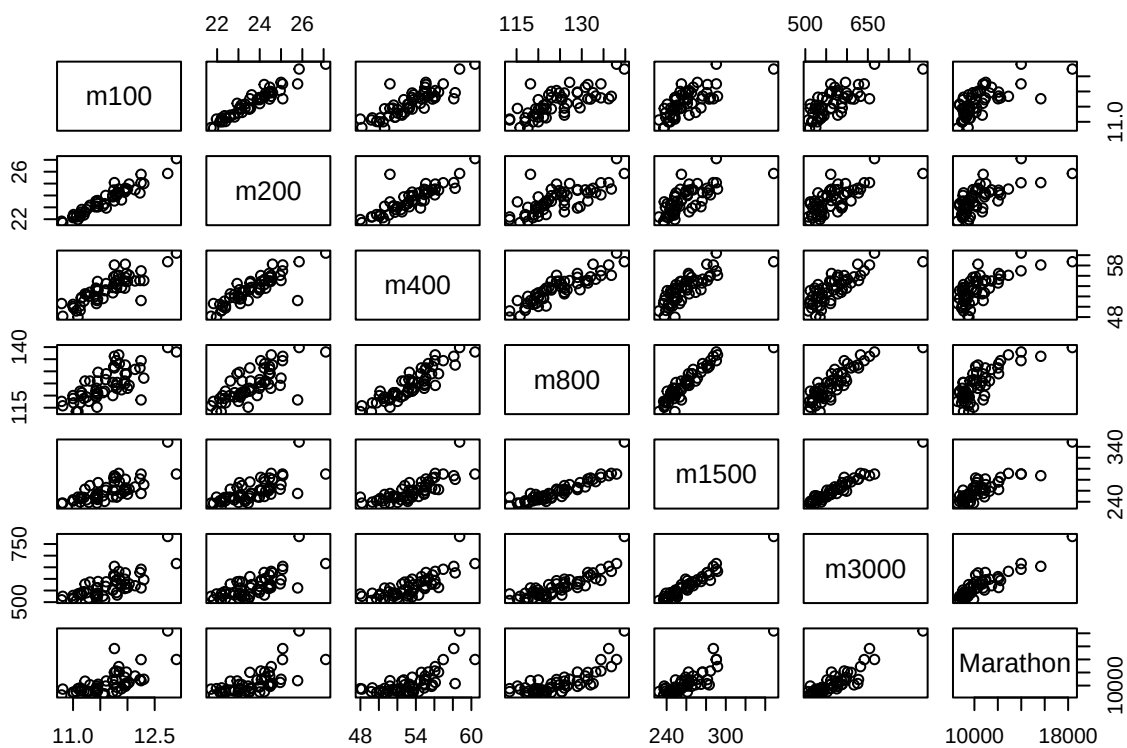
1.3 Multivariate data

```
##### Example 1: dataset record (all the variables)

record = read.table('Lab01/record_mod.txt', header=T)
record
```

```
##           m100  m200  m400  m800 m1500 m3000 Marathon
## argentin 11.61 22.94 54.50 129.0 265.8 587.4  10711.2
## australi 11.20 22.35 51.08 118.8 247.8 544.8   9142.2
## austria  11.43 23.09 50.62 119.4 253.2 560.4   9562.2
## belgium  11.41 23.04 52.00 120.0 248.4 532.8   9471.0
## bermuda  11.46 23.05 53.30 129.6 274.8 588.6  10198.8
## brazil   11.31 23.17 52.80 126.0 269.4 586.2  10125.0
## burma    12.14 24.47 55.00 130.8 267.0 570.6  11461.2
## canada   11.00 22.25 50.06 120.0 243.6 528.6   8967.0
## chile    12.00 24.52 54.90 123.0 253.8 562.2  10282.8
## china    11.95 24.41 54.97 124.8 259.8 558.6  10108.8
## columbia 11.60 24.00 53.26 126.6 261.0 567.6   9925.2
## cookis   12.90 27.10 60.40 138.0 290.4 666.0  13993.2
## costa    11.96 24.60 58.25 132.6 280.8 625.8  10308.0
## czech    11.09 21.97 47.99 113.4 248.4 535.2   9531.0
## denmark  11.42 23.52 53.60 121.8 250.8 522.6   9105.0
## domrep   11.79 24.05 56.05 134.4 284.4 593.4  12232.8
## finland  11.13 22.39 50.14 121.8 246.0 535.2   9253.8
## france   11.15 22.59 51.73 120.0 248.4 538.8   9316.2
## gdr      10.81 21.71 48.16 115.8 237.6 525.0   9460.8
## frg      11.01 22.39 49.75 117.0 241.8 515.4   8911.8
## gbni     11.00 22.13 50.46 118.8 241.8 517.2   8983.2
## greece   11.79 24.08 54.93 124.2 261.0 592.2  10932.0
## guatemal 11.84 24.54 56.09 136.8 291.6 632.4  12904.8
## hungary  11.45 23.06 51.50 120.6 248.4 538.8   9382.2
## india    11.95 24.28 53.60 126.0 259.2 598.8  11281.8
## indonesi 11.85 24.24 55.34 133.2 276.6 601.2  12076.8
## ireland  11.43 23.51 53.24 123.0 246.6 533.4   8962.8
```

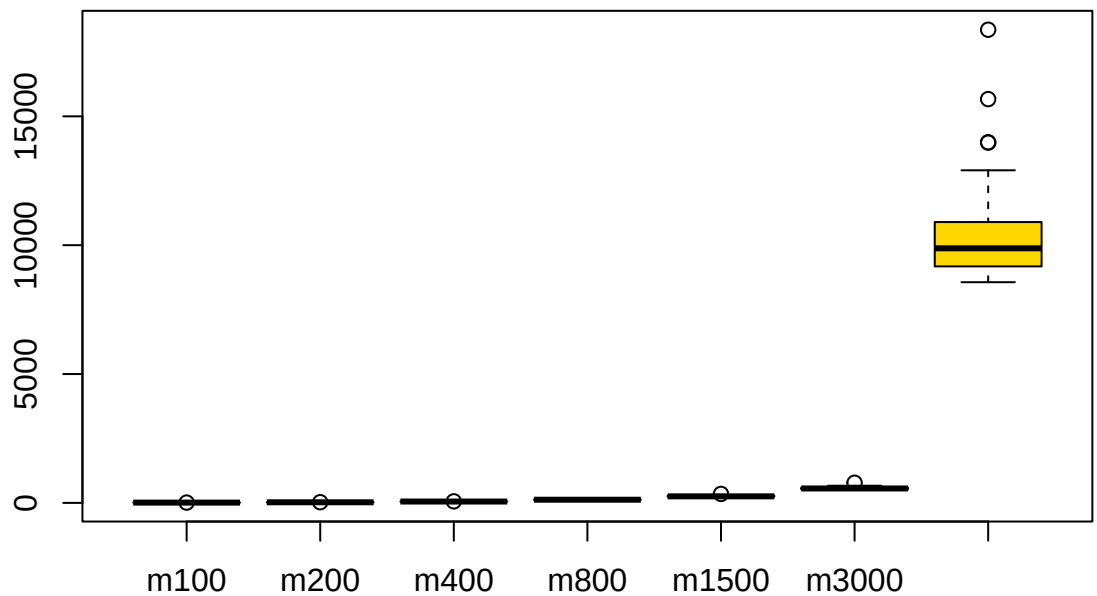
```
## israel   11.45 23.57 54.90 126.0 255.0 562.2   9628.8
## italy    11.29 23.00 52.01 117.6 238.8 517.8   9109.2
## japan    11.73 24.00 53.73 125.4 261.0 552.0   9030.0
## kenya    11.73 23.88 52.70 120.0 249.0 552.0  10863.0
## korea    11.96 24.49 55.70 129.0 265.2 577.2   9879.0
## dprkorea 12.25 25.78 51.20 118.2 255.0 561.0  10750.2
## luxembou 12.03 24.96 56.10 124.2 262.8 578.4  10480.8
## malaysia 12.23 24.21 55.09 131.4 281.4 627.6  10930.2
## mauritiu 11.76 25.08 58.10 136.2 287.4 654.0  15667.8
## mexico   11.89 23.62 53.76 122.4 255.0 575.4   9511.8
## netherla 11.25 22.81 52.38 119.4 243.6 540.6   9148.8
## nz       11.55 23.13 51.60 121.2 250.8 525.6   8728.8
## norway   11.58 23.31 53.12 121.8 240.6 511.8   8728.8
## png      12.25 25.07 56.96 134.4 290.4 641.4  13980.0
## philippi 11.76 23.54 54.60 131.4 276.0 609.6  12022.2
## poland   11.13 22.21 49.29 117.0 239.4 538.2   9649.2
## portugal 11.81 24.22 54.30 125.4 249.6 530.4   9072.0
## rumania  11.44 23.46 51.20 115.2 237.6 511.8   9927.0
## singapor 12.30 25.00 55.08 127.2 271.2 596.4  10966.2
## spain    11.80 23.98 53.59 123.0 248.4 541.2   9756.0
## sweden   11.16 22.82 51.79 121.2 247.2 530.4   9268.8
## switzerl 11.45 23.31 53.11 121.2 244.2 526.2   9205.2
## taipei   11.22 22.62 52.50 126.0 262.8 577.8  10672.2
## thailand 11.75 24.46 55.80 132.0 283.2 616.8  10107.0
## turkey   11.98 24.44 56.45 129.0 262.2 562.8  12064.8
## usa      10.79 21.83 50.62 117.6 237.0 510.0   8563.2
## ussr     11.06 22.19 49.19 113.4 232.2 507.0   9073.2
## wsamoa   12.74 25.85 58.73 139.8 348.6 782.4  18360.0
```

```r
# Scatterplot

pairs(record)  # or plot(record)
```

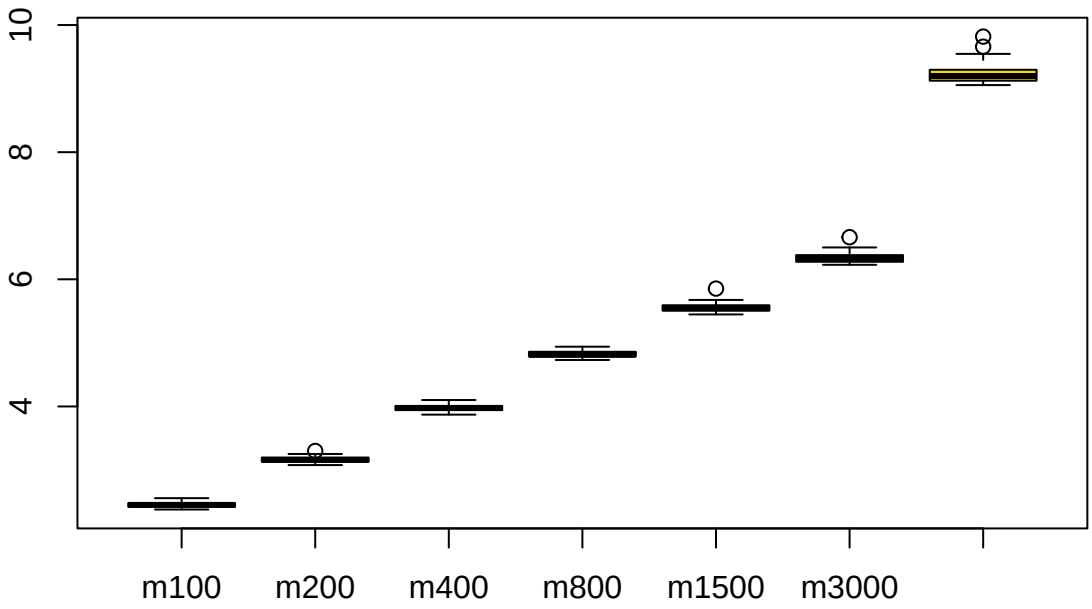



```r
#! prefer pairs than just cor as with that we can see also different
#! correlations than the linear ones (ln x, sin x, ecc)
#! As we dont have to keep the data as we receive them, we can modify them
#! to recover linear models with the transformed variables
```

```r
# Boxplot
boxplot(record, col='gold')
```

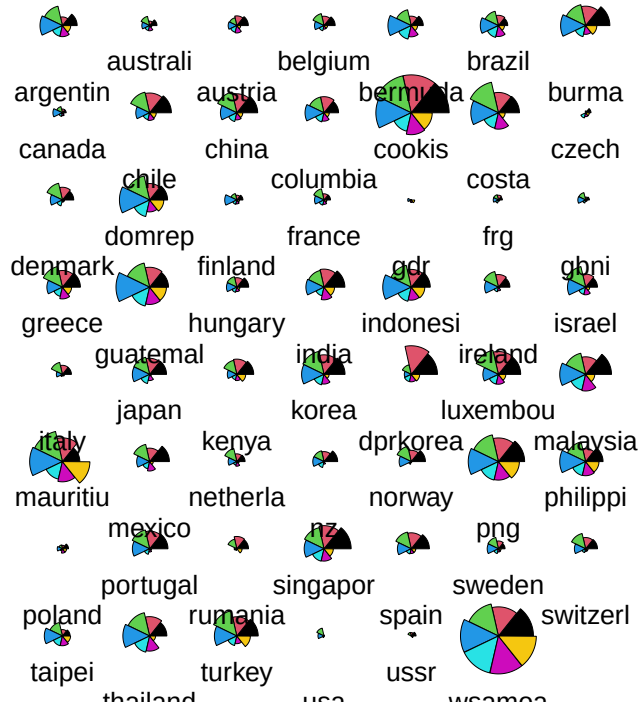



```r
#! we can see that marathon is a direction that captures most of the variability
#! so it can be a principal direction. But thats also because the scale of that
#! variable well is very different from the others (something in spans of hours vs
#! sometghing in spans of seconds)
#! So we have to scale (standardize?) our variables first

#! maybe
# boxplot((record-sapply(record,mean))/sapply(record,sd), col='gold')

boxplot(log(record), col='gold')
```




```r
# Starplot
stars(record, col.stars=rep('gold',55))
```

```r
# Radarplot
stars(record, draw.segments=T)
```




```r
# Chernoff faces
# source('faces.R')
# faces(record)

##### Example 2: cerebral aneurysm
aneurysm = read.table('Lab01/aneurysm.txt', header=T, sep=',')
head(aneurysm)
```

```
##           R1         R2         C1         C2 POSLH ROT
## 1  4.6123692  1.8189348 -0.7113495 -0.8126038     1   1
## 2  3.0635290  0.5358821 -0.2321406  0.2195964     1   2
## 3 -2.5073081 -1.8296361 -0.1477957  1.0275683     2   1
## 4 -0.4831741 -1.3670522 -0.1344654  0.7809211     2   2
## 5  8.6775054 -0.7214820 -0.8779464 -0.8643967     1   1
## 6 -1.4005311  0.3016918  0.3694719  0.8115340     2   1
```

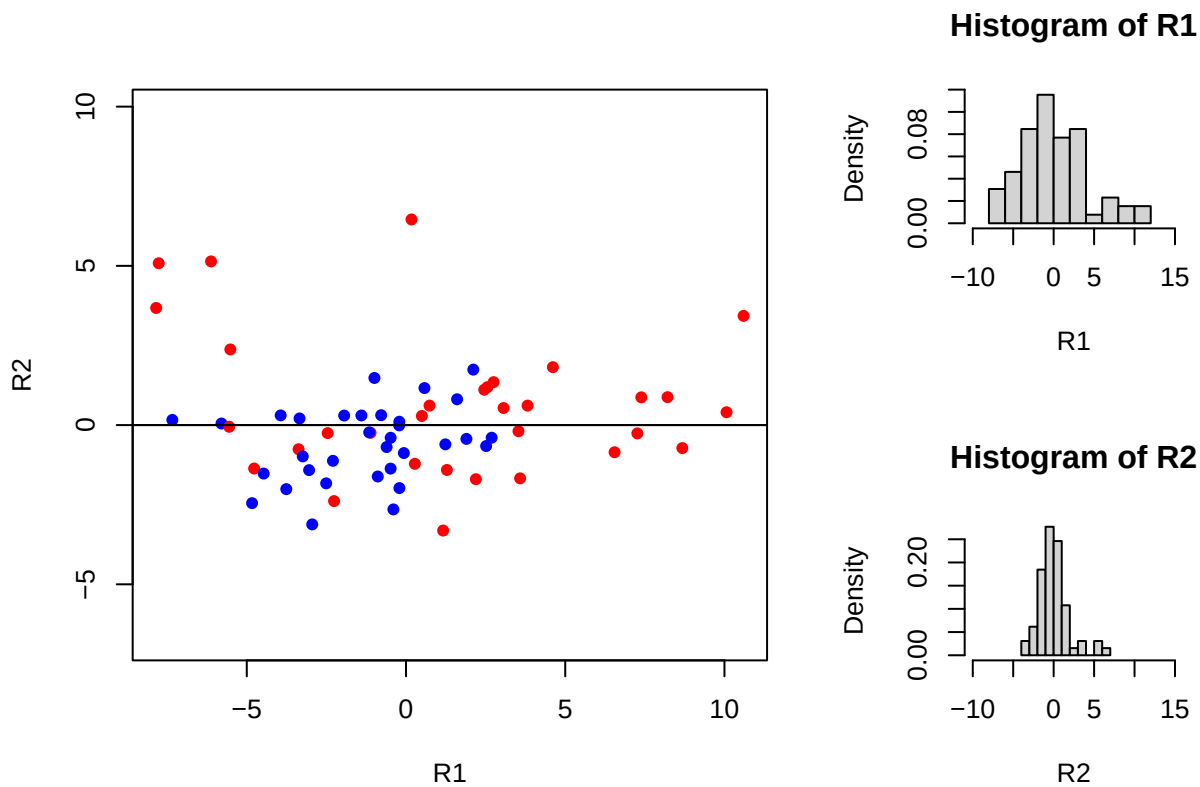
```r
dim(aneurysm)
```

```
## [1] 65  6
```

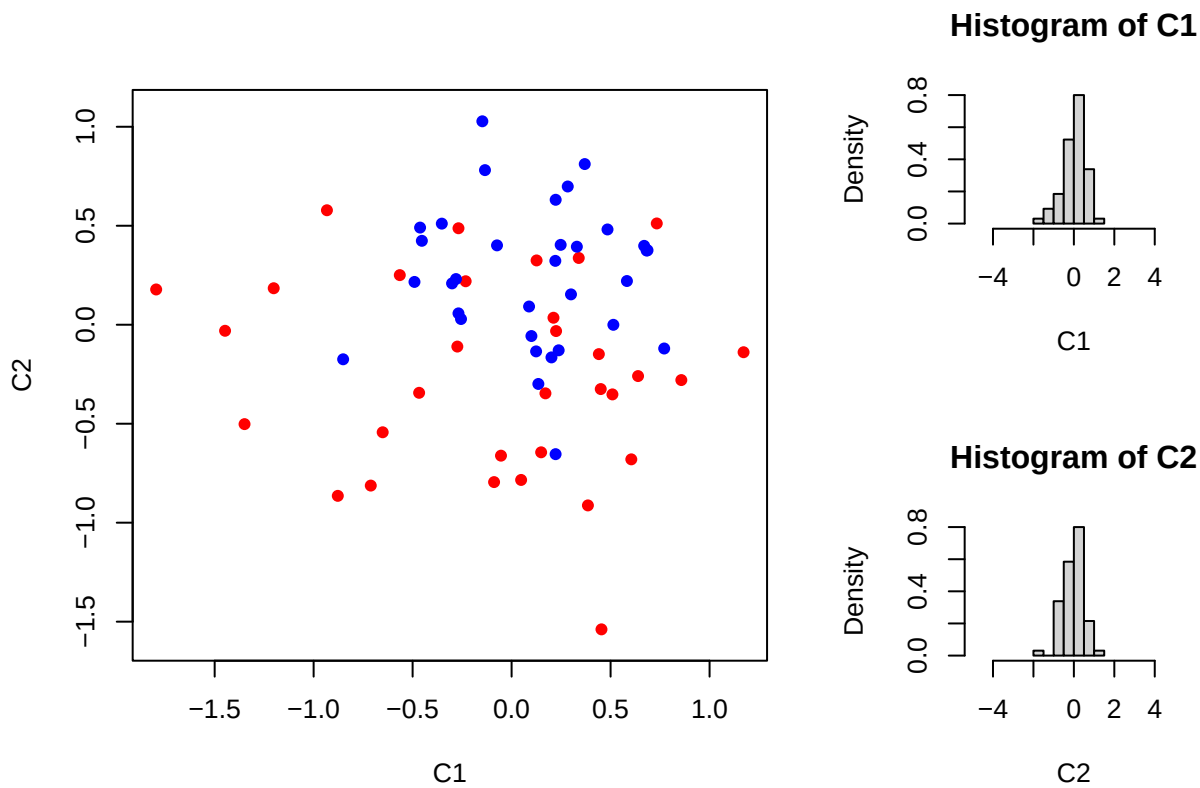
```r
aneurysm.geometry = aneurysm[, 1:4]
aneurysm.position = factor(aneurysm[, 5])

head(aneurysm.geometry)
```

```
##           R1         R2         C1         C2
## 1  4.6123692  1.8189348 -0.7113495 -0.8126038
## 2  3.0635290  0.5358821 -0.2321406  0.2195964
## 3 -2.5073081 -1.8296361 -0.1477957  1.0275683
## 4 -0.4831741 -1.3670522 -0.1344654  0.7809211
## 5  8.6775054 -0.7214820 -0.8779464 -0.8643967
## 6 -1.4005311  0.3016918  0.3694719  0.8115340
```

```
color.position = ifelse(aneurysm.position == '1', 'red', 'blue')

attach(aneurysm.geometry)

layout(cbind(c(1, 1), c(2, 3)), widths=c(2, 1), heights=c(1, 1))
plot(R1, R2, asp=1, col=color.position, pch=16)
abline(h=mean(R1))
abline(h=mean(R2))
hist(R1, prob=T, xlim=c(-10, 15))
hist(R2, prob=T, xlim=c(-10, 15))
```




```
layout(cbind(c(1, 1), c(2, 3)), widths=c(2, 1), heights=c(1, 1))
plot(C1, C2, asp=1, col=color.position, pch=16)
hist(C1, prob=T, xlim=c(-5, 5))
hist(C2, prob=T, xlim=c(-5, 5))
```




```
detach(aneurysm.geometry)

# some statistical indices
sapply(aneurysm.geometry, mean)
```

```
##           R1           R2           C1           C2
## 1.181730e-16 1.798174e-16 1.490403e-16 6.452879e-17
```

```
sapply(aneurysm.geometry, sd)
```

```
##        R1        R2        C1        C2
## 4.1859403 1.8563342 0.5871577 0.4940132
```

```
cov(aneurysm.geometry)
```

```
##               R1            R2            C1            C2
## R1  1.752210e+01 -9.654685e-16 -1.305303e+00 -4.165673e-01
## R2 -9.654685e-16  3.445977e+00 -5.419870e-02 -1.502647e-01
## C1 -1.305303e+00 -5.419870e-02  3.447541e-01  2.946215e-16
## C2 -4.165673e-01 -1.502647e-01  2.946215e-16  2.440490e-01
```

```
cor(aneurysm.geometry)
```

```
##               R1            R2            C1            C2
## R1  1.000000e+00 -1.242479e-16 -5.310843e-01 -2.014436e-01
## R2 -1.242479e-16  1.000000e+00 -4.972537e-02 -1.638560e-01
## C1 -5.310843e-01 -4.972537e-02  1.000000e+00  1.015713e-15
## C2 -2.014436e-01 -1.638560e-01  1.015713e-15  1.000000e+00
```

```
# Attention: rounded zeros!
round(sapply(aneurysm.geometry, mean), 1)
```

```
## R1 R2 C1 C2
##  0  0  0  0
```

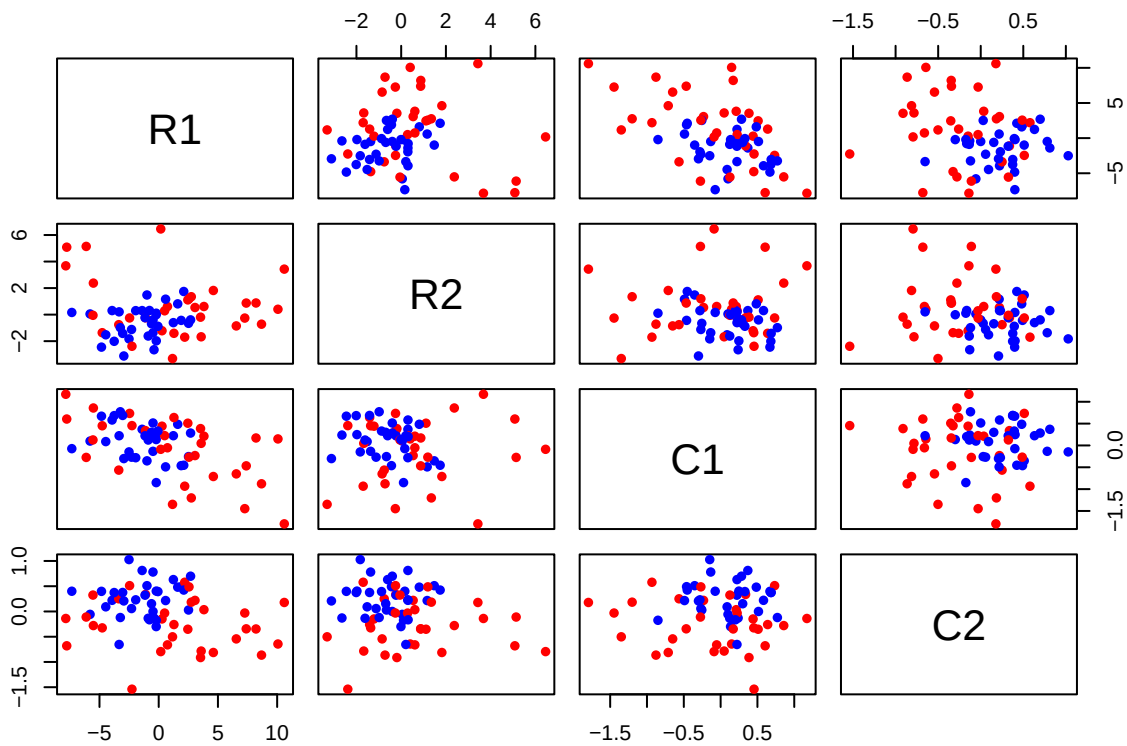
```r
round(cov(aneurysm.geometry), 1)
```

```
##      R1   R2   C1   C2
## R1 17.5  0.0 -1.3 -0.4
## R2  0.0  3.4 -0.1 -0.2
## C1 -1.3 -0.1  0.3  0.0
## C2 -0.4 -0.2  0.0  0.2
```

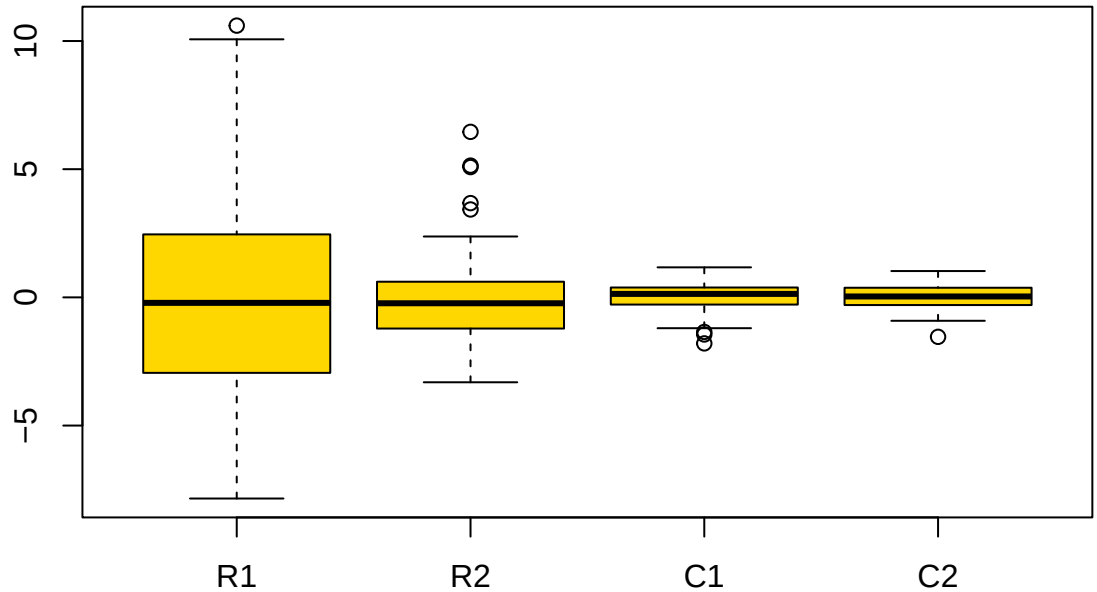
```r
round(cor(aneurysm.geometry), 1)
```

```
##      R1   R2   C1   C2
## R1  1.0  0.0 -0.5 -0.2
## R2  0.0  1.0  0.0 -0.2
## C1 -0.5  0.0  1.0  0.0
## C2 -0.2 -0.2  0.0  1.0
```

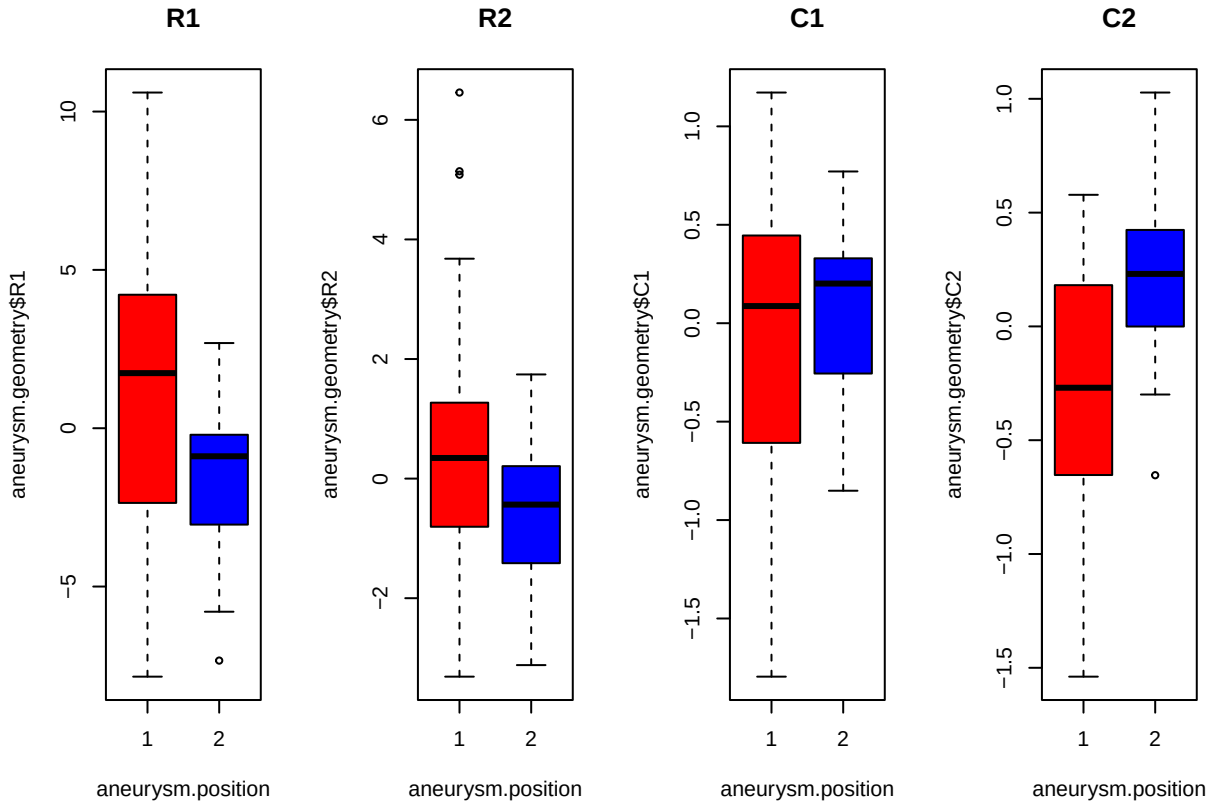
```r
# Scatterplot
pairs(aneurysm.geometry, col=color.position, pch=16)
```



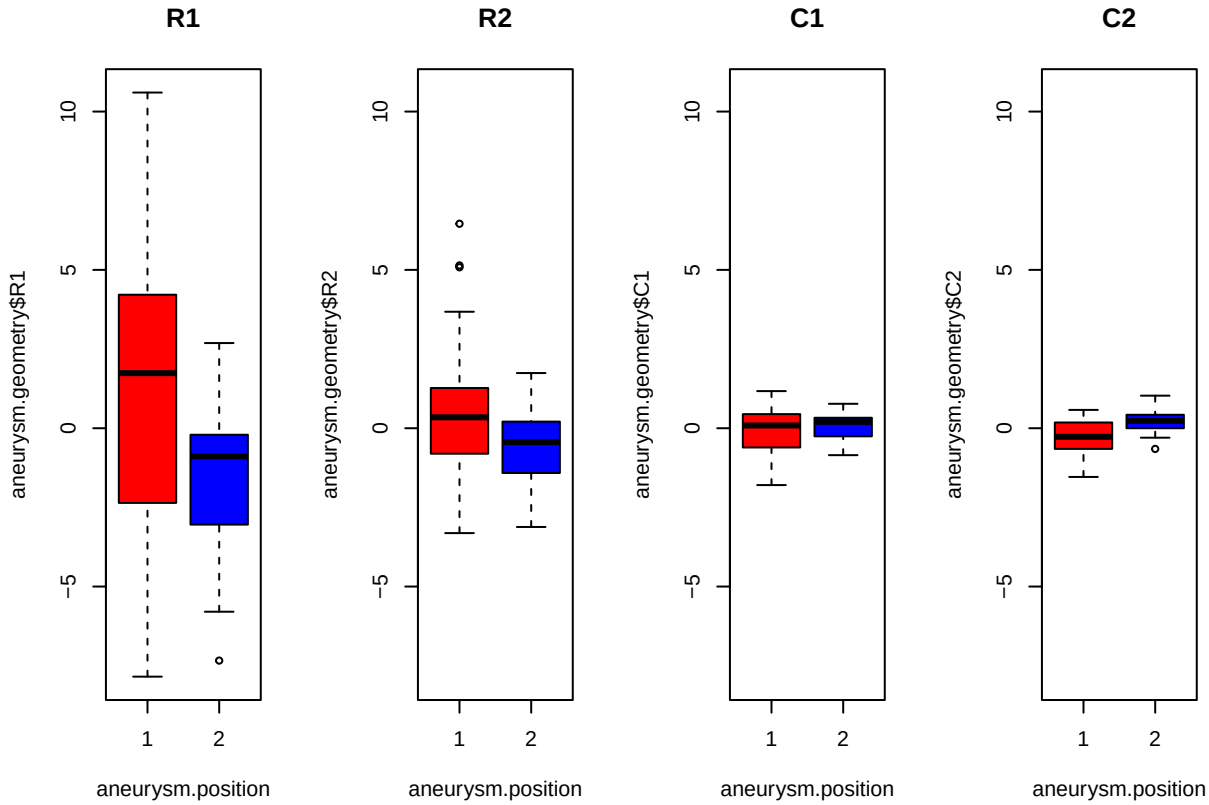

```r
# Boxplot
boxplot(aneurysm.geometry, col='gold')
```




```r
# Stratified boxplots
par(mfrow=c(1, 4))
boxplot(aneurysm.geometry$R1 ~ aneurysm.position, col=c('red', 'blue'), main='R1')
boxplot(aneurysm.geometry$R2 ~ aneurysm.position, col=c('red', 'blue'), main='R2')
boxplot(aneurysm.geometry$C1 ~ aneurysm.position, col=c('red', 'blue'), main='C1')
boxplot(aneurysm.geometry$C2 ~ aneurysm.position, col=c('red', 'blue'), main='C2')
```

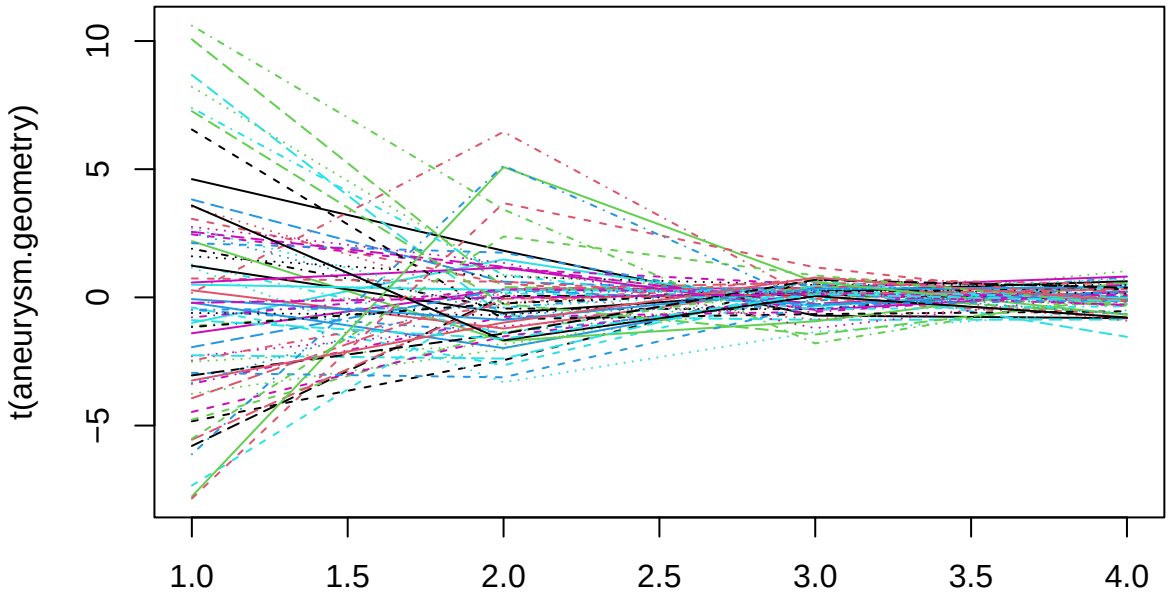



```r
# Stratified boxplots (same scale)
par(mfrow=c(1, 4))
boxplot(aneurysm.geometry$R1 ~ aneurysm.position, col=c('red', 'blue'), main='R1',
        ylim=range(aneurysm.geometry))
boxplot(aneurysm.geometry$R2 ~ aneurysm.position, col=c('red', 'blue'), main='R2',
        ylim=range(aneurysm.geometry))
boxplot(aneurysm.geometry$C1 ~ aneurysm.position, col=c('red', 'blue'), main='C1',
        ylim=range(aneurysm.geometry))
boxplot(aneurysm.geometry$C2 ~ aneurysm.position, col=c('red', 'blue'), main='C2',
        ylim=range(aneurysm.geometry))
```

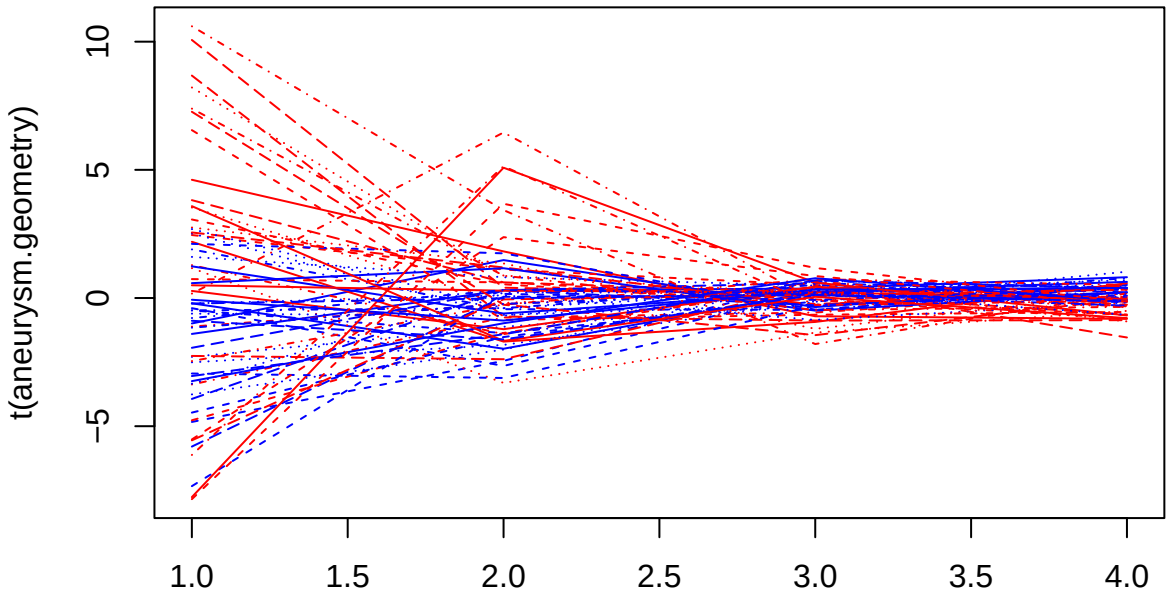



```r
# Chernoff faces
# source('faces.R')
# faces(aneurysm.geometry)
```

```r
# matplot
matplot(t(aneurysm.geometry), type='l')
```




```
matplot(t(aneurysm.geometry), type='l', col=color.position)
```



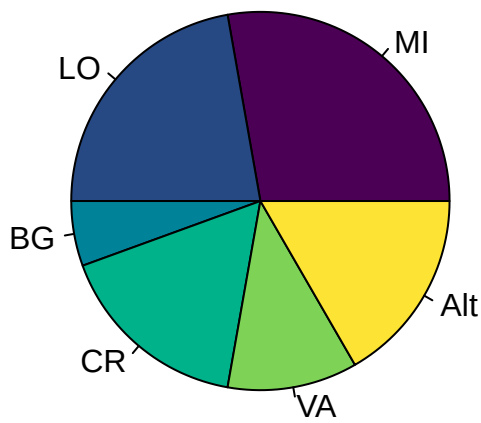
```r
#! on the x we have the different variables (cols?) and we have as many
#! lines as how many observations (rows) we have
```

1.4 Categorical data

```r
##### Visualization of Categorical Data

district = c('MI', 'MI', 'VA', 'BG', 'LO', 'LO', 'CR', 'Alt', 'CR', 'MI',
               'Alt', 'CR', 'LO', 'VA', 'MI', 'Alt', 'LO', 'MI')
district = factor(district, levels=c('MI', 'LO', 'BG', 'CR', 'VA', 'Alt'))
district
```

```
##  [1] MI  MI  VA  BG  LO  LO  CR  Alt CR  MI  Alt CR  LO  VA  MI  Alt LO  MI
## Levels: MI LO BG CR VA Alt
```

```r
# Pie chart (no ordering of levels)

pie(table(district), col=hcl.colors(length(levels(district)), palette = "viridis"))
```

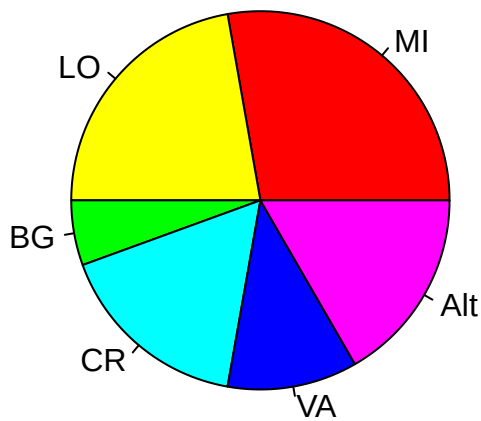



```r
hcl.pals()
```

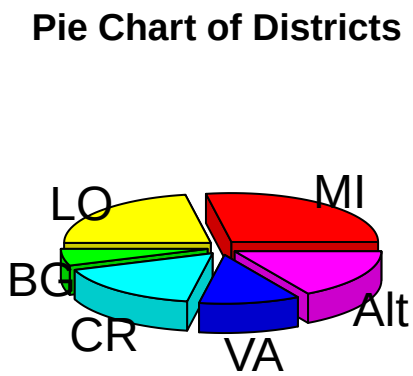
```
##   [1] "Pastel 1"      "Dark 2"        "Dark 3"        "Set 2"
##   [5] "Set 3"         "Warm"          "Cold"          "Harmonic"
##   [9] "Dynamic"       "Grays"         "Light Grays"   "Blues 2"
##  [13] "Blues 3"       "Purples 2"     "Purples 3"     "Reds 2"
##  [17] "Reds 3"        "Greens 2"      "Greens 3"      "Oslo"
##  [21] "Purple-Blue"   "Red-Purple"    "Red-Blue"      "Purple-Orange"
##  [25] "Purple-Yellow" "Blue-Yellow"   "Green-Yellow"  "Red-Yellow"
##  [29] "Heat"          "Heat 2"        "Terrain"       "Terrain 2"
##  [33] "Viridis"       "Plasma"        "Inferno"       "Rocket"
##  [37] "Mako"          "Dark Mint"     "Mint"          "BluGrn"
##  [41] "Teal"          "TealGrn"       "Emrld"         "BluYl"
##  [45] "ag_GrnYl"      "Peach"         "PinkYl"        "Burg"
##  [49] "BurgYl"        "RedOr"         "OrYel"         "Purp"
##  [53] "PurpOr"        "Sunset"        "Magenta"       "SunsetDark"
```

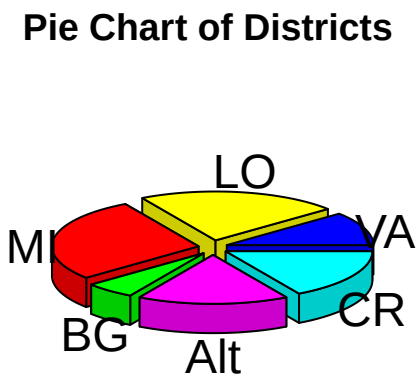
```
##  [57] "ag_Sunset"     "BrwnYl"        "YlOrRd"        "YlOrBr"
##  [61] "OrRd"          "Oranges"       "YlGn"          "YlGnBu"
##  [65] "Reds"          "RdPu"          "PuRd"          "Purples"
##  [69] "PuBuGn"        "PuBu"          "Greens"        "BuGn"
##  [73] "GnBu"          "BuPu"          "Blues"         "Lajolla"
##  [77] "Turku"         "Hawaii"        "Batlow"        "Blue-Red"
##  [81] "Blue-Red 2"    "Blue-Red 3"    "Red-Green"     "Purple-Green"
##  [85] "Purple-Brown"  "Green-Brown"   "Blue-Yellow 2" "Blue-Yellow 3"
##  [89] "Green-Orange"  "Cyan-Magenta"  "Tropic"        "Broc"
##  [93] "Cork"          "Vik"           "Berlin"        "Lisbon"
##  [97] "Tofino"        "ArmyRose"      "Earth"         "Fall"
## [101] "Geyser"        "TealRose"      "Temps"         "PuOr"
## [105] "RdBu"          "RdGy"          "PiYG"          "PRGn"
## [109] "BrBG"          "RdYlBu"        "RdYlGn"        "Spectral"
## [113] "Zissou 1"      "Cividis"       "Roma"
```

```
pie(table(district), col=rainbow(length(levels(district))))
```

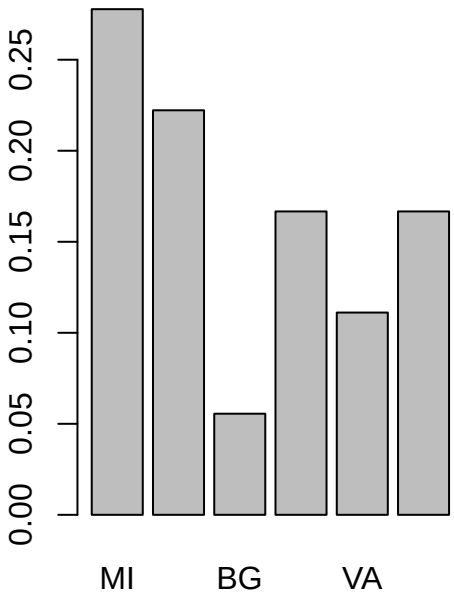


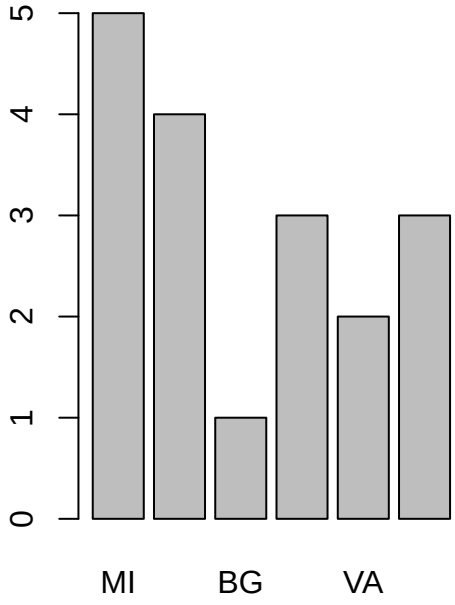

```
# 3D Pie chart (never use them!!)
#! as we want to see the angles in a pie plot, and the 3d stuff deformate them
library(plotrix)

par(mfrow=c(1, 2))

pie3D(
  table(district)[1:length(levels(district))],
  labels = levels(district),
  explode = 0.1,
  main = "Pie Chart of Districts ",
  col = rainbow(length(levels(district)))
)

set.seed(020323)
shuffle = sample(1:length(levels(district)), size=length(levels(district)),
                 replace = F)
pie3D(table(district)[shuffle], labels=levels(district)[shuffle], explode=0.1,
      main="Pie Chart of Districts ", col=rainbow(length(levels(district)))[shuffle])
```






```
# Barplot (levels are ordered)
barplot(table(district) / length(district))

# or
plot(district)   # barplot of absolute frequences
```





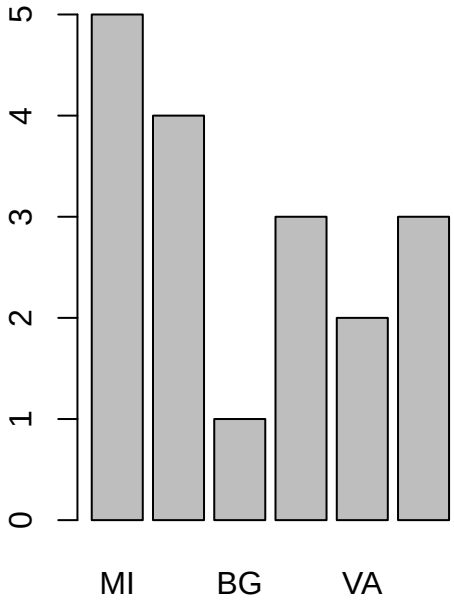
```r
# Remark: Some functions (e.g., the function plot()) may behave differently
# depending on the object it takes as input

is(district)[1]
```

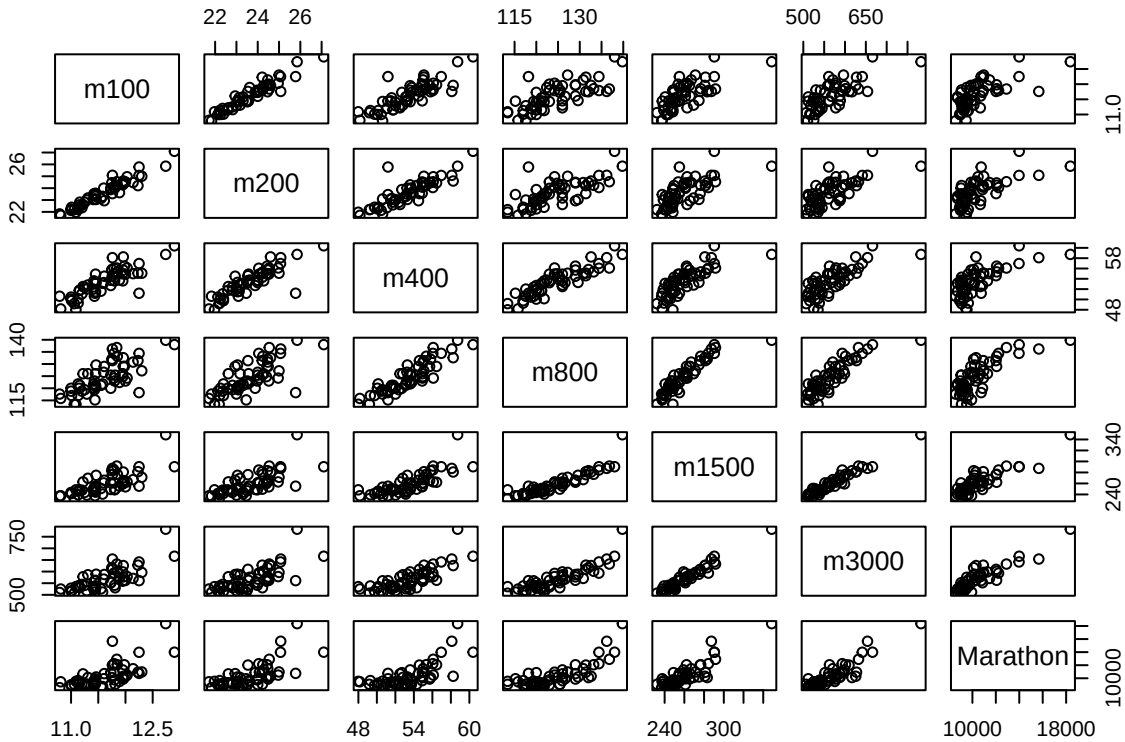
```
## [1] "factor"
```

```r
plot(district)

# record is a data frame
is(record)[1]
```

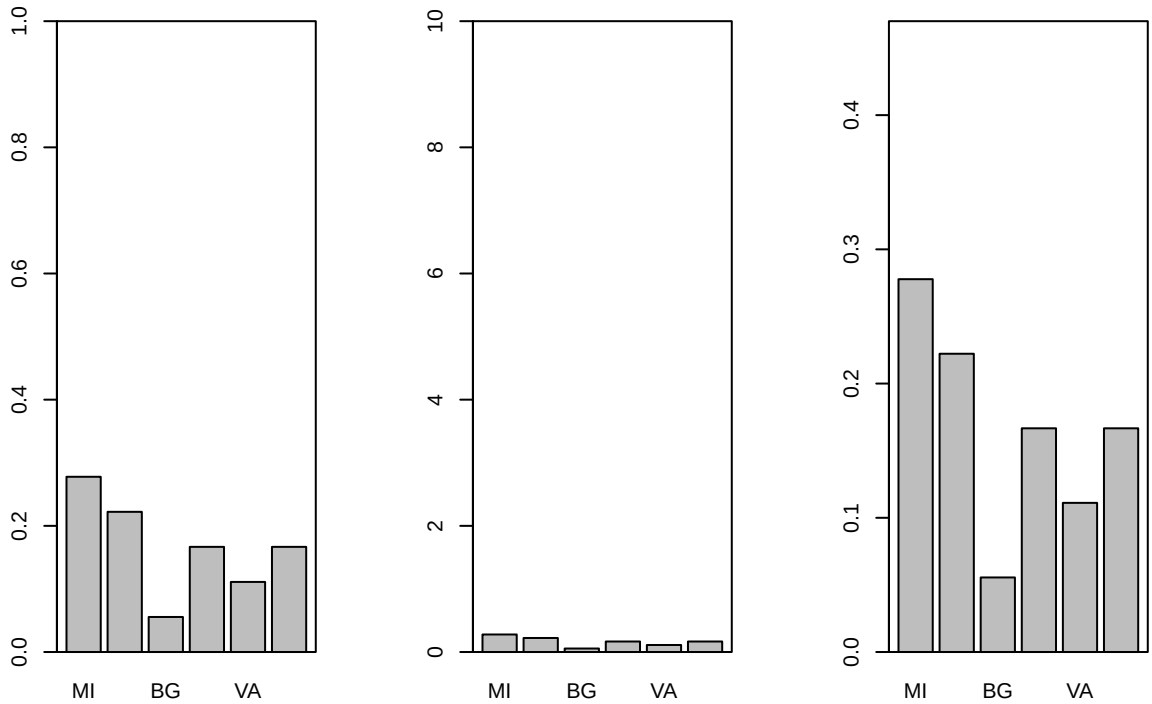
```
## [1] "data.frame"
```

```r
plot(record) # scatterplot
```



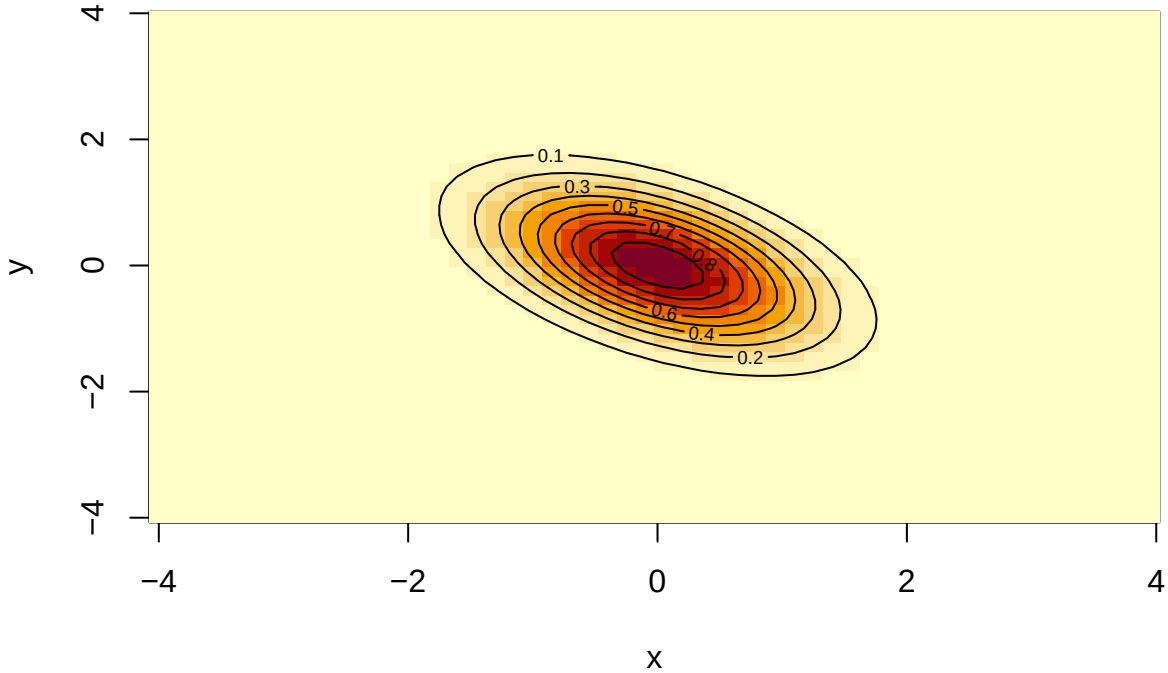



```
# Remark 2: be careful to the scale of representation

par(mfrow=c(1, 3))
barplot(table(district) / length(district), ylim=c(0, 1)); box()
barplot(table(district)/length(district),ylim=c(0, 10)); box()
barplot(table(district)/length(district),ylim=c(0, 0.47)); box()
```

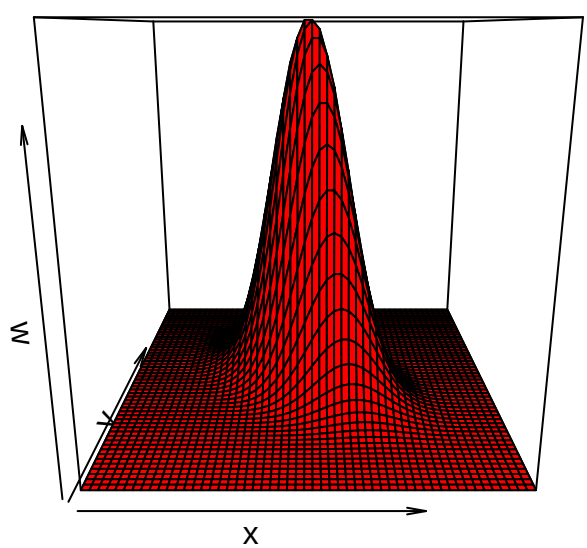


1.5 3d plots: bivariate gaussian

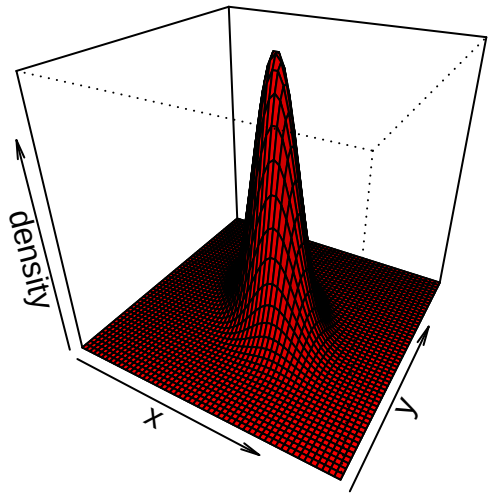
```
### 3d plots (for instance, let's plot a bivariate Gaussian density)

x = seq(-4, 4, 0.15)
y = seq(-4, 4, 0.15)

# To build a function in R
gaussian = function(x, y) {
  exp(-(x^2 + y^2 + x * y))
}

w = matrix(NA, nrow = length(x), ncol=length(y))

# for
for(i in 1:length(x)) {
  for(j in 1:length(y)) {
    w[i, j] = gaussian(x[i], y[j])
  }
}

# or
w = outer(x, y, gaussian)
# help(outer)

image(x, y, w)
contour(x, y, w, add=T)
```



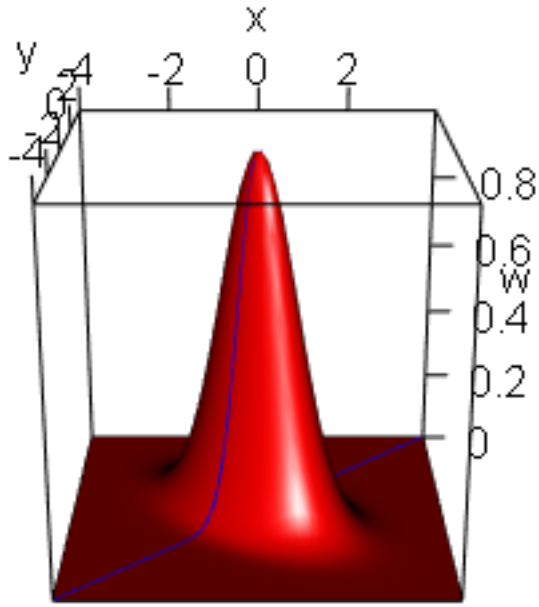

```
persp(x, y, w, col='red')
```



```r
persp(x, y, w, col='red', theta=30, phi=30, shade=.05, zlab='density')
```




```r
# To download a package:
# from RStudio: Tools -> Install Packages -> type PACKAGENAME
#                and click install
# from R: Packages -> Install Packages -> Choose a CRAN mirror
#         (e.g., Italy (Milano)) -> Choose the package and click OK
# or install.packages('PACKAGENAME') in a R console

library(rgl)
options(rgl.printRglwidget = TRUE)
persp3d(x, y, w, col='red', alpha=1)
lines3d(x,x, gaussian(x,x), col='blue', lty=1)
lines3d(x,x, 0, col='blue', lty=2)
```



1.6 Saving plots

```
##### Save plots

# plot(rnorm(10), rnorm(10))
#
# bmp(file="myplot.bmp")
# plot(rnorm(10), rnorm(10))
#
# jpeg(file="myplot.jpeg")
# plot(rnorm(10), rnorm(10))
#
# png(file="myplot.png")
# plot(rnorm(10), rnorm(10))
#
# pdf(file="myplot.pdf")
# plot(rnorm(10), rnorm(10))
#
# pdf(file="myplot2.pdf", onefile=T)
# plot(rnorm(10), rnorm(10))
# plot(rnorm(10), rnorm(10))
# plot(rnorm(10), rnorm(10))
```

2 Lab 02 - First steps into statistics

2.1 Probability density functions

```
###--------------------###
###        LAB 2       ###
###--------------------###

### TOPICS:
### Probability density functions, Cumulative distribution functions, Quantiles
### Random number generation
### QQplots
#_____________________________________________________________________________
```

```r
### Probability density functions, Cumulative distribution functions, Quantiles

### Example: Gaussian distribution
# Probability density function (pdf)
dnorm(0)                  # density function at 0 for a distribution N(0,1)
```

```
## [1] 0.3989423
```

```r
dnorm(0, mean=1, sd=2)    # density function at 0 for a distribution N(1,4)
```

```
## [1] 0.1760327
```

```r
# Cumulative distribution function (cdf)
pnorm(0)       # P(Z<0), with Z ~ N(0,1)
```

```
## [1] 0.5
```

```r
pnorm(0, 1, 2) # P(X<0), with X ~ N(1,4)
```

```
## [1] 0.3085375
```

```r
# Quantiles (inverse of cdf)
qnorm(0.95)        #  = z : P(Z<z)=0.95, with Z ~ N(0,1)
```

```
## [1] 1.644854
```

```r
qnorm(0.95, 1, 2)  #  = z : P(Z<z)=0.95, with Z ~ N(1,4)
```

```
## [1] 4.289707
```

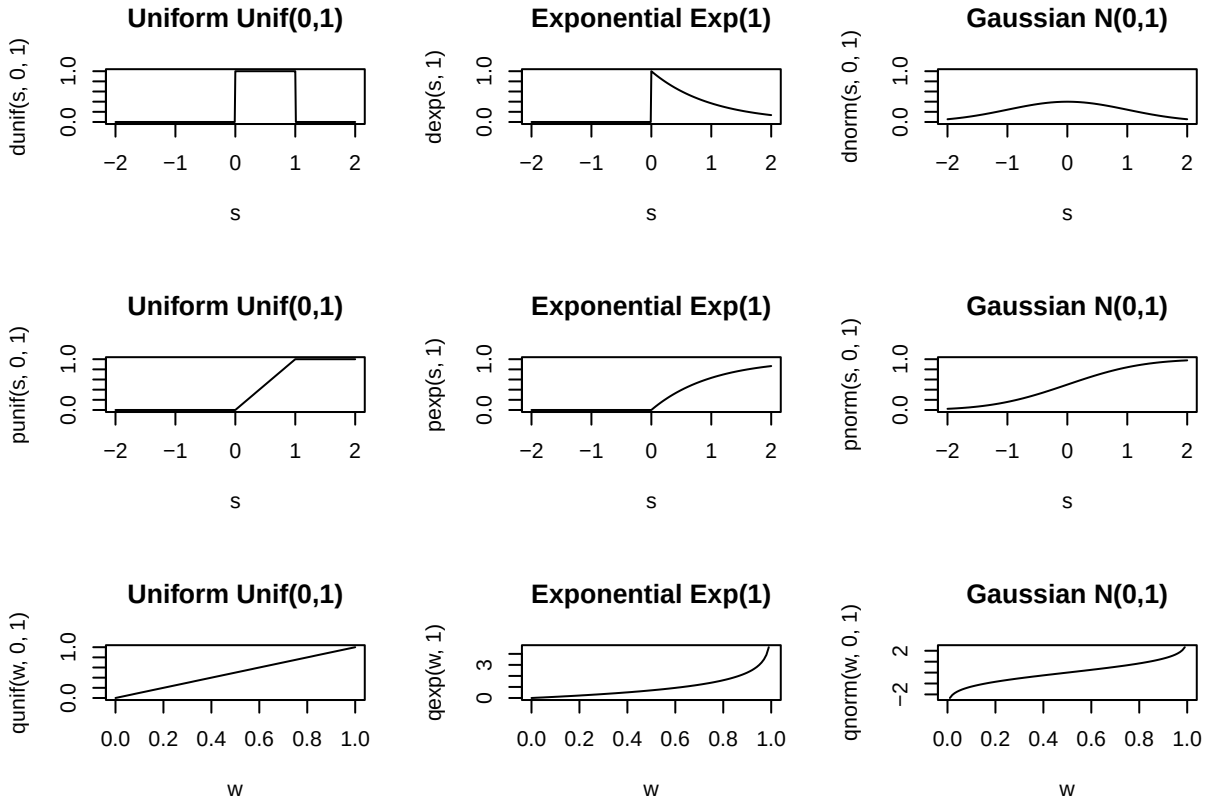
```r
# Random generation
set.seed(8321)
rnorm(10)       # X_1, ..., X_10 ~ N(0,1) iid
```

```
##  [1] -3.54704141 -1.19365815  2.20509244 -0.25489645  0.31960477 -0.14815276
##  [7] -2.31337746 -0.52999541 -1.97466223 -0.07114751
```

```r
rnorm(10, 1, 2) # X_1, ..,, X_10 ~ N(1,4) iid
```

```
##  [1]  2.0507247 -0.2445187 -0.1397217 -1.3241197  1.5056321  4.5602138
##  [7]  2.9616545  0.8553803  2.8535385  0.7593313
```

```r
##########################################################
##### Commands to generate random numbers, obtain pdfs
##### cdf and its inverse for the most popular models:
#
# Commands rnorm(),  dnorm(),  pnorm(),  qnorm(),
#          rexp(),   dexp(),   pexp(),   qexp(),
#          runif(),  dunif(),  punif(),  qunif(),
#          rbinom(), dbinom(), pbinom(), qbinom(),
#          rpois(),  dpois(),  ppois(),  qpois(),
#          rgamma(), dgamma(), pgamma(), qgamma(),
#########################################################
layout(matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), 3, byrow=T))

s <- seq(-2, 2, by=0.01)
plot(s, dunif(s, 0, 1), main='Uniform Unif(0,1)', type='l', ylim=c(0, 1))
plot(s, dexp(s, 1), main='Exponential Exp(1)', type='l', ylim=c(0, 1))
plot(s, dnorm(s, 0, 1), main='Gaussian N(0,1)', type='l', ylim=c(0, 1))
plot(s, punif(s, 0, 1), main='Uniform Unif(0,1)', type='l', ylim=c(0, 1))
plot(s, pexp(s, 1), main='Exponential Exp(1)', type='l', ylim=c(0, 1))
plot(s, pnorm(s, 0, 1), main='Gaussian N(0,1)', type='l', ylim=c(0, 1))

w <- seq(0, 1, by=0.01)
plot(w, qunif(w, 0, 1), main='Uniform Unif(0,1)', type='l')
plot(w, qexp(w, 1),    main='Exponential Exp(1)', type='l')
plot(w, qnorm(w, 0, 1), main='Gaussian N(0,1)', type='l')
```



3 Lab 03 - PCA

3.1 PCA of simulated normal data

```
###--------------------###
### LAB 3 (13/03/2023) ###
###--------------------###

### TOPIC:
### Principal Component Analysis
#_------------------------------------------------------------------------------


##### Principal component analysis of a simulated dataset
# Generate the data
library(mvtnorm)

mu  =  c(1, 2)
sig =  cbind(c(1, 1), c(1, 4))
n   =  100

set.seed(13032023)
X = rmvnorm(n, mu, sig)

# we plot the data
plot(X, asp=1, xlab='Var 1', ylab='Var 2', pch=19, xlim=c(-6,6), ylim=c(-5,10))

# we plot the sample mean
points(colMeans(X)[1], colMeans(X)[2], col='red', pch=19, lwd=3)

# we plot the projection on the x- an y-axis and compute their variance
abline(h=colMeans(X)[2], lty=2, col='grey')
points(X[,1], rep(colMeans(X)[2], n), col='red')
var(X[,1])
```

```
## [1] 1.088467
```

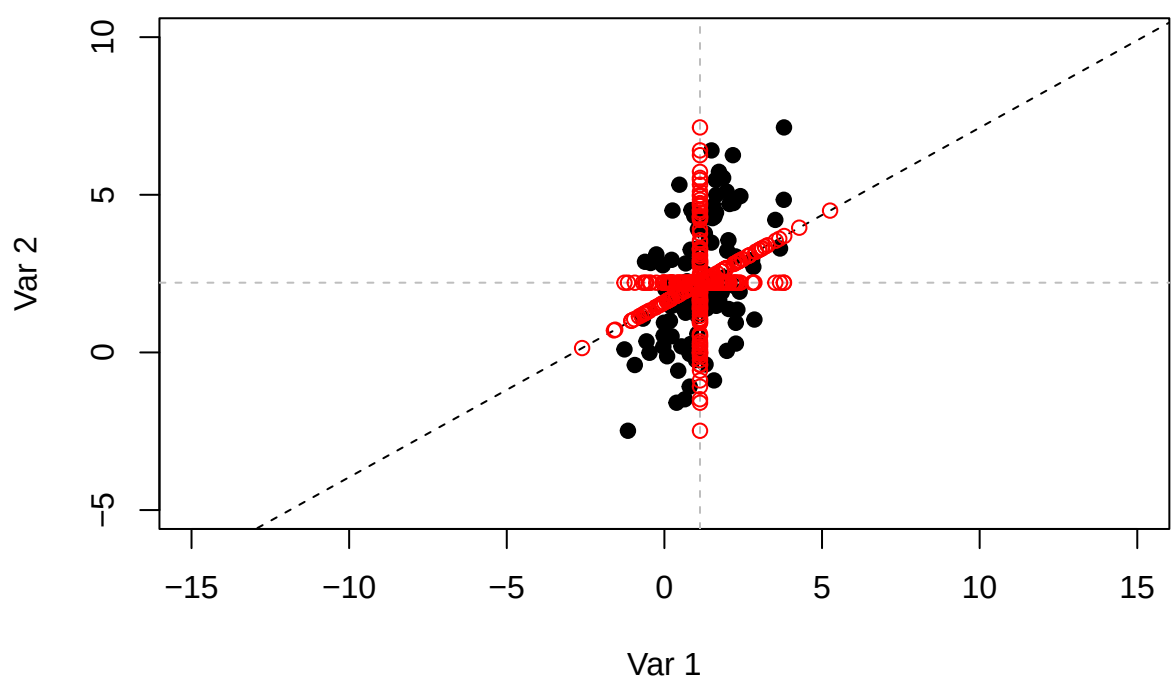
```r
abline(v=colMeans(X)[1], lty=2, col='grey')
points(rep(colMeans(X)[1], n), X[,2], col='red')
var(X[,2])
```

```
## [1] 3.866652
```

```r
# let's compute the variance along all the directions
theta   = seq(0, pi, by=2*pi/360)
Var     = NULL

# Example: along the direction with angle theta:
theta[30]
```

```
## [1] 0.5061455
```

```r
abline(a = colMeans(X)[2] - tan(theta[30])*colMeans(X)[1], b = tan(theta[30]), lty=2)
a = c(cos(theta[30]), sin(theta[30]))
proj30 = a %*% (t(X)-colMeans(X))
points(colMeans(X)[1]+cos(theta[30])*proj30,
       colMeans(X)[2]+sin(theta[30])*proj30, col='red')
```

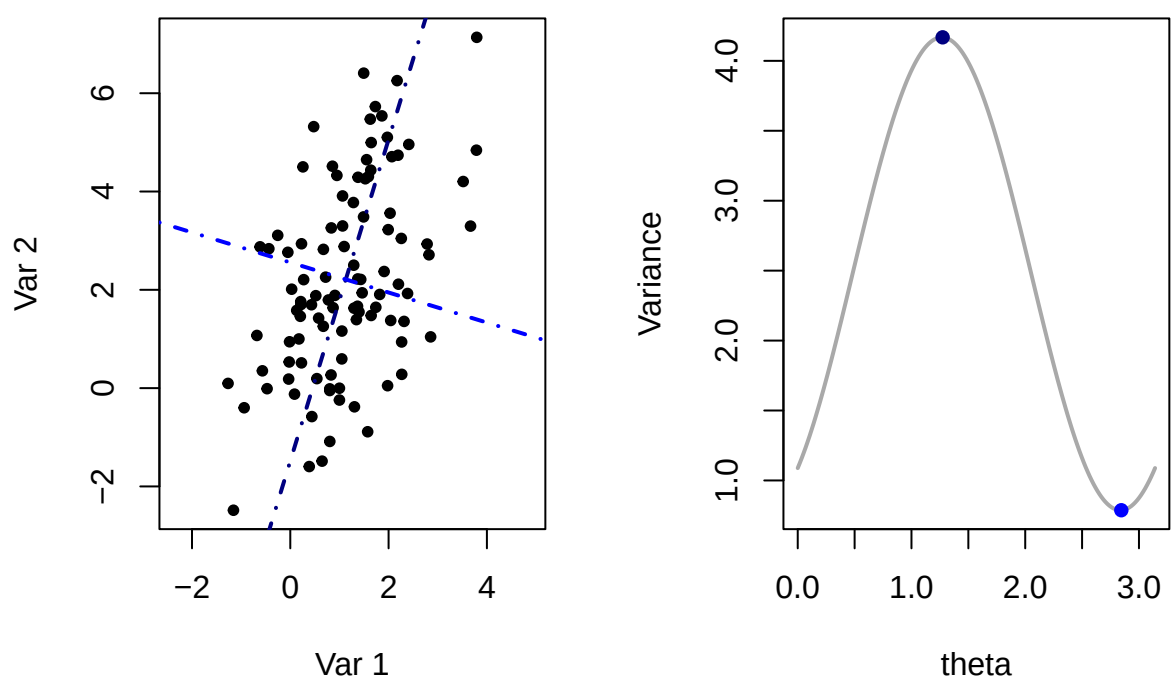



```r
var(X %*% a)
```

```
##          [,1]
## [1,] 2.558756
```

```r
# For cycle
for(i in 1:length(theta))  # for i between 1 and length(theta) repeat:
{
  a   = c(cos(theta[i]), sin(theta[i])) # unit vector in direction theta[i]
  v   = var(X %*% a) # sample variance of the projection of X along the direction identified by vector a
  Var = c(Var, v)
}

# we can compute the min and max of the variance
max.var   = max(Var)
```

```r
max.theta = theta[which.max(Var)]
max.a     = c(cos(max.theta), sin(max.theta))

min.var   = min(Var)
min.theta = theta[which.min(Var)]
min.a     = c(cos(min.theta), sin(min.theta))

# Graphically:
par(mfrow=c(1,2))
plot(X, asp=1, xlab='Var 1', ylab='Var 2', pch=20)
abline(a = colMeans(X)[2] - tan(max.theta)*colMeans(X)[1], b = tan(max.theta),
       lty=4, col='navyblue', lwd=2)
abline(a = colMeans(X)[2] - tan(min.theta)*colMeans(X)[1], b = tan(min.theta),
       lty=4, col='blue', lwd=2)

plot(theta, Var, type = 'l', col='dark grey', lwd = 2,ylab='Variance')
points(max.theta, max.var, pch=16, col='navyblue')
points(min.theta, min.var, pch=16, col='blue')
```




```r
### Let's verify the theory
# we compute the sample covariance matrix
M = colMeans(X)
S = cov(X)

# we compute the eigenvectors and eigenvalues
eigen(S)
```

```
## eigen() decomposition
## $values
## [1] 4.1682352 0.7868833
##
## $vectors
##           [,1]       [,2]
## [1,] 0.2986474 -0.9543635
## [2,] 0.9543635  0.2986474
```

```r
# Note. eigen(S)$vectors returns a matrix whose
#     columns are the eigenvectors of S

# we compare with the values and directions of max/min variability
# we found empirically
max.var
```

```
## [1] 4.168089
```

```r
min.var
```

```
## [1] 0.7870293
```

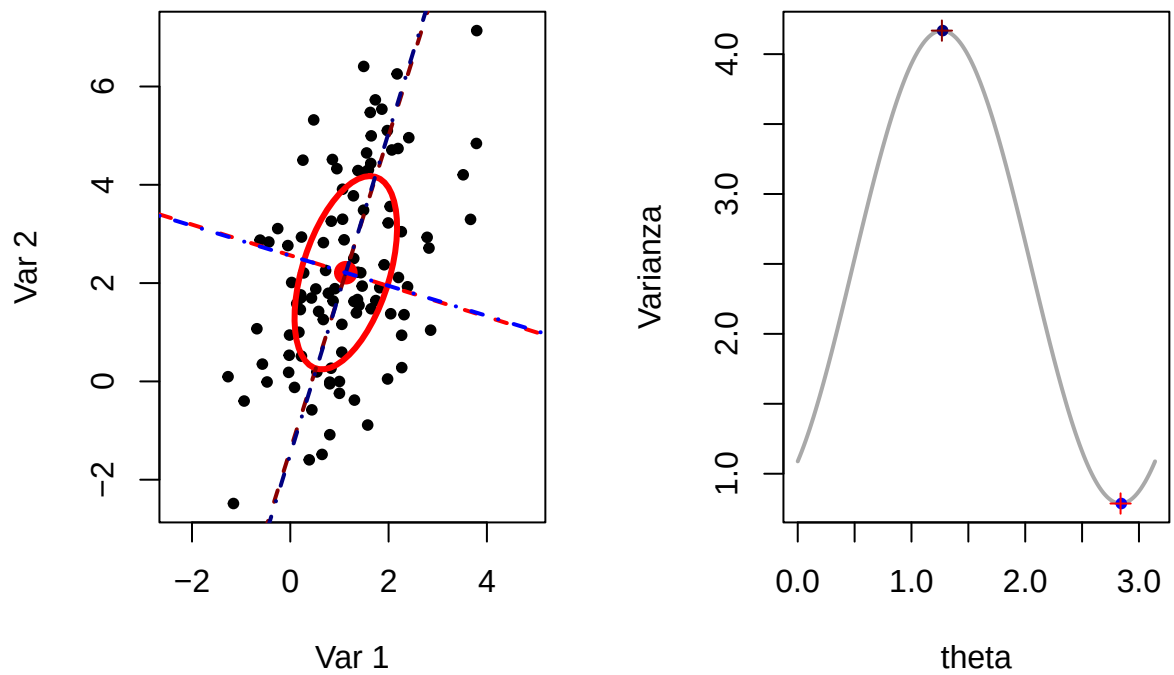
```r
max.a
```

```
## [1] 0.2923717 0.9563048
```

```r
min.a
```

```
## [1] -0.9563048  0.2923717
```

```r
# let's plot the directions of max/min variability
par(mfrow=c(1,2))
plot(X, asp=1, xlab='Var 1', ylab='Var 2', pch=20)
library(car)
ellipse(M, S, 1, add=T, lwd=3, col='red')
abline(a = M[2] - eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1]*M[1],b =
        eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1], lty = 2, col = 'dark red', lwd = 2)
abline(a = M[2] - eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2]*M[1], b
       = eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2], lty = 2, col = 'red', lwd = 2)
abline(a = M[2] - tan(max.theta)*M[1], b = tan(max.theta), lty = 4,
       col = 'navyblue', lwd = 2)
abline(a = M[2] - tan(min.theta)*M[1], b = tan(min.theta), lty = 4,
       col = 'blue', lwd = 2)

plot(theta, Var, type = 'l', col='dark grey', lwd = 2,ylab='Varianza')
points(max.theta, max.var, pch=20, col='navyblue')
points(min.theta, min.var, pch=20, col='blue')
points(atan(eigen(S)$vector[2,1]/eigen(S)$vector[1,1]), max.var, pch=3,
       col='dark red')
points(atan(eigen(S)$vector[2,2]/eigen(S)$vector[1,2])+pi, min.var, pch=3,
       col='red')
```



3.2 PCA on dataset tourists

```
##### Principal component analysis of the dataset 'tourists'

# The dataset "tourists.txt" collects the data on the flow of Italian tourism
# from outside Lombardy to Milan for the year 2015. Each statistical unit
# corresponds to a Region of origin and a month of observation. For each unit,
# the tourists' flow is quantified through the number of nights spent by clients
# in: `5 stars hotels', `4 stars hotels', `3 stars hotels', `2 stars hotels',
# `1 star hotels', `residences', `B&B' and `rented flats'.

tourists = read.table('Lab03/tourists.txt', header=T)
head(tourists)
```

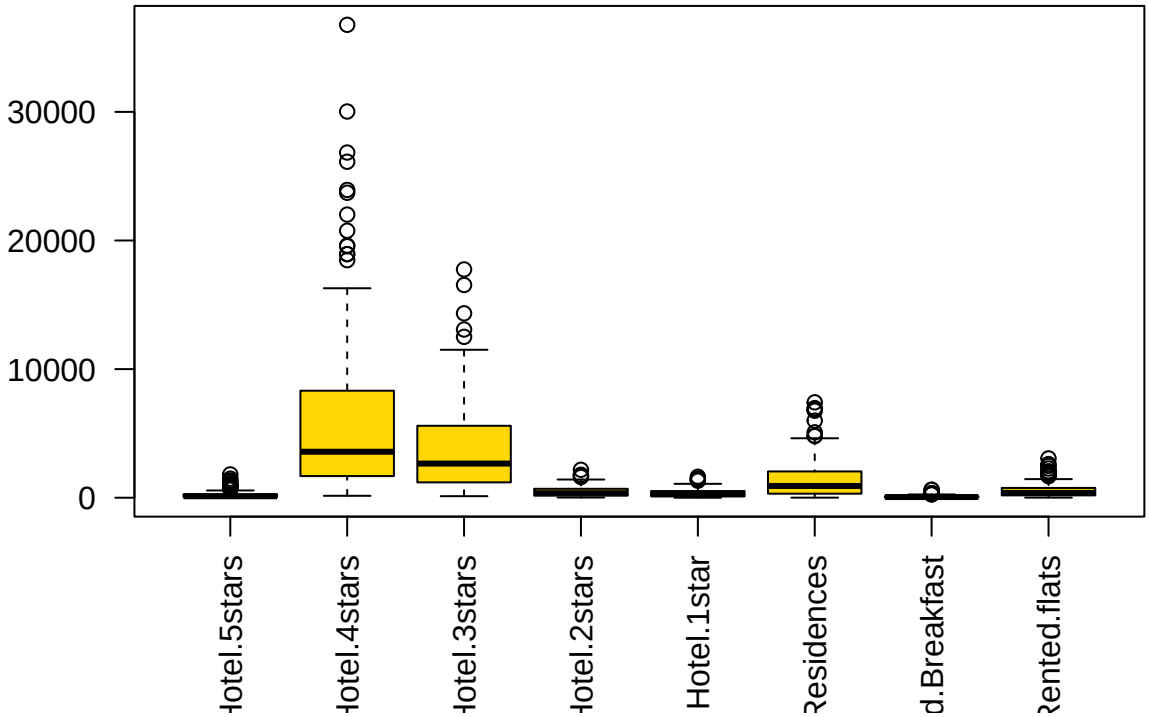
```
##   Month Region.of.origin Hotel.5stars Hotel.4stars Hotel.3stars Hotel.2stars
## 1   Jan         PIEMONTE          255         5733         3878          351
## 2   Feb         PIEMONTE          277         6613         3816          352
## 3   Mar         PIEMONTE          272         7278         4518          523
## 4   Apr         PIEMONTE          181         5311         3550          383
## 5   May         PIEMONTE          301         5885         3231          396
## 6   Jun         PIEMONTE          259         5436         3115          456
##   Hotel.1star Residences Bed.Breakfast Rented.flats
## 1         504       1489            41          551
## 2         380       1617            31          415
## 3         475       1949            42          520
## 4         330       1853            42          542
## 5         421       2050            85          561
## 6         301       2273            64          587
```

```
dim(tourists)
```

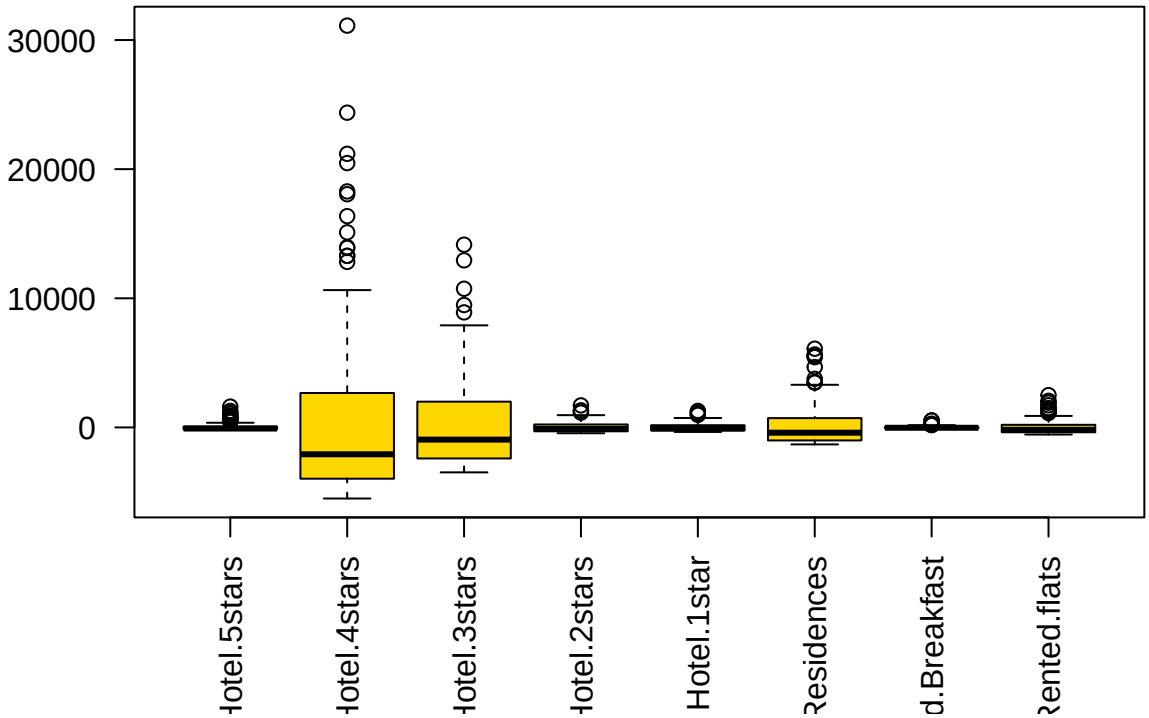
```
## [1] 240  10
```

```
tourists.label = tourists[, 1:2]
tourists = tourists[, -(1:2)]
n = dim(tourists)[1]
p = dim(tourists)[2]
```

```r
# Boxplot
# par(mar=rep(8,4))
boxplot(tourists, las=2, col='gold')
```




```r
# We observe that the variability of the number of nights in 3,4 stars hotels and
# residences is higher than that of the others. This may influence the PCA

# Note: PCA is not about the mean, it is about the variability

# par(mar=rep(8,4))
boxplot(scale(x=tourists, center=T, scale=F), las=2, col='gold')
```




```
#! here center and scale operate respectevely on mean and variance
```

3.2.1 PCA on original data

```
# We perform the PCA on original data
#! PCA is just influenced by the variance between the variables, not their mean
pc.tourists = princomp(tourists, scores=T)
#! there are also better commands for very large datasets

pc.tourists
```

```
## Call:
## princomp(x = tourists, scores = T)
##
## Standard deviations:
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6     Comp.7
## 6573.01508  824.72074  364.23534  162.54342  117.61268   64.14162   61.48464
##     Comp.8
##   34.06981
##
##  8  variables and  240 observations.
```

```
summary(pc.tourists)
```

```
## Importance of components:
##                             Comp.1       Comp.2       Comp.3       Comp.4
## Standard deviation     6573.0150832 824.72074094 3.642353e+02 1.625434e+02
## Proportion of Variance    0.9804356   0.01543489 3.010606e-03 5.995544e-04
## Cumulative Proportion     0.9804356   0.99587044 9.988811e-01 9.994806e-01
##                             Comp.5       Comp.6       Comp.7       Comp.8
## Standard deviation     1.176127e+02 6.414162e+01 6.148464e+01 3.406981e+01
## Proportion of Variance 3.139049e-04 9.336190e-05 8.578730e-05 2.634082e-05
## Cumulative Proportion  9.997945e-01 9.998879e-01 9.999737e-01 1.000000e+00
```

```
# To obtain the rows of the summary:
# standard deviation
pc.tourists$sd
```

```
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6     Comp.7
## 6573.01508  824.72074  364.23534  162.54342  117.61268   64.14162   61.48464
##     Comp.8
##   34.06981
```

```r
# proportion of variance explained by each PC
#! so how much variability each PC captures

pc.tourists$sd^2/sum(pc.tourists$sd^2)
```

```
##       Comp.1       Comp.2       Comp.3       Comp.4       Comp.5       Comp.6
## 9.804356e-01 1.543489e-02 3.010606e-03 5.995544e-04 3.139049e-04 9.336190e-05
##       Comp.7       Comp.8
## 8.578730e-05 2.634082e-05
```

```r
# cumulative proportion of explained variance
cumsum(pc.tourists$sd^2)/sum(pc.tourists$sd^2)
```

```
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8
## 0.9804356 0.9958704 0.9988811 0.9994806 0.9997945 0.9998879 0.9999737 1.0000000
```

3.2.2 Loadings

```r
# loadings (recall: coefficients of the linear combination of the original
#           variables that defines each principal component)
load.tour = pc.tourists$loadings
load.tour
```

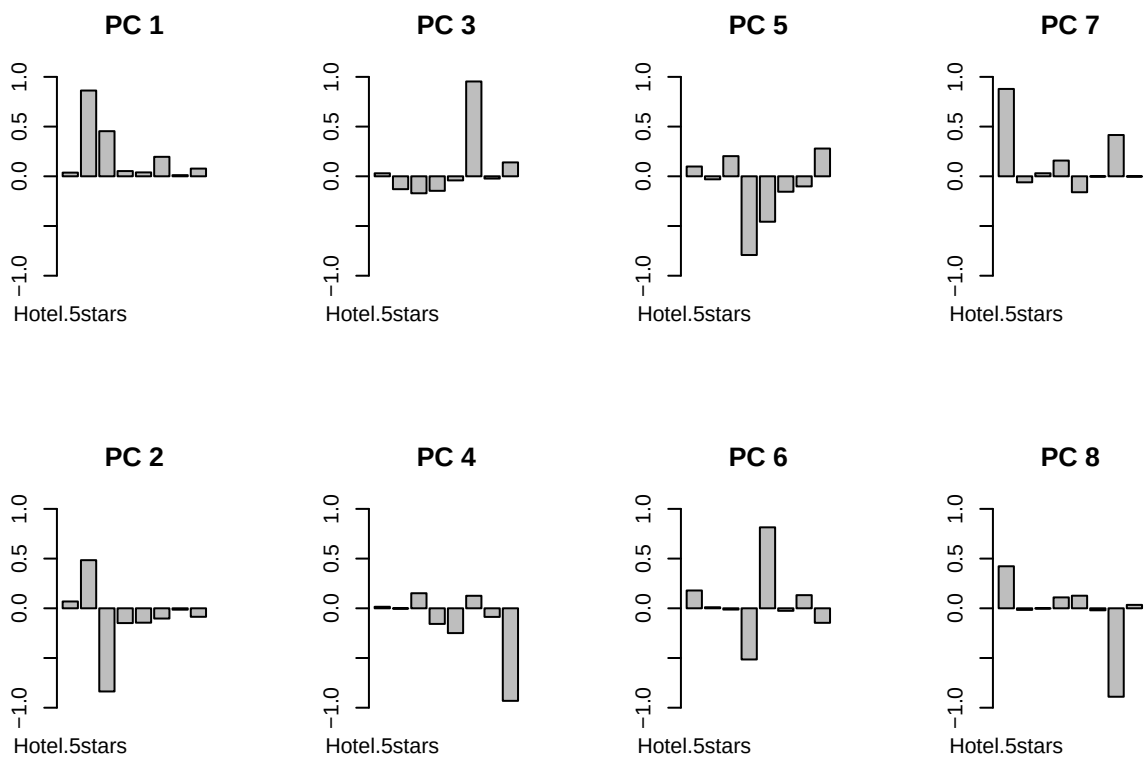
```
##
## Loadings:
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## Hotel.5stars                                       0.179  0.879  0.423
## Hotel.4stars    0.863  0.484 -0.130
## Hotel.3stars    0.454 -0.837 -0.171  0.152  0.203
## Hotel.2stars         -0.149 -0.146 -0.157 -0.792 -0.515  0.158  0.110
## Hotel.1star          -0.145        -0.249 -0.457  0.814 -0.160  0.127
## Residences      0.196 -0.103  0.954  0.126 -0.155
## Bed.Breakfast                             -0.102  0.132  0.415 -0.890
## Rented.flats                  0.140 -0.931  0.279 -0.146
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
```

```r
#! Loadgins show how the PCs are built as linear combinations of the
#! original variables

#! Nel dataset c'erano due variabili ad alta varianza, ma nel PCA solo una sembra
#! importante => quindi quelle due erano molto correlate, "ne basta una"

load.tour[,1:8]
```

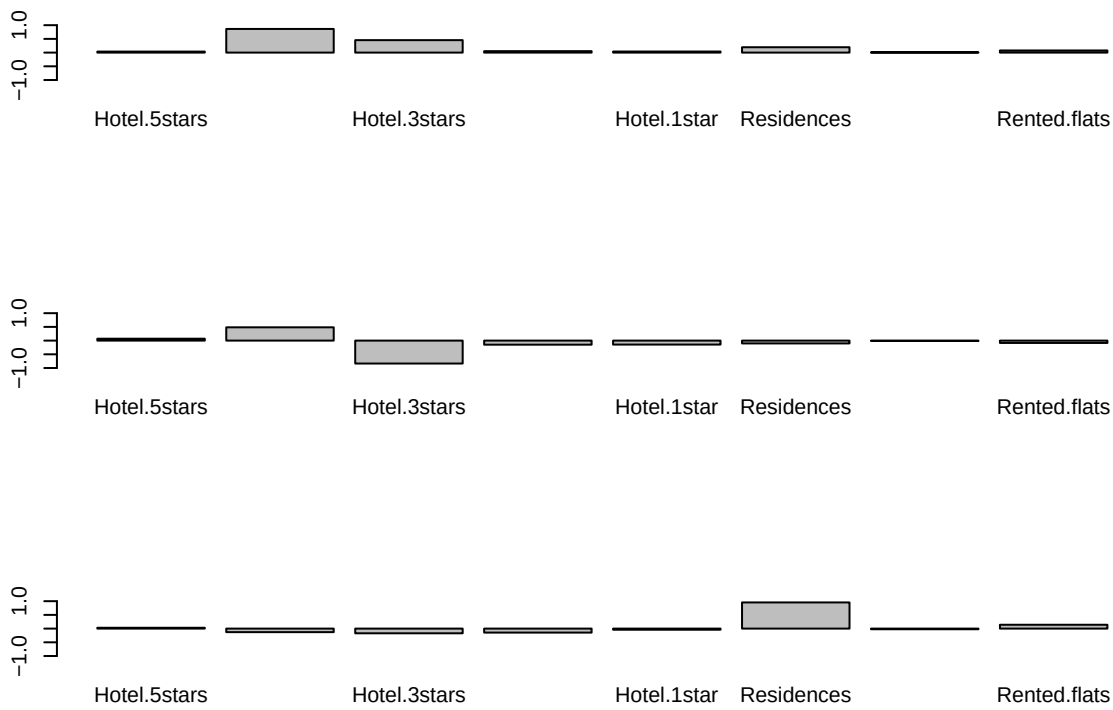
```
##                   Comp.1      Comp.2      Comp.3       Comp.4      Comp.5
## Hotel.5stars  0.03698800  0.06875905  0.03032049  0.015383216  0.09843182
## Hotel.4stars  0.86250762  0.48404327 -0.12963676 -0.003314527 -0.03025384
## Hotel.3stars  0.45374825 -0.83652698 -0.17078613  0.151654002  0.20265148
## Hotel.2stars  0.05252810 -0.14868959 -0.14612870 -0.157100320 -0.79181454
## Hotel.1star   0.03921975 -0.14477781 -0.04127983 -0.249443398 -0.45701223
## Residences    0.19594019 -0.10300007  0.95393089  0.125697570 -0.15539773
## Bed.Breakfast 0.01153809 -0.01380745 -0.02285702 -0.086555157 -0.10166414
## Rented.flats  0.07747214 -0.08560395  0.13979581 -0.930891638  0.27930944
##                    Comp.6       Comp.7      Comp.8
## Hotel.5stars   0.17926970  0.878775217  0.42270057
## Hotel.4stars   0.01048520 -0.060705523 -0.01598857
## Hotel.3stars  -0.01267825  0.030573859 -0.00227378
## Hotel.2stars  -0.51485280  0.158149219  0.10974173
## Hotel.1star    0.81423507 -0.160340573  0.12659831
## Residences    -0.02558117 -0.003028534 -0.02005940
## Bed.Breakfast  0.13233387  0.415205521 -0.88961706
## Rented.flats  -0.14619468 -0.003020767  0.03423665
```

```r
#! This matrix shows how each PC is obtained, as a wheighted average of the
#! original cols/variables
```

```r
# graphical representation of the loadings of the first six principal components
par(mfcol=c(2,4))
for(i in 1:8) barplot(load.tour[,i], ylim = c(-1, 1), main=paste("PC",i))
```



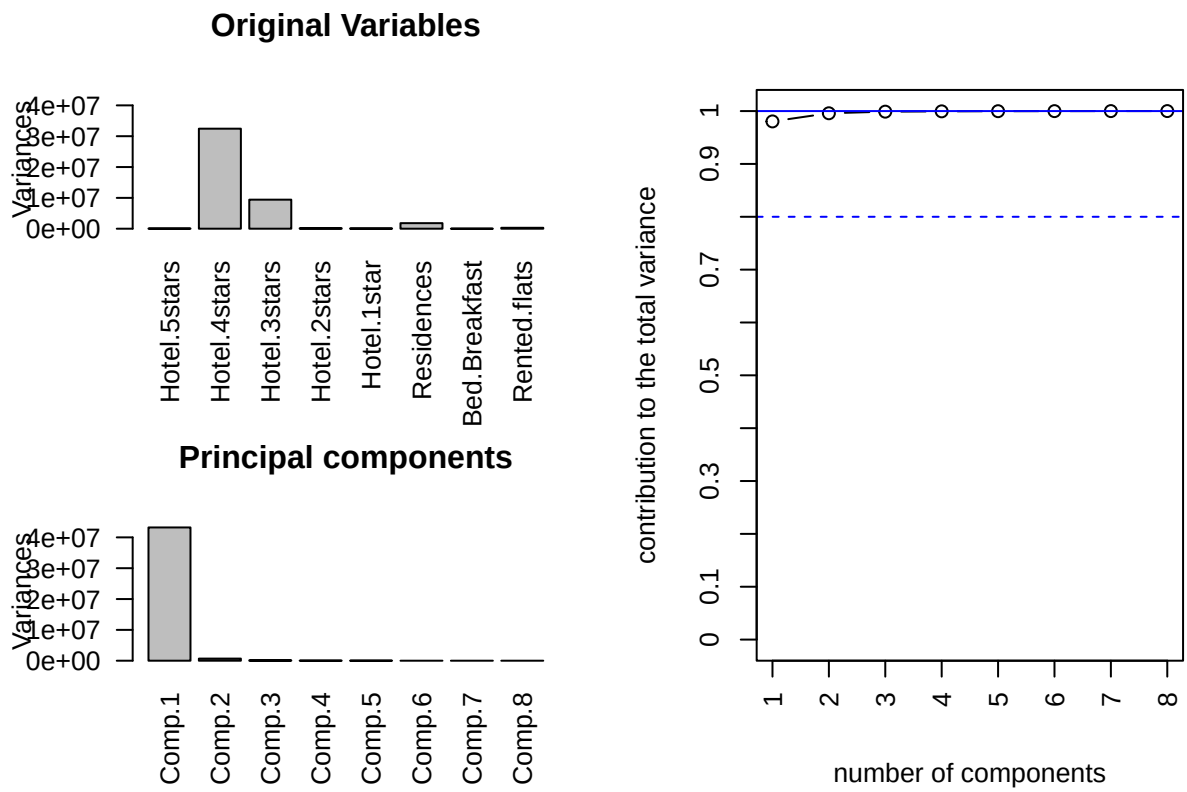

```r
par(mfrow = c(3,1))
for(i in 1:3) barplot(load.tour[,i], ylim = c(-1, 1))
```




```
# Interpretation of the loadings:
# First PCs: weighted average of the number of nights in 3,4 stars hotel and residences
# Second PCs: contrast between the number of nights in 3 and 4 stars hotel
# Third PC: residences

# The loadings reflect the previous observation: the first 3 PCs are
# driven by the variables displaying the highest variability
```

3.2.3 Explained variance

```
# Explained variance
layout(matrix(c(2,3,1,3),2,byrow=T))
plot(pc.tourists, las=2, main='Principal components', ylim=c(0,4.5e7))
barplot(sapply(tourists,sd)^2, las=2, main='Original Variables',
        ylim=c(0,4.5e7), ylab='Variances')
plot(cumsum(pc.tourists$sd^2)/sum(pc.tourists$sd^2), type='b', axes=F,
     xlab='number of components',
     ylab='contribution to the total variance', ylim=c(0,1))
abline(h=1, col='blue')
abline(h=0.8, lty=2, col='blue')
box()
axis(2,at=0:10/10,labels=0:10/10)
axis(1,at=1:ncol(tourists),labels=1:ncol(tourists),las=2)
```



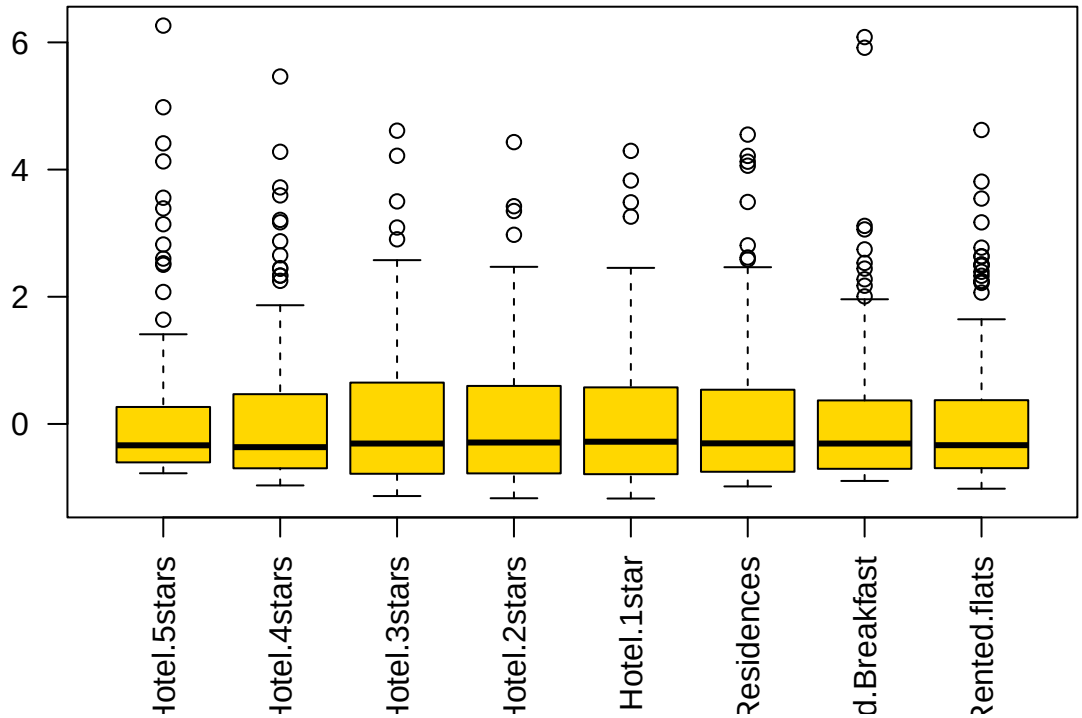

```
# The first PC explains more than 98% of the total variability.
# This is due to the masking effect of those 3 variables over the others
```

3.2.4 PCA on std variables

```
##### Principal component analysis of the dataset 'tourists',
##### but on the standardized variables

# We compute the standardized variables
#! Standardizing makes sense to remove the effect of certain variables having
# higher values
tourists.sd = scale(tourists)
tourists.sd = data.frame(tourists.sd)

head(tourists.sd)
```

```
##   Hotel.5stars Hotel.4stars Hotel.3stars Hotel.2stars Hotel.1star Residences
## 1   0.20868970   0.01397475   0.09114066  -0.29293603   0.5201322  0.1247660
## 2   0.29424616   0.16848570   0.07093516  -0.29033860   0.1037466  0.2203068
## 3   0.27480151   0.28524681   0.29971358   0.15382199   0.4227517  0.4681159
## 4  -0.07909111  -0.06012027  -0.01575296  -0.20981826  -0.0641508  0.3964602
## 5   0.38758048   0.04066301  -0.11971352  -0.17605167   0.2414225  0.5435035
## 6   0.22424542  -0.03817269  -0.15751736  -0.02020584  -0.1615313  0.7099536
##   Bed.Breakfast Rented.flats
## 1   -0.43655633 -0.011801778
## 2   -0.54858961 -0.263408587
## 3   -0.42535300 -0.069153330
## 4   -0.42535300 -0.028452228
## 5    0.05639008  0.006698723
## 6   -0.17887980  0.054800025
```

```
# Boxplot
# par(mar=rep(8,4))
boxplot(tourists.sd, las=2, col='gold')
```




```
pc.tourists = princomp(tourists.sd, scores=T)
pc.tourists
```

```
## Call:
## princomp(x = tourists.sd, scores = T)
##
## Standard deviations:
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8
## 2.6718997 0.6343339 0.4643065 0.3037082 0.2224482 0.1910011 0.1430582 0.1048307
##
##  8  variables and  240 observations.
```

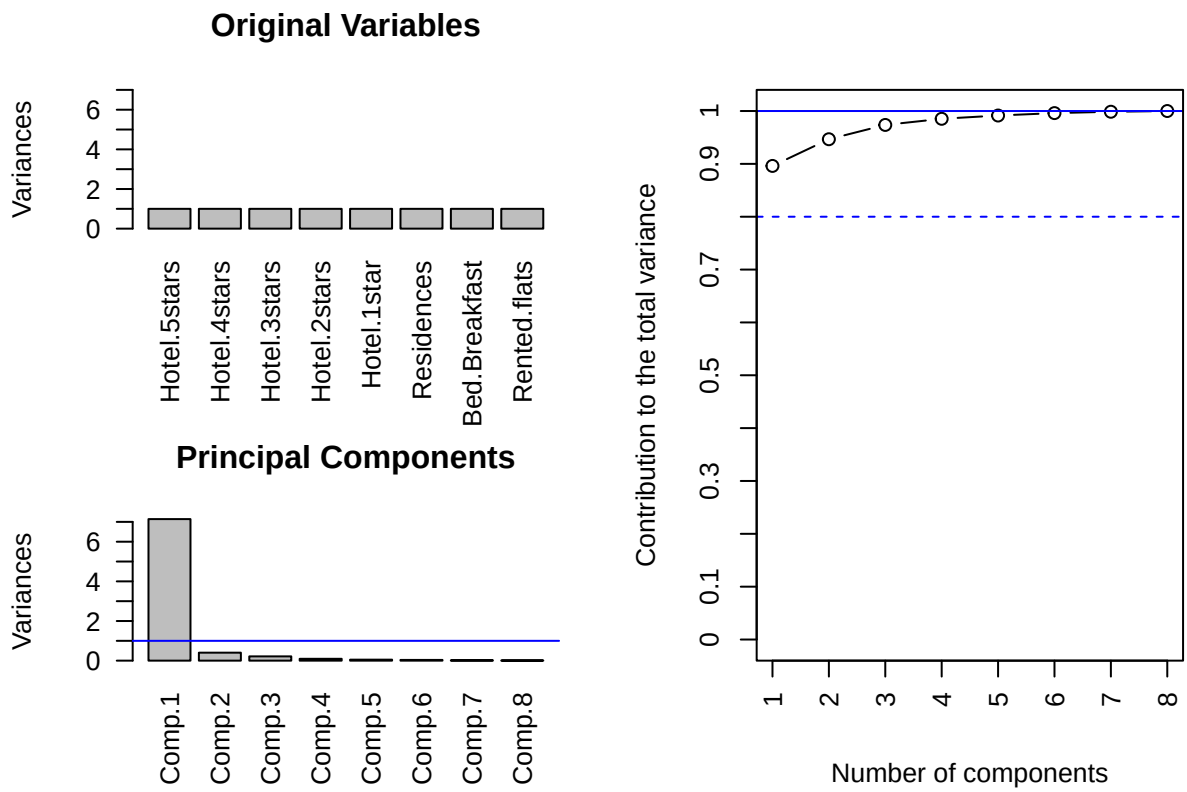
```
summary(pc.tourists)
```

```
## Importance of components:
##                           Comp.1     Comp.2     Comp.3     Comp.4      Comp.5
## Standard deviation     2.6718997 0.63433394 0.46430646 0.30370821 0.222448208
## Proportion of Variance 0.8961148 0.05050789 0.02706031 0.01157808 0.006211281
## Cumulative Proportion  0.8961148 0.94662273 0.97368304 0.98526112 0.991472400
##                             Comp.6      Comp.7      Comp.8
## Standard deviation     0.191001055 0.143058239 0.104830741
## Proportion of Variance 0.004579256 0.002568911 0.001379433
## Cumulative Proportion  0.996051656 0.998620567 1.000000000
```

3.2.5 Explained variance

```
# Explained variance
layout(matrix(c(2, 3, 1, 3), 2, byrow=T))
plot(pc.tourists, las=2, main='Principal Components', ylim=c(0,7))
abline(h=1, col='blue')
barplot(sapply(tourists.sd,sd)^2, las=2, main='Original Variables',
        ylim=c(0,7), ylab='Variances')
plot(cumsum(pc.tourists$sde^2)/sum(pc.tourists$sde^2), type='b', axes=F,
     xlab='Number of components', ylab='Contribution to the total variance',
     ylim=c(0,1))
abline(h=1, col='blue')
abline(h=0.8, lty=2, col='blue')
```

```r
box()
axis(2,at=0:10/10,labels=0:10/10)
axis(1,at=1:ncol(tourists.sd),labels=1:ncol(tourists.sd),las=2)
```



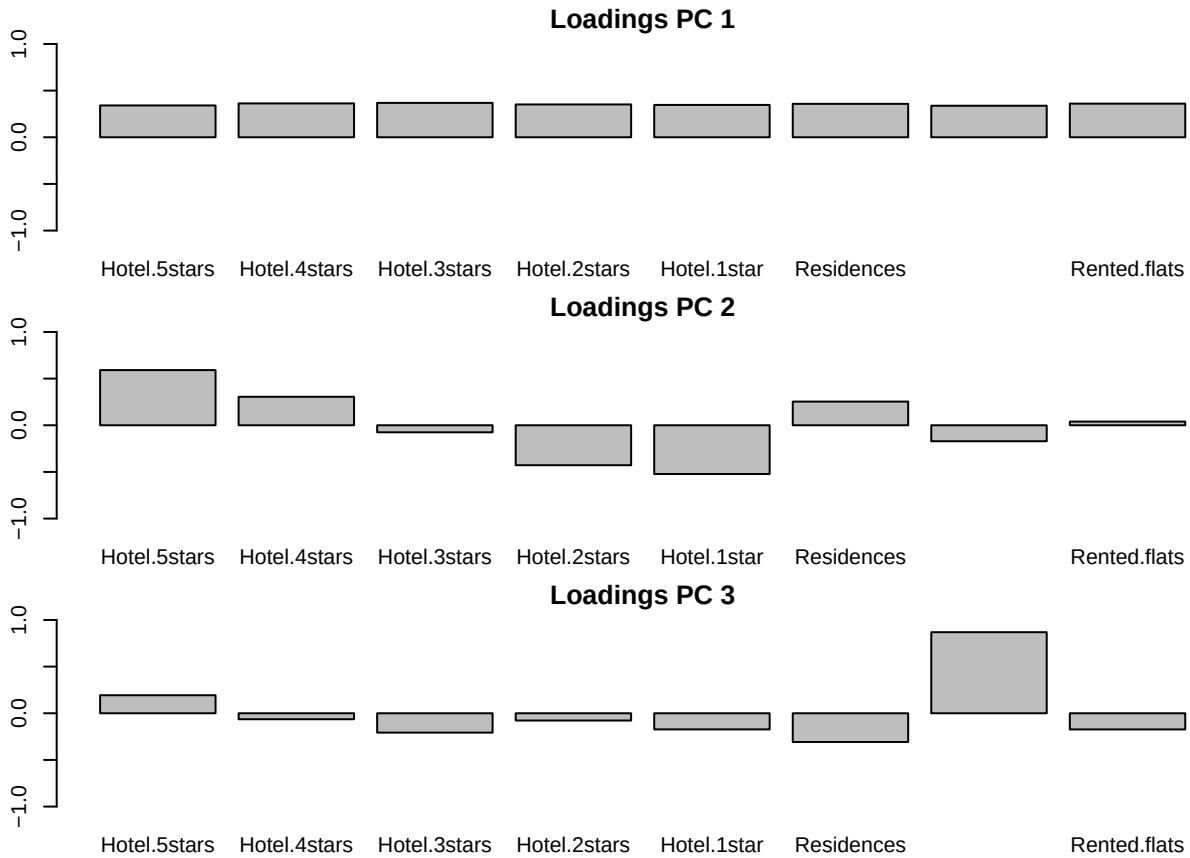

```r
# If we wanted to perform dimensionality reduction, we could keep
# 1 or 2 PCs
```

3.2.6 Loadings

```r
# loadings
load.tour = pc.tourists$loadings
load.tour
```

```
##
## Loadings:
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## Hotel.5stars    0.341  0.591  0.193  0.269  0.147  0.402  0.324  0.369
## Hotel.4stars    0.363  0.305         0.271  0.163        -0.345 -0.743
## Hotel.3stars    0.368        -0.207  0.159        -0.105 -0.699  0.533
## Hotel.2stars    0.352 -0.428         0.502  0.227 -0.437  0.437
## Hotel.1star     0.347 -0.523 -0.173        -0.152  0.724        -0.134
## Residences      0.358  0.254 -0.308 -0.237 -0.708 -0.276  0.274
## Bed.Breakfast   0.338 -0.171  0.869 -0.194 -0.185 -0.135
## Rented.flats    0.360        -0.173 -0.691  0.579 -0.135
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
```

```r
par(mar = c(2,2,2,1), mfrow=c(3,1))
for(i in 1:3)barplot(load.tour[,i], ylim = c(-1, 1),
                     main=paste('Loadings PC ',i,sep=''))
```




```
# Interpretation of the loadings:
# In this case, the first PC represents an average of the number of nights
# spent in all the types of hotels and residences, taken with very similar weights.
# The second PC contrasts the more expensive solutions (4,5 stars hotels and
# residences) against the cheap solutions (1,2 stars hotels and B&B)

#! PC1: almost a simple average of all the variables. So high PC1 will mean high
#! flow of tourists. Where high/large refers to how far (how many std
#! PC2: needs to be a contrast (as PC1 was a mean and directions must be orthogonal)
#! Has a plus sign for 5/4 stars and residence. So high PC2 will mean a
#! preferred/frequent choice for expensive places
#! Use then higher PCs only to discriminate between similar of-the-previous-PCs cases

# High PC1: general high flow of tourists
# Low PC1: general low flow of tourists
# High PC2: high flow for expensive solutions, low flow for cheap solutions
# Low PC2: low flow for expensive solutions, high flow for cheap solutions
```

3.2.7 Scores

```
# scores
scores.tourists = pc.tourists$scores
scores.tourists
```

```
##          Comp.1        Comp.2       Comp.3        Comp.4        Comp.5
## 1    0.07995640  0.0803750037 -0.462324880 -0.0490572579 -1.338738e-01
## 2   -0.08002789  0.4295531054 -0.462937629  0.2184115049 -2.233720e-01
## 3    0.50727996  0.1290731374 -0.614070772  0.2634197670 -2.587780e-01
## 4   -0.16271958  0.2317602688 -0.467514012 -0.1324662019 -2.764895e-01
## 5    0.34064659  0.3283221488 -0.050492103 -0.1547718869 -3.946383e-01
## 6    0.15495852  0.4388325405 -0.275618617 -0.1443574158 -3.784939e-01
## 7   -0.12411897  0.5125701490 -0.509195094 -0.0772748713 -6.620485e-01
## 8   -0.99150260  0.4715920287 -0.501324952 -0.0219038873 -7.996164e-01
## 9    0.45385670  1.0969291745 -0.276681608  0.2902047245 -4.173239e-01
## 10   0.61297730  0.8821535876  0.018901636  0.1405559668 -4.701499e-01
## 11   0.55414143  0.4683274744 -0.088511606  0.4828927685 -1.368862e-01
## 12   0.15721633  0.7853423293 -0.551532124  0.2410497130 -5.860563e-02
## 13  -2.68382753  0.1678392171  0.096556106 -0.0722437539 -8.974784e-02
```

```
## 14  -2.71643588  0.1748588984  0.100927078 -0.0275436144 -5.047977e-02
## 15  -2.70440951  0.2607113634  0.198480813 -0.0402372060 -8.073882e-03
## 16  -2.64808223  0.1013007006  0.081341002  0.0460107855  4.249362e-02
## 17  -2.74811060  0.2298225349  0.115335369  0.0265642512  5.119229e-03
## 18  -2.76470519  0.2278929438  0.107672382  0.0497644398  2.354711e-02
## 19  -2.78344666  0.2662156053  0.108068665 -0.0313727016  1.050348e-02
## 20  -2.82623112  0.2520612016  0.118129443 -0.0549324174 -7.995835e-03
## 21  -2.70299006  0.1931759019  0.120658975 -0.0062229592  6.508170e-03
## 22  -2.61021730  0.3519231940  0.178828474  0.0773660857  9.533317e-02
## 23  -2.79971496  0.2655391831  0.163474344 -0.0220759785  1.764674e-03
## 24  -2.83546903  0.3275736997  0.158696331 -0.0369951727  1.132937e-02
## 25   1.48472985 -0.1602265214 -0.674999744  0.2378512263  5.141582e-01
## 26   2.18970467 -0.4855139034 -0.768892133  0.4042210480  4.254708e-01
## 27   2.41299689 -0.3443833051 -0.301407455  0.3766142520  4.047832e-01
## 28   1.50795150 -0.2389860406 -0.600419775  0.1616029453 -2.323965e-02
## 29   2.63538085 -0.0801380120  0.110234656  0.2388585477  1.313136e-01
## 30   1.99475189 -0.0650947203 -0.180141940  0.4034721799  2.515025e-01
## 31   1.56400353  0.0520543020  0.137270417  0.0801651771  1.891490e-02
## 32   1.22496997 -0.2030321010 -0.272758975  0.3396327370  1.034736e-01
## 33   4.15576759 -0.5806193653  0.714036719  0.8602934040  3.170536e-01
## 34   4.54399256  0.1343036366  1.281281004  0.3516004185  4.784555e-01
## 35   1.71146442  0.5796636726  0.235196275  0.2748625923  1.276056e-01
## 36   0.75718165  0.9859008173 -0.458467691  0.4480466297  4.769059e-01
## 37  -1.64936345  0.2408838821  0.032928462  0.0033578261 -1.213737e-01
## 38  -1.63029775  0.1812709064  0.114031269  0.0583652729 -1.591251e-01
## 39  -1.33671482  0.2079872189  0.337746935  0.0464125721 -1.605010e-01
## 40  -1.18949600  0.1839204200  0.500999124 -0.0236537976 -2.239555e-01
## 41  -1.26520536  0.3839294848  0.032500986  0.0270870274 -1.662628e-01
## 42  -1.10550841  0.1861956038  0.516671888  0.0412546504 -2.850963e-01
## 43  -1.33525797  0.1841903867  0.378271192 -0.0849122853 -1.526128e-01
## 44  -1.25584819 -0.1289983028 -0.010165033  0.0588552815 -1.528515e-01
## 45  -0.49430202 -0.1746513519  0.427346287  0.0513378309 -1.306505e-01
## 46  -0.06538377 -0.1815905726  1.043334090  0.0050272462 -2.114857e-03
## 47  -1.64794688  0.2818991293  0.330772089  0.0233198381 -1.287096e-01
## 48  -1.87291333  0.5224919358  0.080441784  0.0766557182  3.196236e-02
## 49  -1.04696212 -0.0724155424 -0.149025128 -0.1040461580 -1.595505e-01
## 50  -1.14931013 -0.0689006637 -0.078040505 -0.0230596285  3.264381e-02
## 51  -0.82359537 -0.1478170150  0.444735439 -0.1494647038 -1.703677e-01
## 52  -1.21375608  0.0476855615 -0.010417509  0.0510930868 -1.165468e-01
## 53  -0.86620752 -0.0036819871  0.436149294 -0.2036332500 -2.481674e-01
## 54  -1.05163737  0.3166824279  0.081320351  0.0533204987 -5.512081e-02
## 55  -1.42035264  0.1219604496 -0.080087647  0.0191662453 -7.023703e-02
## 56  -1.77500768  0.1578127146 -0.053324804 -0.0107091118 -1.669913e-01
## 57  -0.74378005  0.1340652882  0.513574226  0.1829477386 -2.998406e-02
## 58  -0.57551707  0.2422269126  0.471489604  0.0441046633 -5.523680e-02
## 59  -1.00476857  0.1912458518 -0.014220823  0.1163485295 -2.731783e-01
## 60  -1.27514227  0.3751027707 -0.270887381  0.0835476636  7.991332e-03
## 61   0.37584385  0.0679179204 -0.176153144  0.3446365944 -6.729908e-02
## 62   1.06525546 -0.2872975183 -0.037831786  0.0057446722  3.152182e-02
## 63   1.47753470 -0.2608390278 -0.027298481  0.0173696181  2.458261e-01
## 64   0.73652763 -0.1246203577  0.592685780 -0.2781562538 -2.116078e-01
## 65   1.52751507  0.2205485208  0.020578320  0.0871783028 -3.101780e-01
## 66   1.30935462  0.1638653040  0.549095705  0.2179594477 -1.322902e-01
## 67   0.45976521  0.0648980589  0.329781237  0.2159781737 -2.148012e-01
## 68   0.07366019 -0.3458240758 -0.153778874  0.2496491011 -2.355700e-01
## 69   2.52677741 -0.5417935530  0.700153415  0.3100496248  5.314795e-02
## 70   2.94861913 -0.2196816993  0.930209324  0.4384545542  2.211315e-01
## 71   0.77293774  0.1812136554  0.232143758  0.4009918083  1.520677e-01
## 72   0.14436084  0.6169476441  0.166727907  0.2001042597  1.265625e-01
## 73   1.00896774  0.3841666440  0.218934638  0.4733266000  1.805594e-01
## 74   1.23944301  0.0106066238 -0.421766228  0.5112103749  1.853384e-01
## 75   1.58254386  0.0531773809 -0.280949323  0.7594936367  2.580534e-01
## 76   1.01751894 -0.2188396440  0.282967740  0.2528826658 -2.854490e-02
## 77   1.99936597  0.2170685710 -0.030092777  0.4058229850  1.332560e-01
## 78   1.67208948 -0.2014449123  0.060541783  0.3995361828  1.270653e-02
## 79   0.76957484  0.5780837089 -0.191688945  0.5460102949  3.997043e-01
## 80   0.66460688 -0.5684106116 -0.626128226  0.5768919107  2.320226e-01
## 81   3.60979134 -0.7819807491  0.392127138  0.5925541514  4.843420e-01
## 82   3.88395405 -0.8506749646  0.469434190  0.5981612035  2.808123e-01
## 83   1.06415272  0.1889082679 -0.101204884  0.4549005477 -2.678250e-02
## 84   0.23589180  0.4172924818 -0.268632372  0.2035980425  1.907840e-01
## 85  -1.91798008 -0.0682470671 -0.021040811  0.2232456933 -3.231992e-02
## 86  -1.87734493  0.0020392077  0.083717553  0.1749701710  9.020849e-02
## 87  -1.87777487 -0.0937977383  0.281332770  0.1015168626 -2.076449e-02
## 88  -1.95356683  0.0878557905  0.110813250  0.1177240101  2.086391e-02
```

```
## 89  -1.58921424 -0.0678345718  0.081941785  0.1501150950  3.368113e-03
## 90  -1.52770107 -0.1328288386  0.381652204  0.0122925956  1.523223e-02
## 91  -1.77828933 -0.0958587443  0.158611613  0.1829463297  4.377228e-02
## 92  -1.46530470 -0.2557797310 -0.218455489  0.2786778596  1.489700e-01
## 93  -1.26897639 -0.1609334594 -0.020054942  0.3205539211  2.872290e-02
## 94  -0.73889122 -0.1269026173  0.291621374  0.3149813822  1.303896e-01
## 95  -1.95465956  0.2623840229  0.045050724  0.1847844663  2.355110e-02
## 96  -2.12075398  0.2857628246  0.029553232  0.0604931414  3.484488e-02
## 97  -1.12106677 -0.0754826015 -0.048554473  0.2301479776  1.583450e-02
## 98  -0.62441436 -0.2643513575 -0.020356530  0.1790157405  3.543825e-02
## 99  -0.68004002 -0.3206199110  0.156827981  0.1142770668  9.972869e-02
## 100 -0.77230703 -0.5321524452  0.544765786  0.0755756919  9.578630e-03
## 101 -0.68905434 -0.3680492961  0.561132900  0.0392464357 -9.012264e-03
## 102 -0.94495213 -0.2345992605  0.171153161  0.0931247463  2.085104e-01
## 103 -1.18360240 -0.0378567408  0.115765663  0.2095222778  5.087360e-02
## 104 -0.79163928 -0.3649044663 -0.246277725  0.1994781187  1.548289e-01
## 105  0.86522703 -0.9014503591  0.388487129  0.1994480254  2.480147e-01
## 106  0.80504065 -0.7660095992  1.138937751 -0.0472797062  1.090264e-01
## 107 -0.97729739 -0.1177093060  0.468662016  0.0370593249 -3.443466e-02
## 108 -1.37204285  0.1235686940  0.286368314 -0.0564678163  9.757558e-02
## 109  4.81282490  1.9559132487 -0.264056163 -0.7044890401  3.032554e-01
## 110  5.47186203  1.6470828386 -0.908148000 -0.7376709030  2.742117e-01
## 111  6.35326139  1.4087122801 -1.223878205  0.2412680901  4.793604e-01
## 112  5.55546765  1.5678113057 -0.085756534 -0.5819448365 -2.737489e-01
## 113  9.09634400  2.1187935360 -0.264866827 -0.3906055704 -2.253049e-01
## 114  9.09996827  2.0691453009 -0.120672420  0.0650273090 -4.243087e-01
## 115  6.97332717  2.4366695922  0.051767066 -0.2888272843 -5.068581e-01
## 116  6.05811152 -0.1154111927 -1.688260360 -0.1581908806  7.138017e-02
## 117 12.13665372  0.6669285752  2.122226371 -0.1765052725 -4.887398e-01
## 118 14.21704100  1.1772490487  1.902454629  0.3513525704  1.293498e-01
## 119  5.88618336  2.3370299524  0.420089816 -0.2057584579  3.861699e-01
## 120  4.33853089  3.0034711188 -0.749906694 -0.3516225815  5.470226e-01
## 121 -1.87436096  0.0032579420 -0.102895331 -0.0573296245  1.276794e-01
## 122 -1.62566574  0.0904497417 -0.180818100 -0.1904672531  3.307953e-01
## 123 -1.44739569 -0.0620436174  0.029317280 -0.0008798485  1.702524e-01
## 124 -1.71808483 -0.0786547424  0.026709294 -0.0864544105  1.953743e-02
## 125 -1.33935320  0.0950447474  0.055169469 -0.1216119385  6.910448e-02
## 126 -1.15700785 -0.1878835584  0.245241257 -0.0621395794 -5.196957e-02
## 127 -1.60310264  0.0386242890  0.008526852  0.0709811058 -3.549044e-02
## 128 -1.55609169  0.0068110637 -0.265297510  0.0743377196  4.703956e-02
## 129 -0.70553360 -0.1111277500  0.406726319  0.1059852824  1.116019e-01
## 130 -0.37152095 -0.3426430760  0.651235600 -0.0342204975 -2.336350e-01
## 131 -1.80298443  0.1566588355  0.149688329 -0.1528224138  1.237775e-01
## 132 -1.89628395  0.3193930464 -0.104125180  0.0140868921  1.001545e-01
## 133 -2.66810762  0.2718374969  0.038008200 -0.0445491090 -8.840356e-02
## 134 -2.65572613  0.2824663384  0.127563144 -0.0426909518 -4.555742e-02
## 135 -2.69759533  0.2628851593  0.095074984 -0.0185833802 -3.815432e-02
## 136 -2.65360734  0.2446440914  0.043584080 -0.0481840701  1.541052e-02
## 137 -2.52447134  0.2125200615  0.035852447 -0.0651568800  6.107055e-02
## 138 -2.54876314  0.1786194800  0.040579119  0.0542593589  1.011323e-01
## 139 -2.54017668  0.1773372557  0.082359902 -0.0367968111  5.191725e-02
## 140 -2.51462510  0.1963071422  0.147994845  0.0308894805  6.245364e-02
## 141 -2.22684111  0.0664770197  0.344576284 -0.1178358356  1.124475e-01
## 142 -2.16442439  0.1547189053  0.166058426  0.0007872166  9.880018e-02
## 143 -2.58962359  0.2373922335  0.262236166  0.0035859962 -1.727436e-03
## 144 -2.64967327  0.2426747541  0.116099865  0.0865738162  3.591498e-02
## 145  2.34552653 -0.6170866616 -1.026774759 -0.0269297956  7.855150e-02
## 146  2.87896690 -1.0722509911 -1.421241720 -0.0102276492  1.958110e-02
## 147  2.81503354 -0.7414016185 -1.452288098  0.0194430387 -1.700790e-01
## 148  2.75223850 -1.2914659551 -0.774808487 -0.2333659692 -3.038461e-01
## 149  4.03081563 -1.3337428299 -1.095019049  0.2466995536 -7.300564e-03
## 150  3.06630925 -1.0311694890  0.012743498  0.4848179174 -2.989907e-01
## 151  2.53052348 -0.7224180122 -0.839269532  0.8165547622 -2.212532e-01
## 152  2.72279097 -1.2701695699 -1.177413732  0.6080282813 -3.808569e-01
## 153  6.01097458 -1.6726079092 -0.075034642  0.6128458914 -5.079548e-02
## 154  7.03943332 -1.8309042949  0.512884974  0.2090776169  1.791425e-02
## 155  2.32656104 -0.5629672494 -0.642163722  0.3643818111  1.298225e-01
## 156  1.45106237  0.1189229782 -1.166836344  0.3196987355  3.281143e-01
## 157  1.30850590 -0.4426422539 -0.710214805  0.0931435170 -5.505918e-01
## 158  1.68942298 -0.3243159325 -0.362887744 -0.3824814747 -5.766252e-01
## 159  2.31137119 -0.8914310085 -0.545507366 -0.3222919335 -6.229944e-01
## 160  2.02752514 -1.4731864489 -0.142624703 -0.2511628477 -3.400538e-01
## 161  2.22469513 -0.9728886086  0.421779134  0.0160593899 -2.721278e-01
## 162  1.83636701 -0.8760390753  0.882318371 -0.5119779433 -3.820804e-01
## 163  1.27887960 -0.7368867317  0.355248387  0.2289901681 -2.285620e-01
```

```
## 164  0.62923532 -0.3871980647 -0.635365394  0.1925365178 -2.813970e-01
## 165  3.03658004 -1.2178141639  0.262901306 -0.0077732204 -2.559406e-01
## 166  3.71962299 -1.1537985349  0.747576839 -0.3630661757 -1.916401e-01
## 167  0.82450079 -0.1031426378 -0.569136242 -0.0382949213 -3.866814e-01
## 168  0.47025497  0.1149949211 -0.585518228  0.2341330918 -5.047033e-01
## 169 -2.39923115  0.1876172008 -0.015596503 -0.1452143435  1.367434e-01
## 170 -2.26827358  0.1661217979  0.065436233 -0.2588923869  7.225556e-02
## 171 -2.30811844  0.2710061977 -0.050621306 -0.1865329349  5.413656e-02
## 172 -2.26676784  0.1141257454 -0.002208808 -0.1476640711 -1.113659e-02
## 173 -2.13135201  0.0939278018 -0.070673043 -0.1745222726  5.292334e-02
## 174 -2.27245547  0.2056852268 -0.029867936 -0.1199753607  1.381375e-01
## 175 -2.09370845  0.1177672659 -0.110481037 -0.0727125483  8.347893e-02
## 176 -2.21779385  0.1321885210 -0.156086105 -0.1656668080  3.592566e-02
## 177 -1.90606352  0.1110531297  0.247374840 -0.0114428322  1.774878e-01
## 178 -1.87259182 -0.0031665020  0.367940283 -0.1945735004  1.944308e-01
## 179 -2.36135176  0.2432632753  0.144516759 -0.1870399435  1.310685e-01
## 180 -2.48571371  0.2841904873  0.120020096 -0.0869559691 -1.832149e-02
## 181 -1.26961171 -0.1197719578 -0.309339333  0.1225836457  9.096281e-03
## 182 -0.85063440 -0.5737251400 -0.027018925  0.1568596213 -1.748286e-02
## 183 -0.96816425 -0.2394169490 -0.436794944 -0.0529426002 -1.096960e-01
## 184 -0.95069995 -0.3346894048 -0.238191911  0.0205028978 -3.667052e-01
## 185 -0.36352720 -0.2314990958 -0.082542298 -0.1568463928 -6.659002e-02
## 186 -0.71013129 -0.1820106055  0.148674282 -0.0755388345  4.364364e-02
## 187 -1.24055822 -0.1272039831 -0.177633276  0.2010101001 -7.525121e-02
## 188 -1.74587129  0.0160092776 -0.283925598  0.1730019295 -9.365884e-02
## 189  0.11649731 -0.6406830420  0.240441326 -0.1541132625  1.074930e-01
## 190  0.67825007 -0.6961025898  0.623607136 -0.0961253004 -8.645742e-02
## 191 -0.85182871 -0.0940195508  0.153442789 -0.1443935912  9.870627e-02
## 192 -1.23537446  0.2341509451 -0.128388273 -0.0122219703  5.806335e-02
## 193  1.70190137 -0.5159923097 -0.325765283 -0.8052335651  3.365897e-01
## 194  2.04391592 -1.0900094871 -0.370054750 -0.8594590440  1.042954e-01
## 195  2.56671592 -0.8408823565 -0.595671609 -1.2346573196  6.538437e-01
## 196  1.92807393 -1.3025089753 -0.068416141 -0.8245821755  3.597817e-01
## 197  3.57474705 -1.0429888337  0.133722827 -0.9007778003  5.818653e-01
## 198  2.87533077 -1.1007852024 -0.005827103 -0.6868264902  2.105809e-01
## 199  1.96506994 -0.6421228447 -0.473720667 -0.6700735535  1.078581e-01
## 200  0.50421645 -0.2346095519 -0.935104808  0.0415165054 -2.306028e-01
## 201  3.53254123 -1.1984526609  0.311206880 -0.5188079435  1.590871e-01
## 202  4.96284191 -1.6444007102  1.008005095 -0.9135004720 -2.109789e-01
## 203  1.83739540 -0.7568504090 -0.079103155 -0.7780456295  1.604914e-01
## 204  0.85330826 -0.1639570152 -0.042098037 -0.6489290077  3.921295e-02
## 205 -1.08051837  0.1166351371 -0.150943620 -0.4723251489  1.429969e-01
## 206 -1.11966913  0.0078124576  0.034952127 -0.4913888339 -2.098253e-01
## 207 -0.88856247  0.0398369915 -0.269843495 -0.4899394541 -1.682633e-01
## 208 -1.02010215 -0.1819031625  0.051309079 -0.3640638394 -2.486626e-02
## 209 -0.69781584 -0.2984588157 -0.206998523 -0.1268680646  1.808645e-01
## 210 -0.60810276 -0.2208600430 -0.300977378 -0.4188402613  1.391706e-01
## 211 -1.03662339 -0.0008807372 -0.277666987 -0.2490288650  1.124701e-01
## 212 -1.31274816  0.2217168461 -0.532271477 -0.3235846339 -4.053245e-02
## 213 -0.23911297 -0.3532087332 -0.373745759 -0.2634526169  3.928385e-01
## 214  0.62365939 -0.9963943295  0.154385215 -0.3276605815  3.106555e-01
## 215 -1.06133662 -0.0751566825 -0.086363432 -0.4686898168  7.881957e-02
## 216 -1.14160399  0.2170216228 -0.227823712 -0.3068372922 -3.162610e-02
## 217 -2.62101573  0.2329356326  0.198854615 -0.0544391344 -1.125806e-02
## 218 -2.59535516  0.2685863681  0.137804426 -0.0200449899  3.807445e-02
## 219 -2.52812125  0.2251271229  0.121211391 -0.0183295856  3.015030e-02
## 220 -2.56260304  0.1642870182  0.136827126 -0.0376729704  7.167053e-04
## 221 -2.40968473  0.1884283333  0.304847948 -0.2370001092  1.739058e-01
## 222 -2.43346168  0.2002566782  0.090114487 -0.1310910356  1.750730e-01
## 223 -2.48348814  0.2208985942  0.121192369 -0.0819742555  1.703319e-01
## 224 -2.58059669  0.2063373304  0.040771094 -0.1312054127  1.190415e-01
## 225 -2.45150404  0.2677495125  0.103615032  0.0536655560  9.459541e-02
## 226 -2.25141818  0.3322084109  0.275407458  0.1400881783  8.783822e-02
## 227 -2.52912182  0.2805662553  0.048759781  0.0108984465  9.736371e-02
## 228 -2.66327358  0.3164735641  0.090903671 -0.0215222276  5.912513e-02
## 229 -2.44137437  0.1631749243  0.070571072  0.0368522535  2.206579e-02
## 230 -2.46928029  0.1785961313  0.082905151  0.0959520838  1.675458e-02
## 231 -2.36638077  0.1329292555  0.007302199  0.0015374318  1.263792e-02
## 232 -2.46654244  0.1622182240  0.072083647  0.0287310761 -8.331499e-02
## 233 -2.30978147  0.1390726351 -0.051019054  0.0021681501 -1.059615e-01
## 234 -2.25130819  0.1517332416  0.227609063  0.0212399371 -3.510025e-02
## 235 -2.39078390  0.1680275673  0.131321846 -0.0710154153 -6.477408e-02
## 236 -2.56280920  0.2019065890  0.050915678 -0.0331946131 -6.567511e-05
## 237 -2.18952622  0.1134311580  0.090484499 -0.0990667184  7.944346e-04
## 238 -1.86473890  0.0691739267  0.607726501 -0.0282061755 -3.322496e-02
```

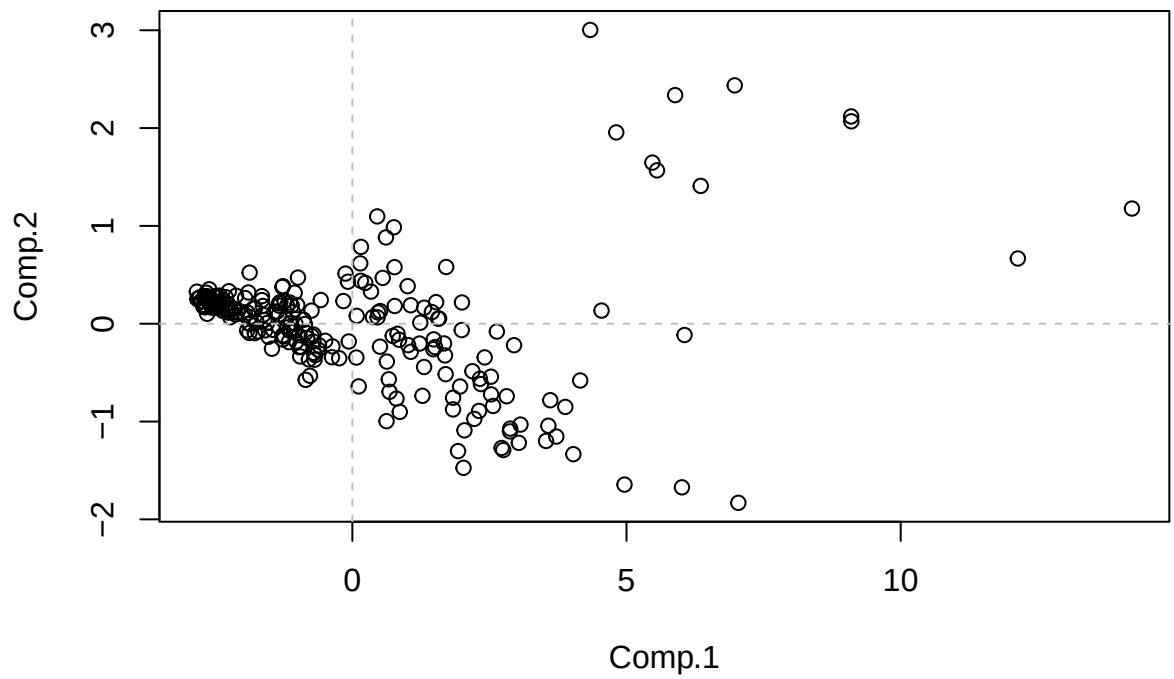
```
## 239 -2.41522729  0.2873411085  0.191661815 -0.0818619601  2.791979e-02
## 240 -2.42389248  0.2248703920 -0.054401809 -0.0601981515  6.937338e-03
##           Comp.6        Comp.7        Comp.8
## 1    0.604599722 -0.0112544994  4.578010e-02
## 2    0.360285676 -0.0355573728 -5.413831e-03
## 3    0.253123411  0.0469944804 -1.476948e-02
## 4   -0.032881081  0.0577570573  1.361058e-03
## 5    0.260875430  0.2816695147 -2.511605e-02
## 6   -0.180827497  0.3905107121  1.565072e-02
## 7   -0.143246990  0.1842061572  7.145989e-02
## 8   -0.045274045  0.1448226194 -3.978882e-02
## 9   -0.052787832  0.3052817354 -1.832597e-01
## 10  -0.143008079  0.2384109383 -1.488152e-01
## 11  -0.388414469  0.2590586301 -2.570082e-01
## 12  -0.187839213  0.2209510565 -2.380269e-01
## 13   0.169623538  0.0913459923 -4.317582e-02
## 14   0.081073439  0.0775652058 -4.010676e-02
## 15   0.013974315  0.0563948275 -8.345500e-03
## 16   0.014654330  0.1517551016 -3.774038e-02
## 17   0.038077634  0.0722894495 -3.233627e-02
## 18  -0.046401204  0.0906376795 -1.104193e-02
## 19   0.030527394  0.0334411223  7.129113e-03
## 20   0.078074758  0.0296762106 -2.303592e-02
## 21   0.119263030  0.0521268278 -1.273404e-02
## 22  -0.005772754  0.0497416281  1.116760e-01
## 23   0.026009536  0.0222823112 -4.833900e-03
## 24   0.010976197  0.0066635313  6.206708e-05
## 25  -0.226033750  0.0340289201 -8.491644e-02
## 26  -0.084907567 -0.0061239705 -2.856177e-01
## 27  -0.258071091 -0.1308480214 -7.770882e-02
## 28  -0.389323486 -0.3711163367  1.565468e-02
## 29  -0.247168162  0.1767451591  9.084577e-02
## 30  -0.146364202  0.2106758467 -1.326824e-02
## 31  -0.593644864 -0.1934051752  3.168913e-02
## 32  -0.449941943 -0.0710800707  8.092383e-02
## 33  -0.468864223  0.1789417619 -3.670275e-01
## 34   0.106862588  0.2220074120 -2.186183e-01
## 35  -0.363163500 -0.1497423285 -3.057122e-01
## 36  -0.126047250  0.1614732436 -8.794485e-02
## 37  -0.027653235 -0.0663112799 -1.301018e-02
## 38   0.073744365 -0.1211805219 -8.571963e-02
## 39   0.046324548 -0.1075869954 -2.601330e-02
## 40  -0.146035777 -0.0817614705 -7.224160e-02
## 41  -0.034593884 -0.0629506848 -7.522272e-02
## 42  -0.147436149 -0.0393342931 -9.561750e-03
## 43  -0.090485495 -0.1576682326 -8.326040e-02
## 44   0.135944940 -0.1660257744 -1.009626e-01
## 45   0.136573211 -0.1387747173 -7.155993e-02
## 46  -0.085438326  0.0643484418  2.514561e-02
## 47  -0.117602363 -0.1221928036 -4.774979e-02
## 48   0.063102771 -0.0139196816  5.861457e-02
## 49   0.199487698 -0.0878352383 -6.506110e-02
## 50   0.140394789 -0.0338563390 -5.844965e-02
## 51  -0.092978242 -0.1197421042 -1.407003e-01
## 52  -0.241034694  0.1039791973 -3.736159e-02
## 53  -0.090319708 -0.0079559480 -1.265274e-01
## 54   0.092632004  0.1224496600  1.295570e-01
## 55  -0.001582579  0.0900407993  2.751577e-02
## 56  -0.055768975  0.1904339212 -4.336171e-02
## 57  -0.045268063  0.3191042790  1.004597e-01
## 58   0.256197100  0.1617221553  3.133828e-02
## 59  -0.119500299  0.0851430090 -4.274350e-02
## 60  -0.021480943  0.1467695367  3.977895e-02
## 61   0.261930351 -0.1350958475 -5.185962e-02
## 62   0.160133110 -0.1051929434 -2.689262e-01
## 63   0.109932247 -0.1500221374  8.127618e-03
## 64  -0.092400369 -0.2457131697 -9.584680e-02
## 65  -0.134910273 -0.0295321251  1.930735e-01
## 66  -0.128108492  0.0638417790  2.375564e-01
## 67   0.083992891 -0.1360158140  1.053418e-01
## 68   0.094782774 -0.0985564957 -3.660811e-02
## 69   0.071686980 -0.0060735380 -6.954385e-02
## 70   0.162009388  0.0122830642  3.068585e-02
## 71   0.249672188 -0.1254947765  4.229457e-02
## 72   0.000865671 -0.1507402075  1.452024e-01
```

```
## 73   0.007395576 -0.3278672183  3.105477e-02
## 74   0.592111000 -0.4073852336 -2.445385e-01
## 75   0.369788585 -0.1341901782  1.105647e-01
## 76  -0.284500047 -0.4056749737  2.580995e-01
## 77   0.115510780 -0.0518100365  3.702976e-01
## 78  -0.138023701 -0.1297272727 -3.509778e-02
## 79  -0.191441634  0.2119944960  2.213595e-01
## 80  -0.121791530 -0.3194743872  1.522288e-01
## 81  -0.223346892 -0.0751403715  8.019454e-02
## 82  -0.020133427  0.3703792893  1.407522e-01
## 83  -0.171034245  0.0971476465  2.958029e-02
## 84  -0.074651924 -0.1268601836  5.121799e-02
## 85   0.165156645  0.0575426470 -7.196398e-02
## 86   0.088222217  0.0418794164 -3.516927e-02
## 87   0.035107798 -0.1085605251 -7.607477e-02
## 88  -0.123389377  0.0286181811  3.689243e-02
## 89  -0.027002356 -0.0719567106 -6.825051e-02
## 90   0.173931634 -0.0658280849 -1.165217e-01
## 91   0.116835530 -0.0201023168 -4.572361e-03
## 92   0.199025383  0.0176322141 -2.193377e-02
## 93   0.036586447  0.0085851659  1.721300e-02
## 94   0.075152721  0.1408967118  1.266229e-01
## 95  -0.017522028  0.1088984317  4.172416e-02
## 96  -0.011087756  0.0319644355  3.848412e-02
## 97  -0.037471690 -0.0007096703 -6.052615e-02
## 98  -0.108850110 -0.1392662586 -1.396004e-01
## 99  -0.289173410  0.0758494825 -7.268499e-02
## 100 -0.036885460 -0.1226918958  2.824006e-02
## 101  0.034556515 -0.2919842657 -1.496947e-02
## 102 -0.030604974 -0.1594275316  8.303961e-03
## 103  0.003812184 -0.2375338901  9.946856e-02
## 104  0.126859957 -0.1194076935  2.070921e-02
## 105 -0.010802478 -0.0001985892  1.422492e-01
## 106  0.233606709  0.0153655375  9.174755e-02
## 107 -0.063873477  0.0448876402 -1.511585e-02
## 108 -0.035095105 -0.0976186069  1.360313e-01
## 109  0.162167908 -0.0629809027 -1.245843e-01
## 110  0.460473405 -0.2204402222 -2.523376e-01
## 111  0.134079471 -0.0316633629 -3.243479e-01
## 112 -0.102711750  0.0769250830 -2.389082e-01
## 113  0.235435653  0.2638092290  1.836699e-01
## 114 -0.514167857 -0.0596021262  2.629620e-01
## 115 -0.127009120 -0.2508521210  3.711651e-02
## 116 -0.305848776 -0.3989715751  6.338996e-02
## 117  0.403618081 -0.2364124272  1.963052e-01
## 118  0.466755194  0.1816508893 -1.838869e-02
## 119 -0.080013345 -0.1866897882  3.028141e-02
## 120 -0.057935631  0.1403582834  6.061831e-02
## 121  0.093040042 -0.0569384339 -8.572783e-02
## 122 -0.096048460  0.0799125437 -4.964383e-02
## 123 -0.043689629  0.0437640522 -3.161352e-02
## 124 -0.035343348  0.0280485362 -5.829938e-02
## 125 -0.073814477 -0.0674978026 -4.282094e-02
## 126  0.071840741 -0.0916713087 -4.237754e-02
## 127 -0.029776788 -0.2038296585  4.664674e-02
## 128  0.207856779 -0.2438560171 -1.926805e-02
## 129 -0.037708104  0.0185389046  1.369945e-01
## 130  0.151195839 -0.0723747578 -5.649956e-02
## 131 -0.019028706 -0.1297160262 -6.177220e-02
## 132  0.237353327 -0.0150918837  4.732348e-02
## 133  0.061121215 -0.0067886255 -4.164082e-02
## 134  0.032966936 -0.0375834974 -3.558256e-02
## 135  0.032374287 -0.0513303008 -9.023713e-03
## 136 -0.007190614  0.0241466447  4.448691e-03
## 137 -0.022282536 -0.0453332362  3.333136e-02
## 138 -0.019670565  0.0297986746 -3.262430e-02
## 139 -0.005028219  0.0021179788 -3.973663e-02
## 140 -0.063973230  0.0088766540 -7.066871e-02
## 141 -0.073834250 -0.0274560444 -4.770226e-02
## 142  0.058986243  0.0736448422  4.859517e-02
## 143 -0.005599863 -0.0366258620 -2.069919e-02
## 144 -0.076623444  0.0094374598  2.193413e-02
## 145 -0.060096847  0.1332907371  6.109105e-02
## 146  0.167229885 -0.1491356014  3.135633e-02
## 147  0.165273424 -0.1897266141 -6.788864e-02
```

```
## 148  0.530951975  0.1219316265  6.138860e-02
## 149  0.217504465  0.0962310185 -1.458731e-01
## 150 -0.191494113 -0.3053273930  1.901231e-01
## 151 -0.458420868  0.0393502941 -7.053419e-03
## 152  0.260790327 -0.1134703094 -3.723988e-02
## 153  0.734674337  0.3667901587  6.083257e-02
## 154 -0.122451919  0.1225477153 -3.613493e-01
## 155 -0.098200879  0.1364436332  1.244720e-01
## 156  0.094676552  0.4187613951  1.690889e-01
## 157 -0.464526720  0.4652669218  3.289131e-02
## 158 -0.093429570  0.0339023011 -7.367016e-02
## 159 -0.158068220 -0.0015112481 -5.272431e-02
## 160  0.227328541  0.1175670649 -5.340162e-02
## 161  0.067427026 -0.1827233060 -8.346440e-02
## 162 -0.297692164 -0.2799790364 -1.577288e-01
## 163 -0.480980273 -0.1201866368  5.046367e-03
## 164  0.134321272 -0.2543525514  1.328944e-01
## 165  0.319081338  0.2414408142 -3.347644e-03
## 166 -0.587656475  0.1745968867 -5.944829e-02
## 167 -0.319453885 -0.1469178553  1.146370e-01
## 168  0.197768956 -0.0626749697  1.135858e-01
## 169 -0.123190654  0.0090215821  4.295520e-02
## 170  0.021160257  0.0204109604 -1.806706e-02
## 171 -0.012475512 -0.0423979901  3.153473e-02
## 172  0.095859545  0.0424102762 -2.801732e-02
## 173  0.106374088 -0.0385548019 -1.019035e-02
## 174 -0.122684861  0.0203149290  1.194089e-02
## 175 -0.149023777 -0.0345825914  1.046743e-01
## 176  0.129651501 -0.0455521979  1.313557e-02
## 177 -0.030708182  0.0187867775  1.081263e-01
## 178 -0.132105181 -0.0036441880 -4.140002e-02
## 179 -0.041463601 -0.0216904397 -4.492696e-03
## 180  0.040852411  0.0274652658  6.485087e-03
## 181  0.121877806  0.0912006194 -5.016219e-03
## 182 -0.105362385  0.1066238340 -2.514599e-02
## 183  0.124380696 -0.1170557841 -4.882208e-02
## 184  0.307350322  0.0889235417  4.263310e-02
## 185  0.217770816 -0.1166550465  8.879557e-02
## 186 -0.020246520 -0.0264683294 -2.028692e-02
## 187 -0.058268933 -0.1109403039  4.016859e-02
## 188 -0.090408660 -0.0618429569  5.770283e-02
## 189  0.295812705 -0.0300162430 -1.217866e-01
## 190  0.194732997  0.0037521635  3.750556e-02
## 191  0.036581181 -0.1635340490  1.170944e-01
## 192  0.305968398 -0.0798891989  1.100429e-01
## 193 -0.020131115  0.1048404415  1.506959e-01
## 194 -0.017700392  0.0581922043 -1.475944e-01
## 195  0.060677796 -0.0331518655  3.261715e-02
## 196 -0.145977408  0.1851008981  4.251247e-02
## 197 -0.371824993  0.0063564988  1.186189e-01
## 198 -0.181259055  0.0706810571  5.677194e-03
## 199 -0.123245431 -0.1358257208 -5.461921e-02
## 200 -0.108407171  0.0680314873  2.628675e-01
## 201  0.024982962  0.2869114576  1.266891e-01
## 202 -0.054852996 -0.0430569394  7.288614e-02
## 203 -0.175130902 -0.1264940519 -4.314553e-02
## 204  0.187509180  0.1202807598  5.095178e-02
## 205 -0.123615791 -0.0091168779  6.223744e-02
## 206 -0.099820233  0.0438521881 -1.108263e-01
## 207  0.006849122 -0.0113933122 -2.742697e-02
## 208 -0.093525454 -0.0249858418  5.802672e-04
## 209  0.024618944 -0.0068686768  9.935726e-02
## 210  0.022841309 -0.0092294410  1.009142e-01
## 211 -0.179022102 -0.0422346515  3.609588e-02
## 212 -0.114611072 -0.0488251842  1.926363e-01
## 213  0.192162398  0.1705846064  1.065461e-01
## 214  0.043519926  0.1940141218  7.363091e-02
## 215  0.176995601  0.0169661320 -1.964525e-02
## 216 -0.084810757  0.1586269793  3.881679e-05
## 217  0.069022613  0.0186474337 -2.458776e-02
## 218  0.043501045  0.0472924086  1.080433e-02
## 219  0.071404198 -0.0532024018  2.364237e-02
## 220  0.106116281  0.0159552268 -1.673390e-03
## 221  0.005184649  0.0193117937  8.180476e-03
## 222  0.015851496  0.0652350785  2.557228e-02
```

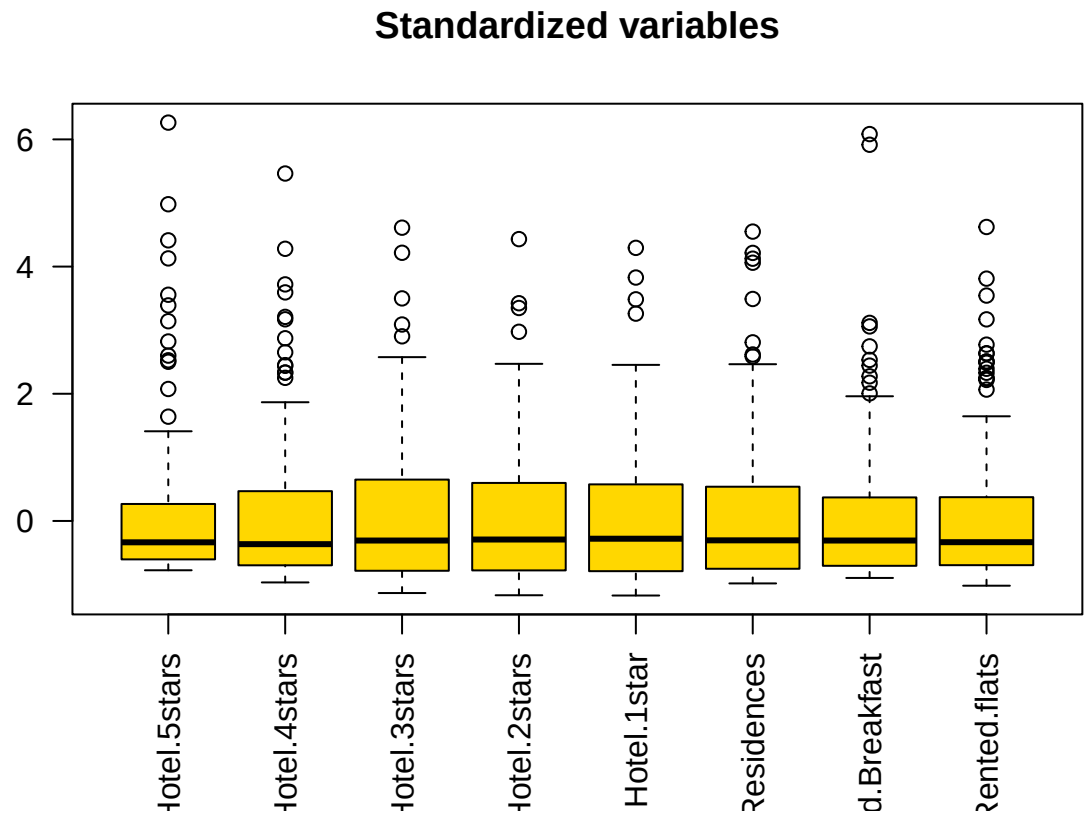
```
## 223 -0.032134752  0.1203905020  2.875604e-03
## 224  0.027596099  0.0096610575  4.299770e-03
## 225  0.029431260  0.0034399399  2.223972e-02
## 226 -0.001866489  0.1231245008  1.029336e-01
## 227 -0.075901255  0.0011480853  6.966421e-02
## 228  0.072485811  0.0278226048  3.120437e-02
## 229  0.081985979  0.0482134790 -2.620135e-02
## 230  0.014509834  0.0134112907 -1.062769e-02
## 231  0.100115056 -0.0253578923 -2.380173e-02
## 232  0.032428577  0.0282417065 -1.967261e-02
## 233  0.121285738  0.0740391296 -4.511249e-02
## 234 -0.087201651  0.0803320337 -1.226694e-02
## 235  0.066733301  0.0036749389 -1.723957e-02
## 236 -0.022011017  0.0559198679 -3.131705e-02
## 237 -0.007388126  0.1136815227 -2.162455e-04
## 238  0.032090827  0.0262775184  5.430626e-02
## 239 -0.019616677 -0.0334668414  1.708846e-02
## 240 -0.010607204  0.0102906154  4.018632e-02
```

```
plot(scores.tourists[, 1:2])
# library(rgl)
# plot3d(scores.tourists[,1:3])
abline(h=0, v=0, lty=2, col='grey')
```

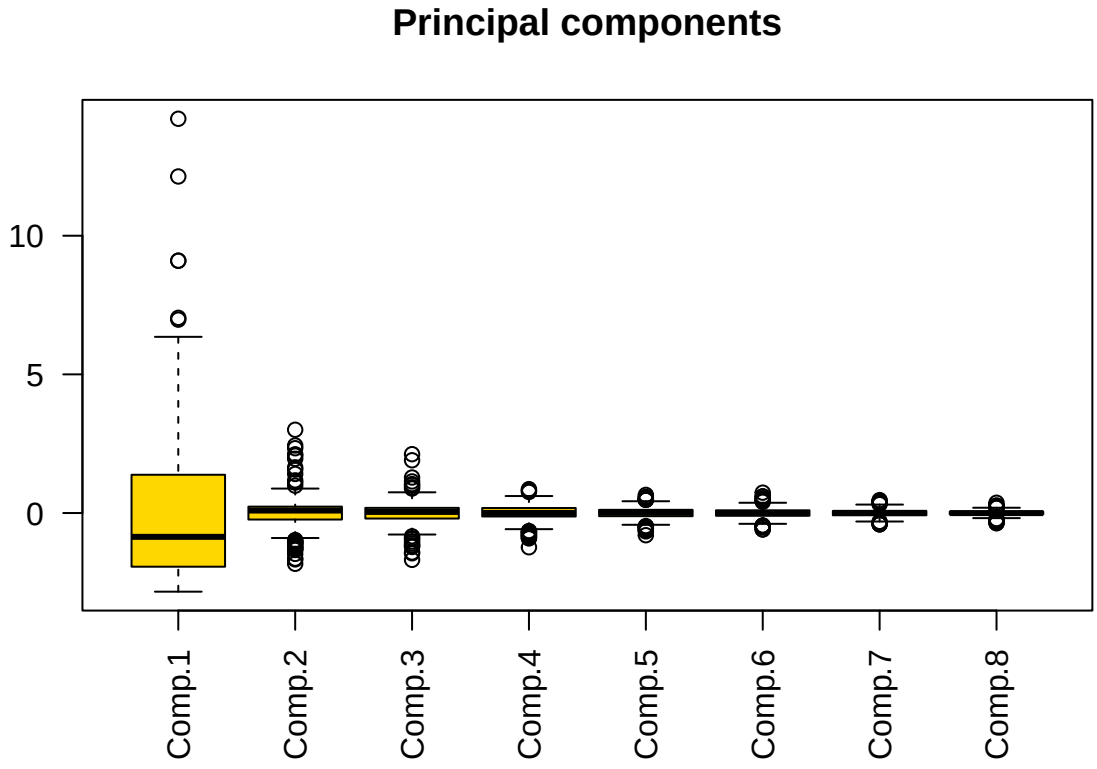



```
#! Scores show which data have high/low values wrt the PCs
#! Like data are very concentrated in small and negative values for PC1, and
#! small and positive for PC2. So it shows that less people (small and negative
#! PC1) went to more expensive places (small but positive PC2)
```

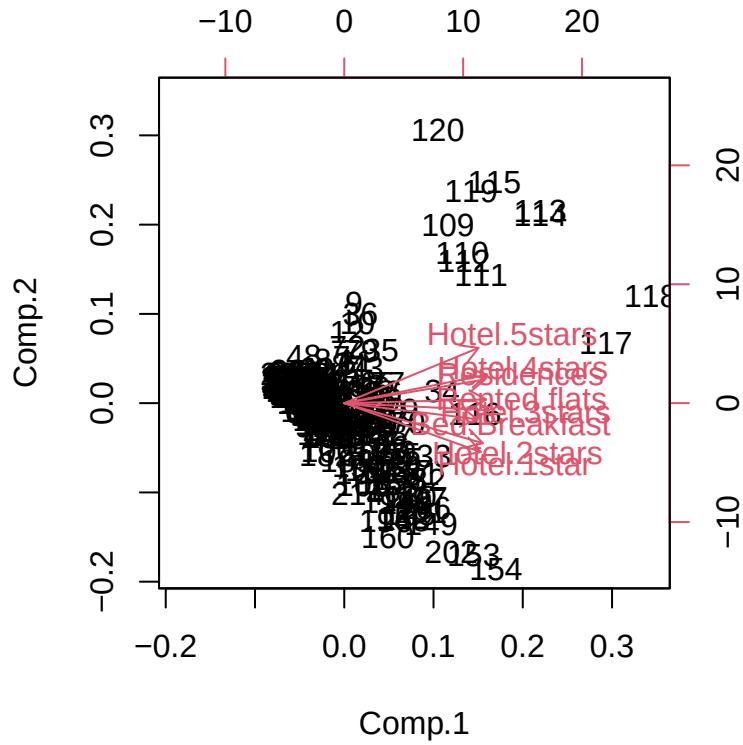
```
# layout(matrix(c(1,2), 2))
# par(mfrow=c(2,1))
boxplot(tourists.sd, las=2, col='gold', main='Standardized variables')
```




```
scores.tourists = data.frame(scores.tourists)
boxplot(scores.tourists, las=2, col='gold', main='Principal components')
```



```r
biplot(pc.tourists)
```




```r
#! arrows show where to go to have higher values for the PCs
#! Like to increment PC1 look at which arrows go towards positive/high PC1 axis
```

```r
# Let's use the categorical variables to further interpret the results
head(tourists.label)
```

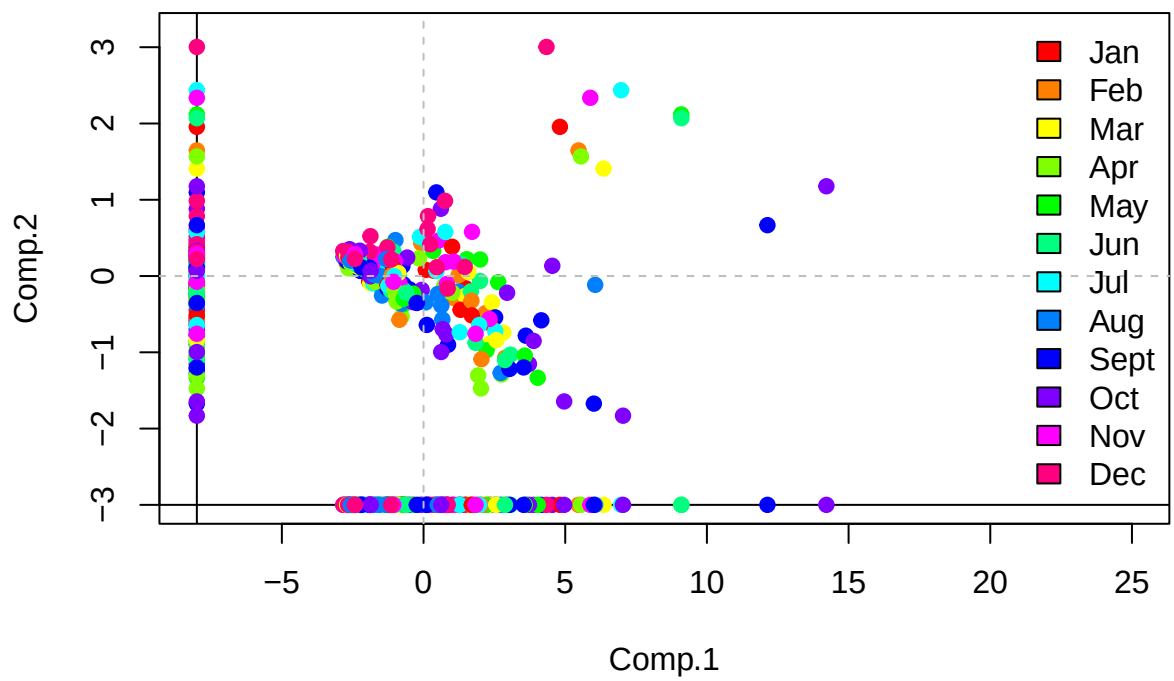
```
##   Month Region.of.origin
## 1   Jan         PIEMONTE
## 2   Feb         PIEMONTE
## 3   Mar         PIEMONTE
## 4   Apr         PIEMONTE
## 5   May         PIEMONTE
## 6   Jun         PIEMONTE
```

```r
# Color according to Month
tourists.label[, 1]
```

```
##   [1] "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"
##  [11] "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"
##  [21] "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"
##  [31] "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"
##  [41] "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"
##  [51] "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"
##  [61] "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"
##  [71] "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"
##  [81] "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"
##  [91] "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"
## [101] "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"
## [111] "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"
## [121] "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"
## [131] "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"
## [141] "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"
## [151] "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"
## [161] "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"
## [171] "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"
```

```
## [181] "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"
## [191] "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"
## [201] "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"
## [211] "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"  "Mar"  "Apr"
## [221] "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"  "Jan"  "Feb"
## [231] "Mar"  "Apr"  "May"  "Jun"  "Jul"  "Aug"  "Sept" "Oct"  "Nov"  "Dec"
```

```
# We order the labels according to time order
months = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sept", "Oct",
           "Nov", "Dec")
tourists.label[, 1] = factor(tourists.label[, 1], levels=months)

col.ramp = rainbow(12)
col.lab1 = rep(NA, n)
for(i in 1:n)
  col.lab1[i] = col.ramp[which(tourists.label[i,1] == levels(tourists.label[,1]))]

plot(scores.tourists[,1:2], col=col.lab1, pch=19, xlim=c(-8,25), ylim=c(-3,3.2))
abline(h=-3, v=-8, col=1)
points(scores.tourists[,1], rep(-3, n), col=col.lab1, pch=19)
points(rep(-8, n), scores.tourists[,2], col=col.lab1, pch=19)
abline(h=0, v=0, lty=2, col='grey')
legend('topright', levels(tourists.label[,1]), fill=rainbow(12), bty='n')
```

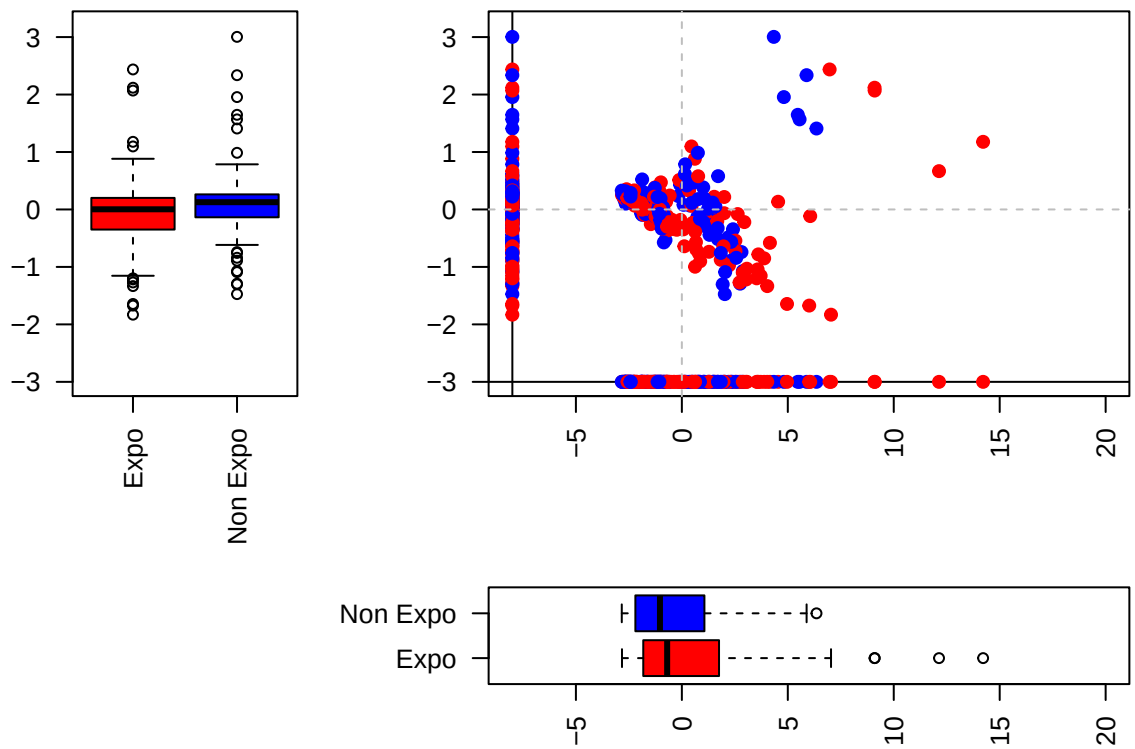



```
# Months of Expo 2015: May to Oct
expo.label=factor(ifelse(tourists.label[,1] %in% c("May", "Jun", "Jul", "Aug",
                                                   "Sept", "Oct"), 'Expo', 'Non Expo'))
col.expo = ifelse(tourists.label[,1] %in% c("May","Jun","Jul","Aug","Sept","Oct"),
                  'red', 'blue')
```

3.2.8 Scores in categories

```
layout(cbind(c(2,4), c(1,3)), widths = c(2,4), heights=c(4,2))
par(mar=rep(3,4)) # just for margin size, not really necessary

plot(scores.tourists[,1:2], col=col.expo, pch=19, xlim=c(-8,20), ylim=c(-3,3.2), las=2)
abline(h=-3, v=-8, col=1)
```

```r
points(scores.tourists[,1], rep(-3,n), col=col.expo, pch=19)
points(rep(-8, n), scores.tourists[,2], col=col.expo, pch=19)
abline(h=0, v=0, lty=2, col='grey')
boxplot(scores.tourists[,2]~expo.label, col=c('red', 'blue'), ylim=c(-3,3.2), las=2)
boxplot(scores.tourists[,1]~expo.label, col=c('red', 'blue'), ylim=c(-8,20),
        horizontal=T, las=2,)
```

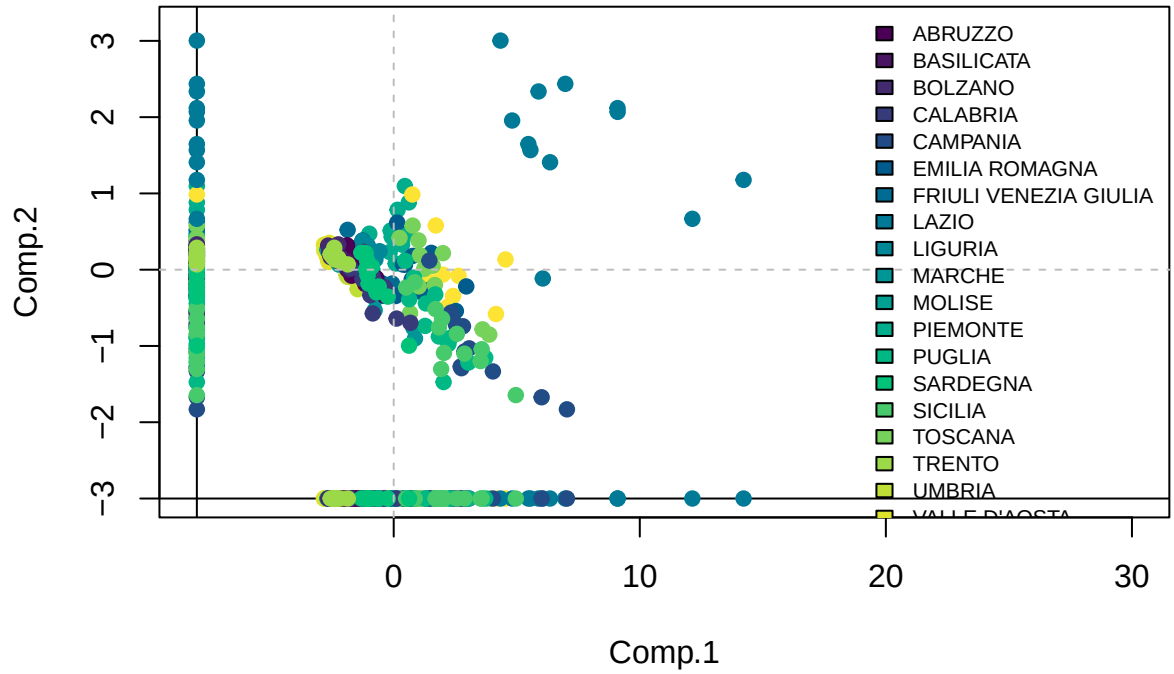



```r
# Color according to Region of Origin
tourists.label[,2]
```

```
##   [1] "PIEMONTE"               "PIEMONTE"               "PIEMONTE"
##   [4] "PIEMONTE"               "PIEMONTE"               "PIEMONTE"
##   [7] "PIEMONTE"               "PIEMONTE"               "PIEMONTE"
##  [10] "PIEMONTE"               "PIEMONTE"               "PIEMONTE"
##  [13] "VALLE D'AOSTA"          "VALLE D'AOSTA"          "VALLE D'AOSTA"
##  [16] "VALLE D'AOSTA"          "VALLE D'AOSTA"          "VALLE D'AOSTA"
##  [19] "VALLE D'AOSTA"          "VALLE D'AOSTA"          "VALLE D'AOSTA"
##  [22] "VALLE D'AOSTA"          "VALLE D'AOSTA"          "VALLE D'AOSTA"
##  [25] "VENETO"                 "VENETO"                 "VENETO"
##  [28] "VENETO"                 "VENETO"                 "VENETO"
##  [31] "VENETO"                 "VENETO"                 "VENETO"
##  [34] "VENETO"                 "VENETO"                 "VENETO"
##  [37] "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA "
##  [40] "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA "
##  [43] "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA "
##  [46] "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA " "FRIULI VENEZIA GIULIA "
##  [49] "LIGURIA"                "LIGURIA"                "LIGURIA"
##  [52] "LIGURIA"                "LIGURIA"                "LIGURIA"
##  [55] "LIGURIA"                "LIGURIA"                "LIGURIA"
##  [58] "LIGURIA"                "LIGURIA"                "LIGURIA"
##  [61] "EMILIA ROMAGNA"         "EMILIA ROMAGNA"         "EMILIA ROMAGNA"
##  [64] "EMILIA ROMAGNA"         "EMILIA ROMAGNA"         "EMILIA ROMAGNA"
##  [67] "EMILIA ROMAGNA"         "EMILIA ROMAGNA"         "EMILIA ROMAGNA"
##  [70] "EMILIA ROMAGNA"         "EMILIA ROMAGNA"         "EMILIA ROMAGNA"
##  [73] "TOSCANA"                "TOSCANA"                "TOSCANA"
##  [76] "TOSCANA"                "TOSCANA"                "TOSCANA"
##  [79] "TOSCANA"                "TOSCANA"                "TOSCANA"
##  [82] "TOSCANA"                "TOSCANA"                "TOSCANA"
```

```
##  [85] "UMBRIA"                 "UMBRIA"                 "UMBRIA"
##  [88] "UMBRIA"                 "UMBRIA"                 "UMBRIA"
##  [91] "UMBRIA"                 "UMBRIA"                 "UMBRIA"
##  [94] "UMBRIA"                 "UMBRIA"                 "UMBRIA"
##  [97] "MARCHE"                 "MARCHE"                 "MARCHE"
## [100] "MARCHE"                 "MARCHE"                 "MARCHE"
## [103] "MARCHE"                 "MARCHE"                 "MARCHE"
## [106] "MARCHE"                 "MARCHE"                 "MARCHE"
## [109] "LAZIO"                  "LAZIO"                  "LAZIO"
## [112] "LAZIO"                  "LAZIO"                  "LAZIO"
## [115] "LAZIO"                  "LAZIO"                  "LAZIO"
## [118] "LAZIO"                  "LAZIO"                  "LAZIO"
## [121] "ABRUZZO"                "ABRUZZO"                "ABRUZZO"
## [124] "ABRUZZO"                "ABRUZZO"                "ABRUZZO"
## [127] "ABRUZZO"                "ABRUZZO"                "ABRUZZO"
## [130] "ABRUZZO"                "ABRUZZO"                "ABRUZZO"
## [133] "MOLISE"                 "MOLISE"                 "MOLISE"
## [136] "MOLISE"                 "MOLISE"                 "MOLISE"
## [139] "MOLISE"                 "MOLISE"                 "MOLISE"
## [142] "MOLISE"                 "MOLISE"                 "MOLISE"
## [145] "CAMPANIA"               "CAMPANIA"               "CAMPANIA"
## [148] "CAMPANIA"               "CAMPANIA"               "CAMPANIA"
## [151] "CAMPANIA"               "CAMPANIA"               "CAMPANIA"
## [154] "CAMPANIA"               "CAMPANIA"               "CAMPANIA"
## [157] "PUGLIA"                 "PUGLIA"                 "PUGLIA"
## [160] "PUGLIA"                 "PUGLIA"                 "PUGLIA"
## [163] "PUGLIA"                 "PUGLIA"                 "PUGLIA"
## [166] "PUGLIA"                 "PUGLIA"                 "PUGLIA"
## [169] "BASILICATA"             "BASILICATA"             "BASILICATA"
## [172] "BASILICATA"             "BASILICATA"             "BASILICATA"
## [175] "BASILICATA"             "BASILICATA"             "BASILICATA"
## [178] "BASILICATA"             "BASILICATA"             "BASILICATA"
## [181] "CALABRIA"               "CALABRIA"               "CALABRIA"
## [184] "CALABRIA"               "CALABRIA"               "CALABRIA"
## [187] "CALABRIA"               "CALABRIA"               "CALABRIA"
## [190] "CALABRIA"               "CALABRIA"               "CALABRIA"
## [193] "SICILIA"                "SICILIA"                "SICILIA"
## [196] "SICILIA"                "SICILIA"                "SICILIA"
## [199] "SICILIA"                "SICILIA"                "SICILIA"
## [202] "SICILIA"                "SICILIA"                "SICILIA"
## [205] "SARDEGNA"               "SARDEGNA"               "SARDEGNA"
## [208] "SARDEGNA"               "SARDEGNA"               "SARDEGNA"
## [211] "SARDEGNA"               "SARDEGNA"               "SARDEGNA"
## [214] "SARDEGNA"               "SARDEGNA"               "SARDEGNA"
## [217] "BOLZANO"                "BOLZANO"                "BOLZANO"
## [220] "BOLZANO"                "BOLZANO"                "BOLZANO"
## [223] "BOLZANO"                "BOLZANO"                "BOLZANO"
## [226] "BOLZANO"                "BOLZANO"                "BOLZANO"
## [229] "TRENTO"                 "TRENTO"                 "TRENTO"
## [232] "TRENTO"                 "TRENTO"                 "TRENTO"
## [235] "TRENTO"                 "TRENTO"                 "TRENTO"
## [238] "TRENTO"                 "TRENTO"                 "TRENTO"
```

```r
# col.ramp = rainbow(20)
col.ramp = hcl.colors(20, palette = "viridis")
col.lab2 = rep(NA, n)
for(i in 1:n)
  col.lab2[i] = col.ramp[which(tourists.label[i,2] ==
                               levels(factor(tourists.label[,2])))]

plot(scores.tourists[,1:2], col=col.lab2, pch=19, xlim=c(-8,30), ylim=c(-3,3.2))
abline(h=-3, v=-8, col=1)
points(scores.tourists[,1], rep(-3, n), col=col.lab2, pch=19)
points(rep(-8, n), scores.tourists[,2], col=col.lab2, pch=19)
abline(h=0, v=0, lty=2, col='grey')
legend('topright', levels(factor(tourists.label[,2])), fill=col.ramp, bty='n', cex = 0.7)
```




```
#! I valori isolati riequilibrano il fatto di avere covarianza nulla (che
#! perché sono indipendenti le PCs). Ma comunque possiamo leggere un
#! trend/correlazione (se appunto
#! togliessimo quelli outliers): all'aumentare di PC1 si vede diminuire PC2
#! Che ci sta: maggior flusso di persone, più ci saranno scelte per posti economici
```

```
##### Homework: try to perform the PCA on the log-transformed data
tourists.mod = tourists
for(i in 1:8) tourists.mod[which(tourists[,i]==0),i] = 1
tourists.log = log(tourists.mod)
head(tourists.log)
```

```
##   Hotel.5stars Hotel.4stars Hotel.3stars Hotel.2stars Hotel.1star Residences
## 1     5.541264     8.653994     8.263075     5.860786    6.222576   7.305860
## 2     5.624018     8.796793     8.246958     5.863631    5.940171   7.388328
## 3     5.605802     8.892611     8.415825     6.259581    6.163315   7.575072
## 4     5.198497     8.577535     8.174703     5.948035    5.799093   7.524561
## 5     5.707110     8.680162     8.080547     5.981414    6.042633   7.625595
## 6     5.556828     8.600799     8.043984     6.122493    5.707110   7.728856
##   Bed.Breakfast Rented.flats
## 1      3.713572     6.311735
## 2      3.433987     6.028279
## 3      3.737670     6.253829
## 4      3.737670     6.295266
## 5      4.442651     6.329721
## 6      4.158883     6.375025
```

3.3 PCA on dataset food

```
##### Principal component analysis of the dataset 'food'

# upload the data
food = read.table('Lab03/Food.txt', header=T)
head(food)
```

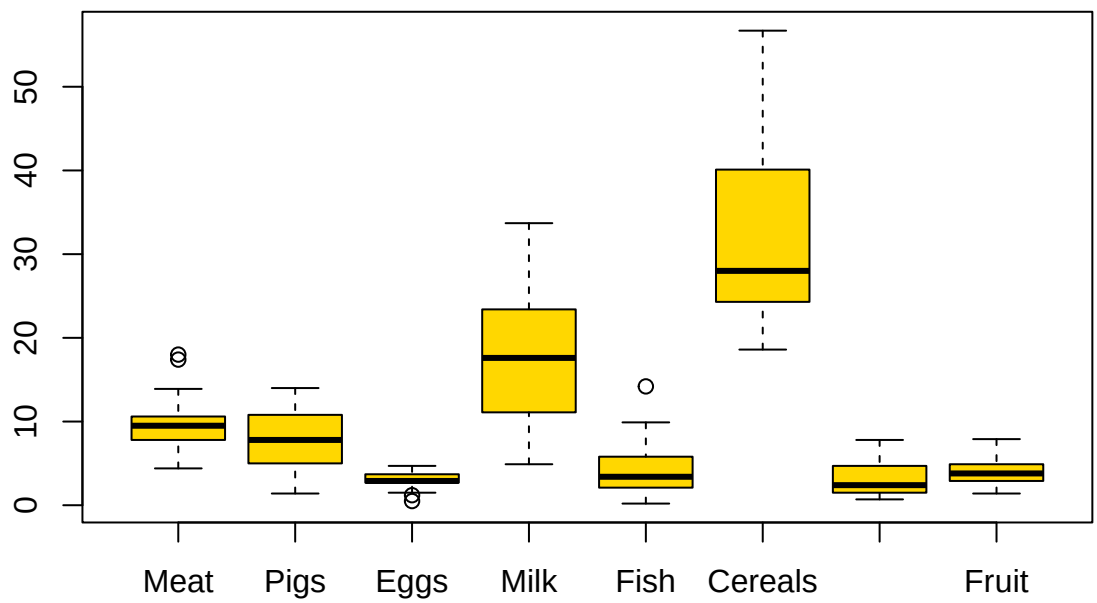
```
##                Meat Pigs Eggs Milk Fish Cereals Pulse Fruit
## Albania        10.1  1.4  0.5  8.9  0.2    42.3   5.5   1.7
```

```
## Austria         8.9 14.0  4.3 19.9  2.1    28.0   1.3   4.3
## Belg.Lux.      13.5  9.3  4.1 17.5  4.5    26.6   2.1   4.0
## Bulgaria        7.8  6.0  1.6  8.3  1.2    56.7   3.7   4.2
## Czechoslovakia  9.7 11.4  2.8 12.5  2.0    34.3   1.1   4.0
## Denmark        10.6 10.8  3.7 25.0  9.9    21.9   0.7   2.4
```

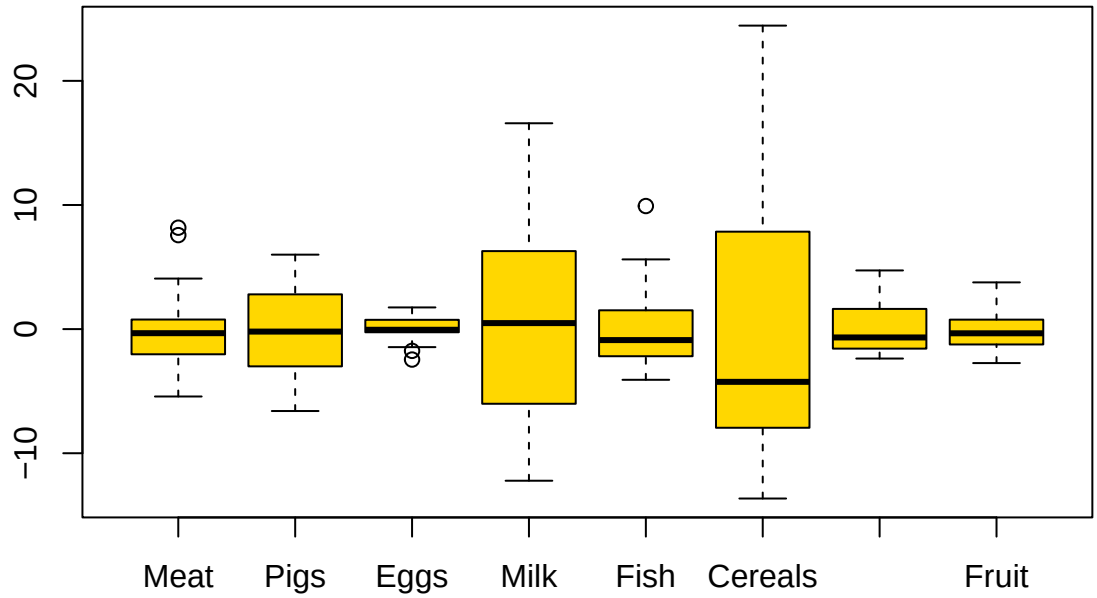
```r
dim(food)
```

```
## [1] 25  8
```

```r
n = dim(food)[1]
p = dim(food)[2]

# exploration
boxplot(food,col='gold')
```




```r
boxplot(scale(x=food, center = T, scale=F), col='gold')
```




```r
S = cov(food)
round(S, digits = 2)
```

```
##           Meat   Pigs  Eggs   Milk   Fish Cereals Pulse Fruit
## Meat     11.20   1.50  2.13  11.96   0.69  -18.36 -2.32 -0.45
## Pigs      1.50  13.01  2.33   6.05  -1.87  -17.34 -4.48  0.00
## Eggs      2.13   2.33  1.19   4.37   0.41   -8.83 -1.22 -0.03
## Milk     11.96   6.05  4.37  50.54   3.36  -46.26 -8.77 -5.24
## Fish      0.69  -1.87  0.41   3.36  11.58  -19.58 -0.99  1.63
## Cereals -18.36 -17.34 -8.83 -46.26 -19.58  120.45 14.19  0.92
## Pulse    -2.32  -4.48 -1.22  -8.77  -0.99   14.19  3.94  1.34
## Fruit    -0.45   0.00 -0.03  -5.24   1.63    0.92  1.34  3.25
```

```r
R = cor(food)
round(R, digits = 2)
```

```
##          Meat  Pigs  Eggs  Milk  Fish Cereals Pulse Fruit
## Meat     1.00  0.12  0.58  0.50  0.06   -0.50 -0.35 -0.07
## Pigs     0.12  1.00  0.59  0.24 -0.15   -0.44 -0.63  0.00
## Eggs     0.58  0.59  1.00  0.56  0.11   -0.74 -0.56 -0.01
## Milk     0.50  0.24  0.56  1.00  0.14   -0.59 -0.62 -0.41
## Fish     0.06 -0.15  0.11  0.14  1.00   -0.52 -0.15  0.27
## Cereals -0.50 -0.44 -0.74 -0.59 -0.52    1.00  0.65  0.05
## Pulse   -0.35 -0.63 -0.56 -0.62 -0.15    0.65  1.00  0.37
## Fruit   -0.07  0.00 -0.01 -0.41  0.27    0.05  0.37  1.00
```

```r
var.gen = det(S)
var.gen
```

```
## [1] 1073887
```

```r
var.tot = sum( diag(S) )
var.tot
```

```
## [1] 215.1638
```

3.3.1 PCA on std variables

```r
##### Principal component analysis of the dataset 'food',
##### based on the correlation matrix
food.sd = scale(food)
food.sd = data.frame(food.sd)

head(food.sd)
```

```
##                       Meat       Pigs       Eggs        Milk        Fish
## Albania         0.08126490 -1.8299828 -2.2437259 -1.15570645 -1.20028213
## Austria        -0.27725673  1.6636208  1.2335002  0.39161231 -0.64187467
## Belg.Lux.       1.09707621  0.3604512  1.0504883  0.05401549  0.06348211
## Bulgaria       -0.60590157 -0.5545403 -1.2371605 -1.24010566 -0.90638347
## Czechoslovakia -0.03824231  0.9427184 -0.1390890 -0.64931122 -0.67126454
## Denmark         0.23064892  0.7763564  0.6844645  1.10900556  1.65053488
##                   Cereals      Pulse       Fruit
## Albania         0.9159176  1.2227536 -1.35040507
## Austria        -0.3870690 -0.8923886  0.09091397
## Belg.Lux.      -0.5146342 -0.4895043 -0.07539207
## Bulgaria        2.2280161  0.3162641  0.03547862
## Czechoslovakia  0.1869740 -0.9931096 -0.07539207
## Denmark        -0.9428885 -1.1945517 -0.96235764
```

```r
sapply(food.sd,mean)
```

```
##          Meat          Pigs          Eggs          Milk          Fish
##  1.796913e-16  5.724587e-19  4.775694e-17  5.744537e-17  6.304419e-17
##       Cereals         Pulse         Fruit
##  2.642418e-16 -1.112825e-17 -6.606911e-17
```

```r
sapply(food.sd,sd)
```

```
##    Meat    Pigs    Eggs    Milk    Fish Cereals   Pulse   Fruit
##       1       1       1       1       1       1       1       1
```

```r
cov(food.sd)
```

```
##                Meat         Pigs        Eggs       Milk        Fish     Cereals
## Meat     1.00000000  0.124155626  0.58236284  0.5025975  0.06095745 -0.49987746
## Pigs     0.12415563  1.000000000  0.59222166  0.2360289 -0.15214735 -0.43819850
## Eggs     0.58236284  0.592221664  1.00000000  0.5619534  0.11150484 -0.73589172
## Milk     0.50259751  0.236028903  0.56195338  1.0000000  0.13874572 -0.59292569
## Fish     0.06095745 -0.152147350  0.11150484  0.1387457  1.00000000 -0.52423080
## Cereals -0.49987746 -0.438198497 -0.73589172 -0.5929257 -0.52423080  1.00000000
## Pulse   -0.34944855 -0.625738013 -0.55997614 -0.6212026 -0.14715294  0.65099727
## Fruit   -0.07422123 -0.000128088 -0.01472764 -0.4086207  0.26613865  0.04654808
##              Pulse        Fruit
## Meat    -0.3494486 -0.074221231
## Pigs    -0.6257380 -0.000128088
## Eggs    -0.5599761 -0.014727635
## Milk    -0.6212026 -0.408620702
## Fish    -0.1471529  0.266138652
## Cereals  0.6509973  0.046548078
## Pulse    1.0000000  0.374969710
## Fruit    0.3749697  1.000000000
```

```r
# PC on correlation matrix
pc.food = princomp(food.sd, scores=T)
pc.food
```

```
## Call:
## princomp(x = food.sd, scores = T)
##
## Standard deviations:
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8
## 1.8861623 1.1828748 1.0381046 0.9126749 0.5616426 0.5182049 0.3489782 0.3267995
##
##  8  variables and  25 observations.
```

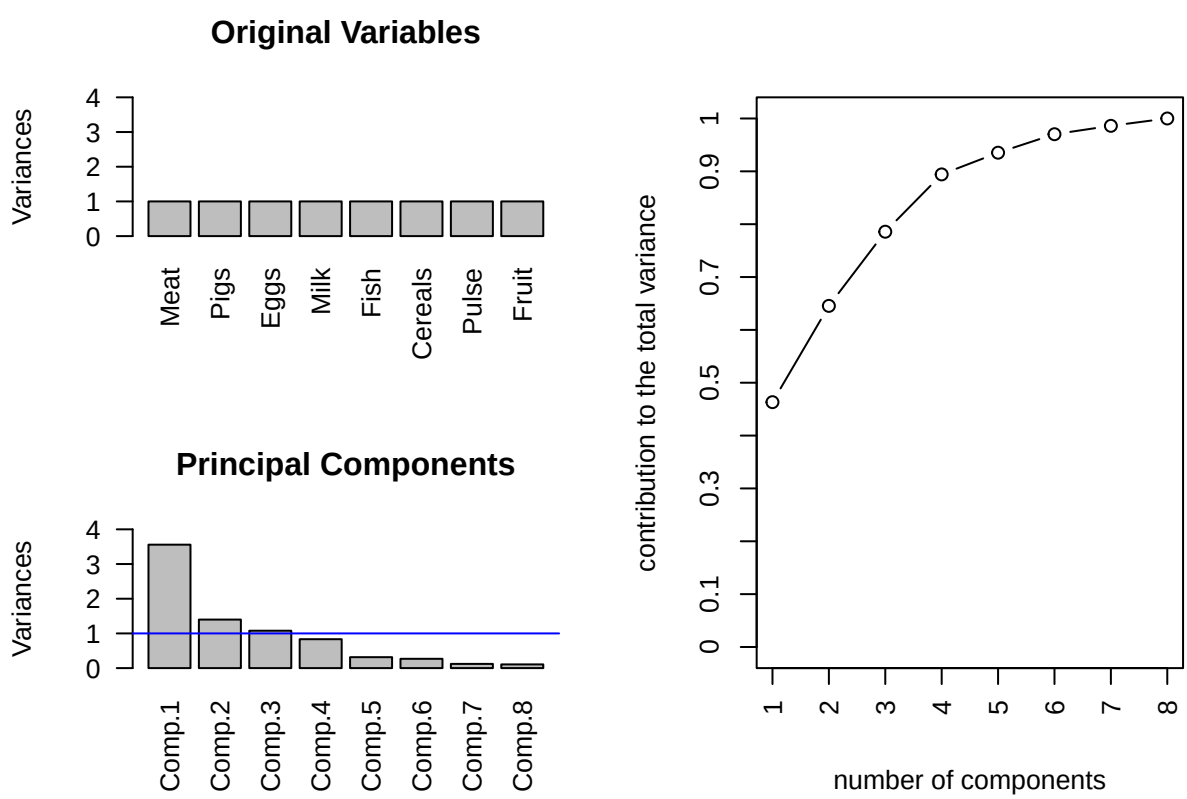
```r
summary(pc.food)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3    Comp.4     Comp.5
## Standard deviation     1.8861623 1.1828748 1.0381046 0.9126749 0.56164261
## Proportion of Variance 0.4632302 0.1821866 0.1403205 0.1084604 0.04107323
## Cumulative Proportion  0.4632302 0.6454168 0.7857372 0.8941976 0.93527083
##                            Comp.6     Comp.7     Comp.8
## Standard deviation     0.51820489 0.34897823 0.32679953
## Proportion of Variance 0.03496567 0.01585753 0.01390598
## Cumulative Proportion  0.97023649 0.98609402 1.00000000
```

```r
#! We have more PCs so that means that the original variables were not really
#correlated
#! between them, as we need "to take more" to explain the variability
```

3.3.2 Explained variance

```r
# explained variance
layout(matrix(c(2,3,1,3), 2, byrow=T))
barplot(pc.food$sdev^2, las=2, main='Principal Components', ylim=c(0,4),
        ylab='Variances')
abline(h=1, col='blue')
barplot(sapply(food.sd,sd)^2, las=2, main='Original Variables', ylim=c(0,4),
        ylab='Variances')
plot(cumsum(pc.food$sdev^2)/sum(pc.food$sde^2), type='b', axes=F,
     xlab='number of components',
     ylab='contribution to the total variance', ylim=c(0,1))
box()
axis(2, at=0:10/10, labels=0:10/10)
axis(1, at=1:ncol(food.sd), labels=1:ncol(food.sd), las=2)
```



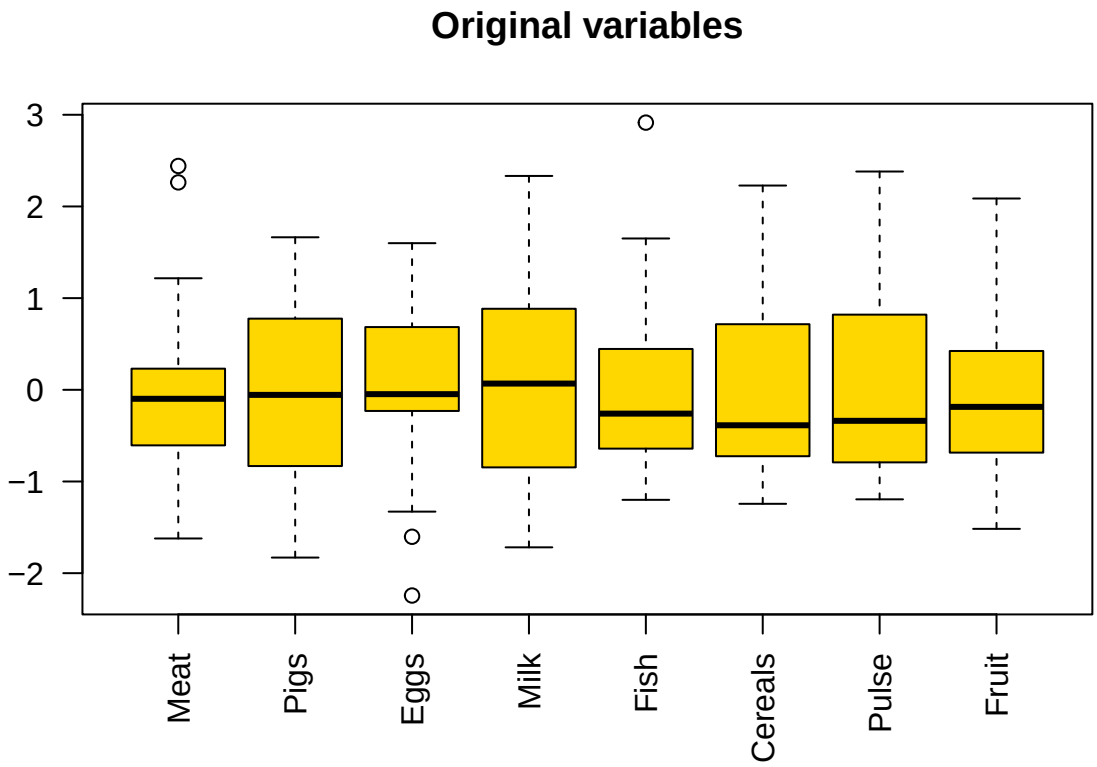

```r
#! Elbow method: stop when there is no more an increase. But also choose
#! where to stop according to usage: just 2 for plotting, or 4,5,ecc (like not 3)
```

3.3.3 Scores

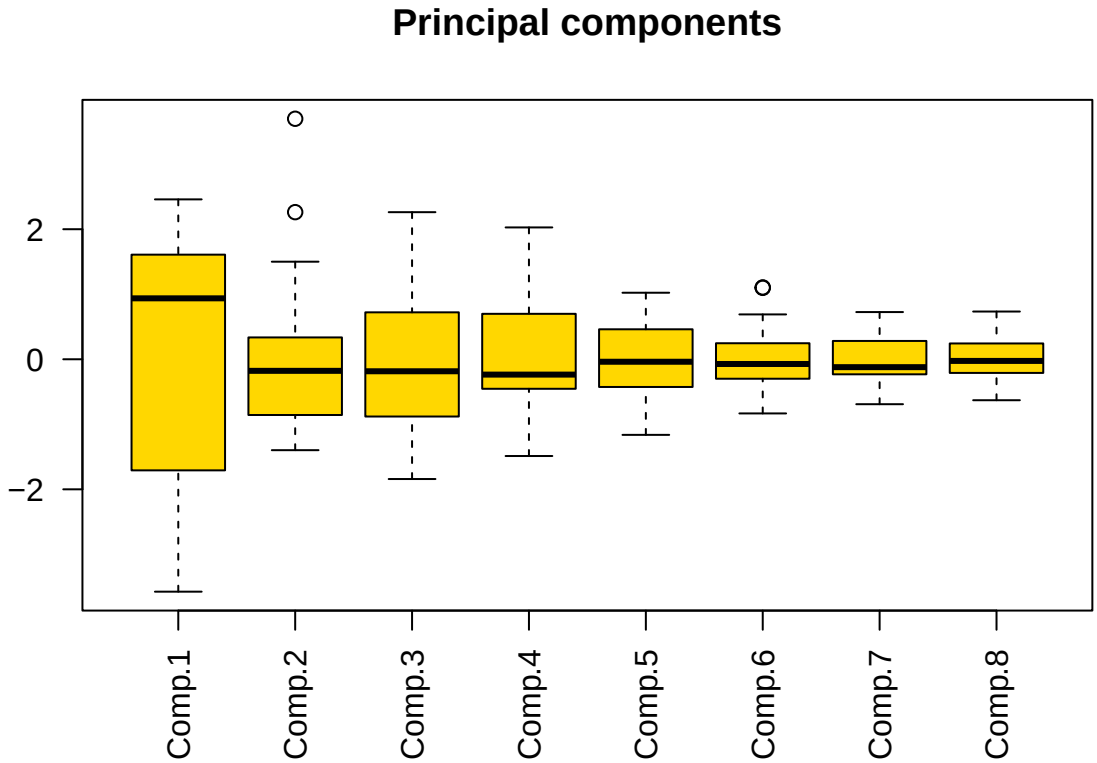
```r
# scores
scores.food = pc.food$scores
scores.food
```

```
##                    Comp.1      Comp.2      Comp.3      Comp.4      Comp.5
## Albania        -2.9433087 -1.39784577  1.59579157  0.51865055  1.02312362
## Austria         1.6088967 -0.65303900 -1.60173367 -0.29576121 -0.41017838
## Belg.Lux.       1.4305750  0.13012471 -0.18457485  0.69882194  0.46094938
## Bulgaria       -2.6987861 -1.01527684 -0.32106745  0.10006012  0.50422643
## Czechoslovakia  0.2291677 -0.77905956 -1.09265804 -0.35739081  0.79542148
## Denmark         2.3581438  0.33593816  0.72155963 -1.25045496  0.10746893
## East.Germany    1.0265458  0.03543671 -1.00547079 -1.06770964  0.81655892
## Finland         1.4946205 -0.91993053  2.26188646 -0.90558398 -0.87271192
## France          1.4578704  1.16770477 -0.26886024  1.72779644  0.78100286
## Greece         -1.9676698  1.50049623  0.65465893  1.44389702 -1.16263980
## Hungary        -1.4992850 -0.86348570 -1.84143689 -0.25670385 -0.44142219
## Ireland         2.4589165 -0.83584085  0.05923610  0.91701056 -0.28053597
## Italy          -1.2248015  0.88280004 -0.38585241  0.70362372 -0.42616209
## Netherlands     1.7289839 -0.68764413 -0.93750485 -0.40396400 -0.46696262
## Norway          0.9371536  0.65744996  1.84863218 -1.16578584  0.02215766
## Poland         -0.2241108  0.20374860 -1.21791396 -0.45404500 -0.74607465
## Portugal       -1.8320943  3.69830072 -0.15062438 -1.48774457  0.64472157
## Romania        -2.6764470 -1.31832638  0.09586213 -0.23579239 -0.03809418
## Spain          -1.7085235  2.26229323 -0.18661730  0.26584692 -0.43829804
## Sweden          1.8125966 -0.03893831  1.17234917 -0.95200940 -0.11517191
## Switzerland     1.2434829 -0.17798963 -0.25512282  0.78600017 -0.20925150
## United.Kingdom  1.7770936  0.21267090  0.96350058  2.02747724  0.38044596
## USSR           -1.2265405 -0.85692030  0.90243405  0.04553400  0.11666926
## West.Germany    2.0111090 -0.19552357 -0.88033615 -0.02701079  0.21611435
## Yugoslavia     -3.5735888 -1.34714345  0.05386298 -0.37476225 -0.26135715
##                     Comp.6       Comp.7      Comp.8
## Albania        -0.21591341  0.725645168 -0.52072792
## Austria        -0.10508561 -0.194017204  0.02950793
## Belg.Lux.      -0.27201570 -0.275370464  0.07699587
## Bulgaria        0.63305623 -0.691012123  0.26475668
## Czechoslovakia  0.37907340 -0.148389240 -0.17071810
## Denmark        -0.07236453 -0.084206447  0.73405976
## East.Germany   -0.54667117 -0.379007798 -0.19586885
## Finland         0.55035304  0.004404454 -0.20997419
## France          1.09832225  0.281809132  0.39781017
## Greece         -0.20546148  0.095672389  0.52216305
## Hungary        -0.83267286  0.449022718  0.35712664
## Ireland        -0.29952589 -0.196990122 -0.02536487
## Italy           0.24411480 -0.367526917 -0.53238524
## Netherlands     0.01605099  0.615309810 -0.03296313
## Norway          0.04777780 -0.121262212 -0.10298441
## Poland          1.10469865 -0.176806171 -0.34388815
## Portugal        0.14791122  0.345563084  0.18981961
## Romania        -0.19090122  0.157180723  0.29773139
## Spain          -0.76638645 -0.231248628 -0.62974580
## Sweden         -0.40316636  0.048682824 -0.02923600
## Switzerland     0.68957665  0.606591520 -0.22310786
## United.Kingdom -0.77649963 -0.305338525  0.08100276
## USSR            0.24644860 -0.319923038  0.06251316
## West.Germany   -0.40571166  0.291793947 -0.23973124
## Yugoslavia     -0.06500764 -0.130576880  0.24320874
```

```r
# variability of the original variables / scores

# layout(matrix(c(1,    2),2))
boxplot(food.sd, las=2, col='gold', main='Original variables')
```




```
scores.food = data.frame(scores.food)
boxplot(scores.food, las=2, col='gold', main='Principal components')
```

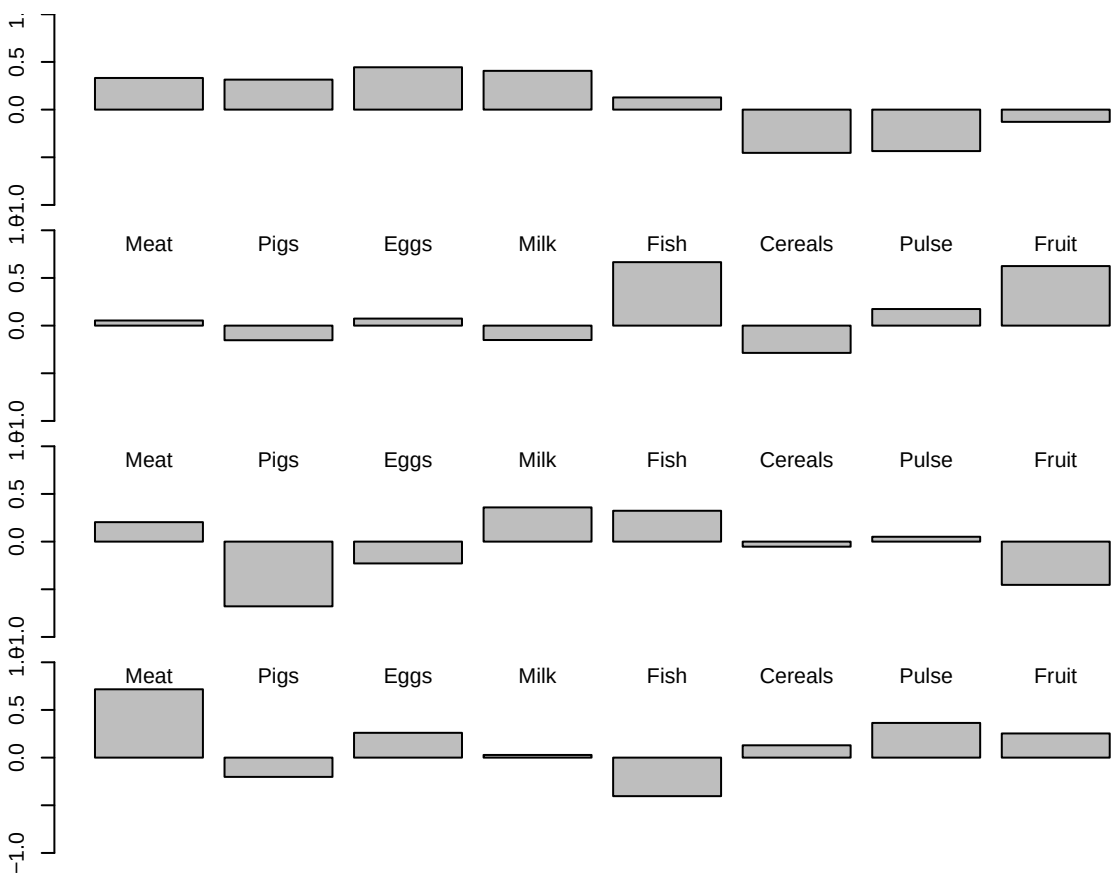


3.3.4 Loadings

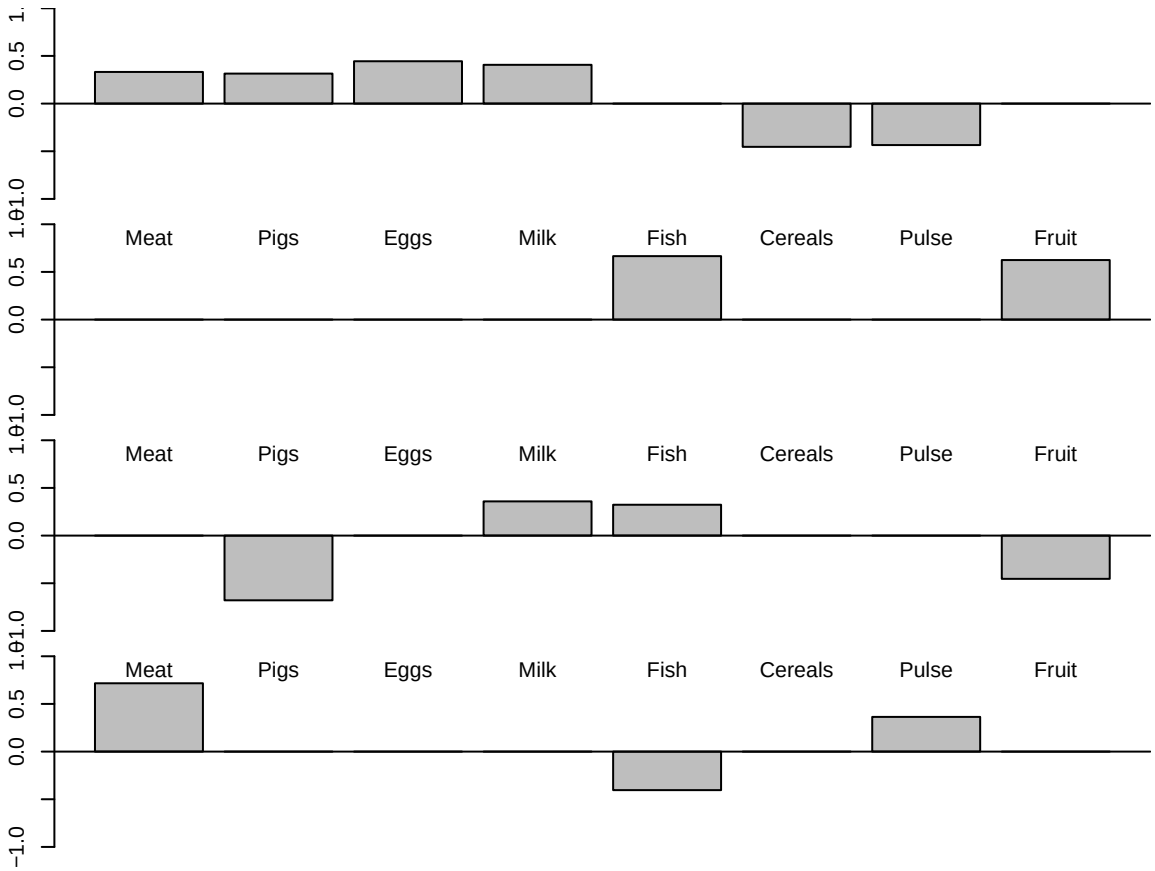
```r
# loadings
load.food    = pc.food$loadings
load.food
```

```
##
## Loadings:
##         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## Meat     0.332         0.204  0.717  0.485  0.197  0.123  0.206
## Pigs     0.315 -0.153 -0.679 -0.202                0.372  0.479
## Eggs     0.444        -0.229  0.260 -0.268 -0.530 -0.566
## Milk     0.407 -0.151  0.359        -0.703  0.385  0.154  0.127
## Fish     0.128  0.666  0.324 -0.404  0.130        -0.114  0.492
## Cereals -0.454 -0.286         0.129         0.322 -0.524  0.558
## Pulse   -0.435  0.174         0.364 -0.345 -0.464  0.460  0.317
## Fruit   -0.128  0.625 -0.454  0.254 -0.235  0.458        -0.234
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
```

```r
par(mar=c(1,4,0,2), mfrow = c(4,1))

for(i in 1:4) {
  barplot(load.food[,i], ylim = c(-1, 1))
}
```



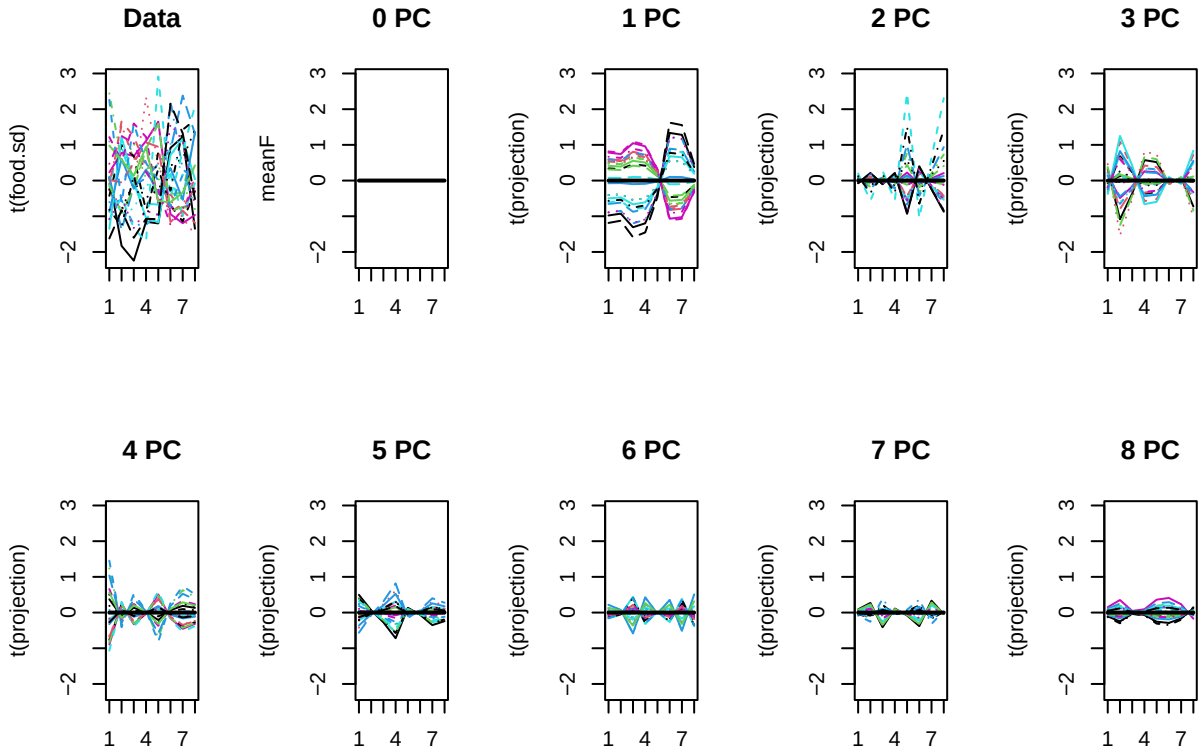

```r
# let's plot only the most significant loadings
par(mar = c(1,4,0,2), mfrow = c(4,1))
for(i in 1:4) {
  barplot(ifelse(abs(load.food[,i]) < 0.3, 0, load.food[,i]),
          ylim = c(-1, 1))
  abline(h=0)
}
```



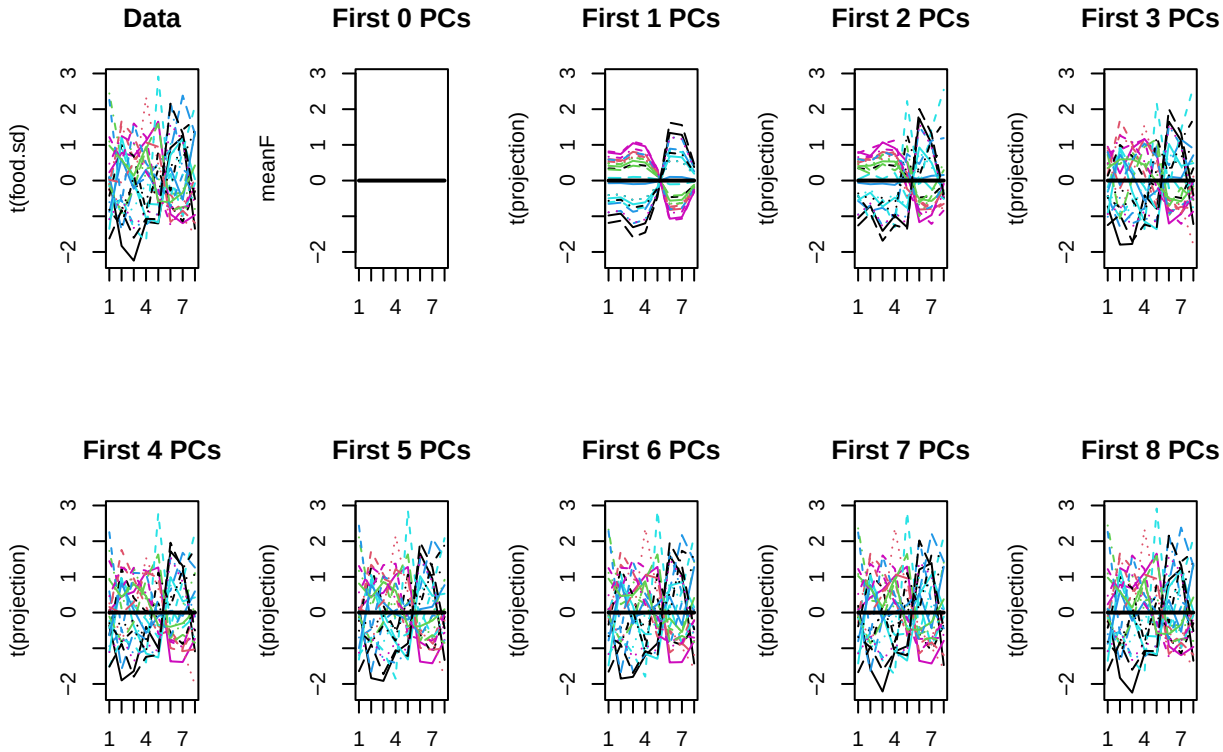

```
#! PC1: contrast between animal proteins and vegetables
#! so high PC1 for diets rich of animal proteins
#! PC2: high for diets with lots of fish and fruit
#! Anyway a switch of sign is permitted, according to our interpretation agility
```

3.3.5 Projection on PCs space

```
# Projection on the space generated by the k-th principal component

par(mfrow=c(2,5))
matplot(t(food.sd), type='l', main='Data', ylim=range(food.sd))

meanF = colMeans(food.sd)
matplot(meanF, type='l', main = '0 PC', lwd=2, ylim=range(food.sd))
for(i in 1:8) {
  projection = matrix(meanF, dim(food.sd)[[1]], dim(food.sd)[[2]], byrow=T) +
    scores.food[,i] %*% t(load.food[,i])
  matplot(t(projection), type='l', main = paste(i, 'PC'), ylim=range(food.sd))
  matplot(meanF, type='l', lwd=2, add=T)
}
```

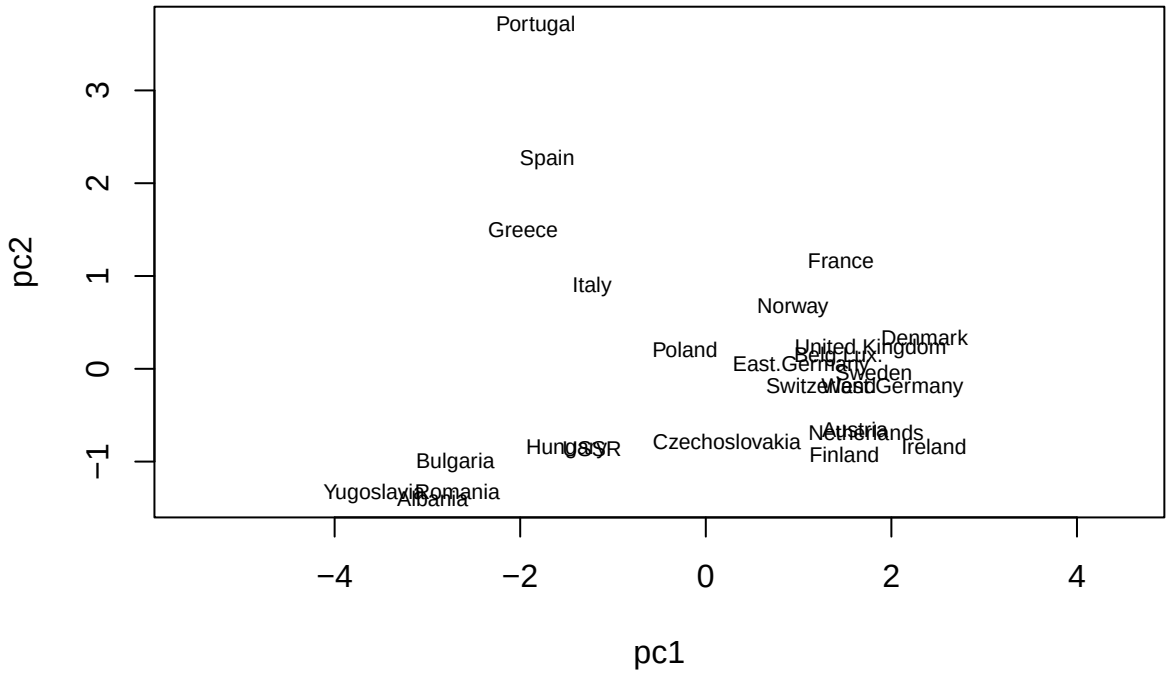



```
#! this section shows how to reconstruct the data through the projections
#! on the PCs, like the decomposition of a signal to send (and the more we
#! select, the more accurate we will be)

# Projection on the space generated by the first k principal components
par(mfrow=c(2,5))
matplot(t(food.sd), type='l', main = 'Data', ylim=range(food.sd))
meanF = colMeans(food.sd)
matplot(meanF, type='l', main = 'First 0 PCs', lwd=2, ylim=range(food.sd))
projection = matrix(meanF, dim(food.sd)[[1]], dim(food.sd)[[2]], byrow=T)
for(i in 1:8)
{
  projection = projection + scores.food[,i] %*% t(load.food[,i])
  matplot(t(projection), type='l', main = paste('First', i, 'PCs'),
        ylim=range(food.sd))
  matplot(meanF, type='l', lwd=2, add=T)
}
```

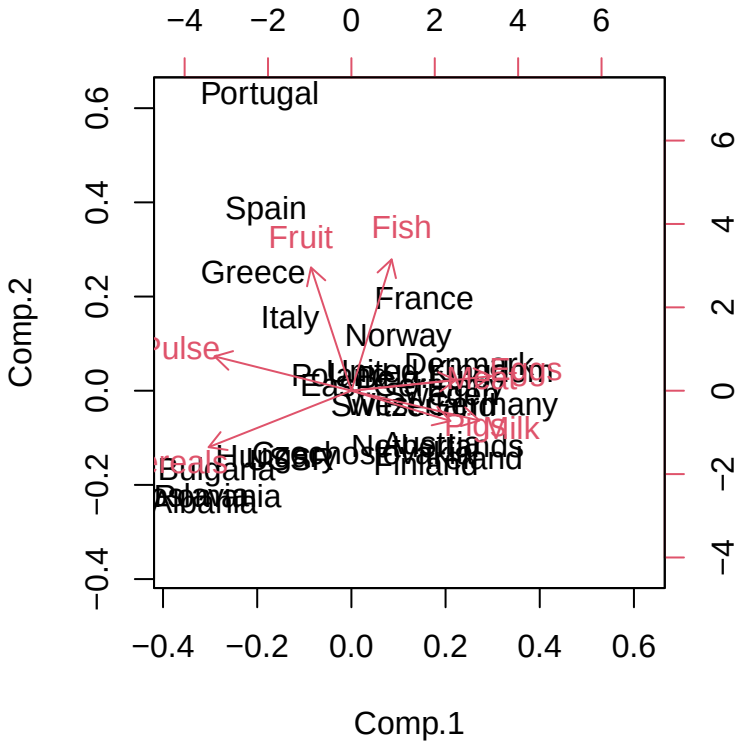


3.3.6 Biplot

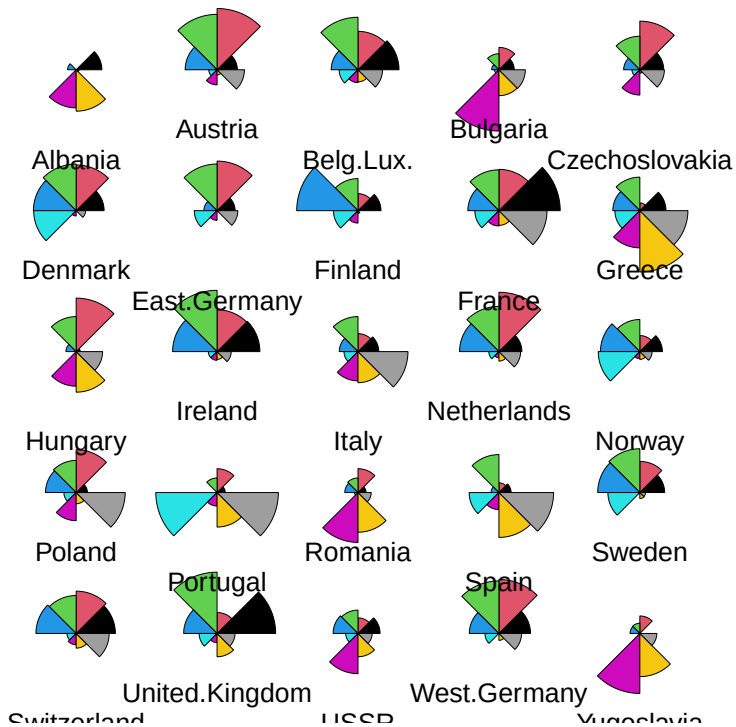
```
# Scores
# par(mfrow=c(1,1))
plot(scores.food[,1], scores.food[,2],type="n", xlab="pc1", ylab="pc2", asp=1,
     xlim=c(-4,3))
text(scores.food[,1], scores.food[,2], dimnames(food)[[1]], cex=0.7)
```




```
biplot(pc.food)
```



```
stars(food.sd, draw.segments=T)
```



3.4 R^p geom view of the PCs

```
### p-dimensional geometrical interpretation of the principal components
library(rgl)
options(rgl.printRglwidget = TRUE)

# theoretical mean and covariance matrix of the model
mu  = c(0, 2, 3)
mu
```

```
## [1] 0 2 3
```

```
sig = rbind(c(9, 1, 1), c(1, 4, 1), c(1, 1, 1))
sig
```

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

```
nobs = 100

X = rmvnorm(nobs, mu, sig)

# sample mean and covariance matrix
M = colMeans(X)
M
```

```
## [1] 0.2635653 1.5331359 2.9361469
```

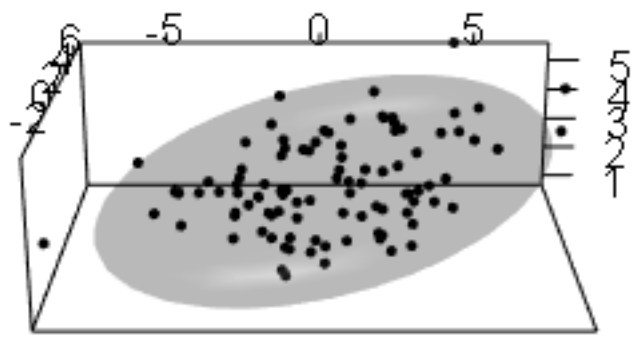
```
S = cov(X)
S
```

```
##          [,1]      [,2]      [,3]
## [1,] 7.386892 1.7961658 0.8694460
## [2,] 1.796166 4.6499368 0.9449611
## [3,] 0.869446 0.9449611 0.8374250
```

```
open3d()                    # open a new
```

```
## wgl
##   4
```

```
points3d(X, asp=1, size=4,)  # plot the points
axes3d()                    # add the axes
plot3d(ellipse3d(S, centre=M, level= 9/10), alpha=0.15, add=TRUE,
       xlab = "asse x") # add the ellipsoid
```



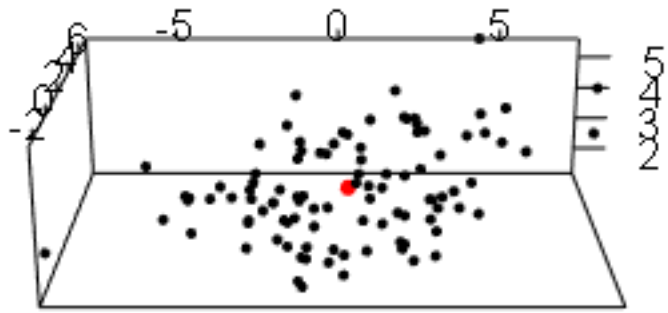

```
# principal components
PC = princomp(X)
summary(PC)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3
## Standard deviation     2.8950407 1.9458341 0.76025001
## Proportion of Variance 0.6575853 0.2970670 0.04534774
## Cumulative Proportion  0.6575853 0.9546523 1.00000000
```

```
# "0" principal component: the best approximation of dimension 0 (a point)
open3d()
```

```
## wgl
##   5
```

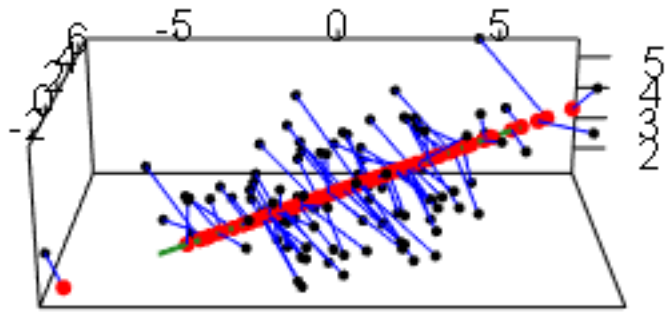
```
points3d(X, asp=1, size=4)
axes3d()

points3d(t(M), col='red', size=6)
```




```r
for(i in 1:100)
  lines3d(rbind(X[i,], M))

# I principal component: the best approximation of dimension 1 (a line)
open3d()
```

```
## wgl
##   6
```

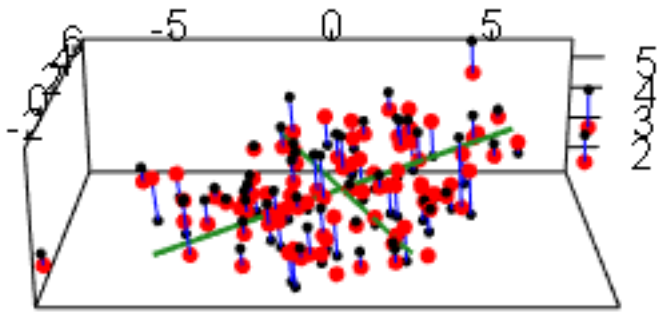
```r
points3d(X, asp=1, size=4)
axes3d()

PC1 = NULL
for(i in 1:nobs) {
  PC1 = rbind(PC1, PC$loadings[,1]*PC$scores[i,1] + M)
}
points3d(PC1, col='red', size=6)

for(i in 1:nobs) {
  lines3d(rbind(X[i,], PC1[i,]), col='blue')
}

lines3d(rbind(M + 2*PC$sdev[1] * PC$loadings[,1],
              M - 2*PC$sdev[1] * PC$loadings[,1]), col='forestgreen',lwd=2)
```




```r
# I and II principal components: the best approximation of dimension 2 (a plane)
open3d()
```

```
## wgl
##   7
```

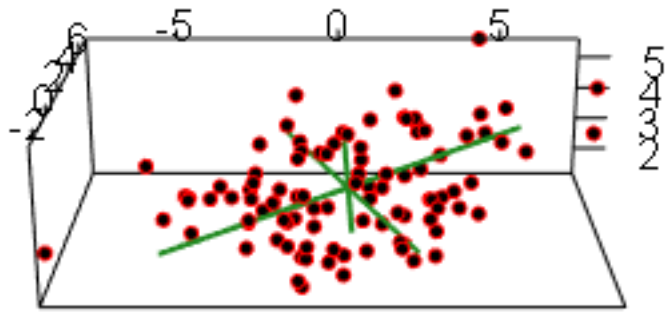
```r
points3d(X, asp=1, size=4)
axes3d()

PC12 = NULL
for(i in 1:nobs) {
  PC12 = rbind(PC12, PC$loadings[,1]*PC$scores[i,1] + PC$loadings[,2]
             *PC$scores[i,2] + M)
}
points3d(PC12, col='red', size=6)

for(i in 1:nobs) {
  lines3d(rbind(X[i,], PC12[i,]),col='blue')
}

lines3d(rbind(M + 2*PC$sdev[1] * PC$loadings[,1], M - 2*PC$sdev[1] *
                PC$loadings[,1]), col='forestgreen',lwd=2)
lines3d(rbind(M + 2*PC$sdev[2] * PC$loadings[,2], M - 2*PC$sdev[2] *
                PC$loadings[,2]), col='forestgreen',lwd=2)
```




```r
# I, II and III principal components: the best approximation of dimension
# 3 (the entire space)
open3d()
```

```
## wgl
##   8
```

```r
points3d(X, asp=1, size=4)
axes3d()

PC123 = NULL
for(i in 1:nobs) {
  PC123 = rbind(PC123, PC$loadings[,1]*PC$scores[i,1] + PC$loadings[,2]*
                PC$scores[i,2] + PC$loadings[,3]*PC$scores[i,3] + M)
}
points3d(PC123, col='red', size=6)

for(i in 1:nobs) {
  lines3d(rbind(X[i,], PC123[i,]),col='blue')
}

lines3d(rbind(M + 2*PC$sdev[1] * PC$loadings[,1], M - 2*PC$sdev[1] *
                PC$loadings[,1]), col='forestgreen',lwd=2)
lines3d(rbind(M + 2*PC$sdev[2] * PC$loadings[,2], M - 2*PC$sdev[2] *
                PC$loadings[,2]), col='forestgreen',lwd=2)
lines3d(rbind(M + 2*PC$sdev[3] * PC$loadings[,3], M - 2*PC$sdev[3] *
                PC$loadings[,3]), col='forestgreen',lwd=2)
```



3.5 Exam exercise

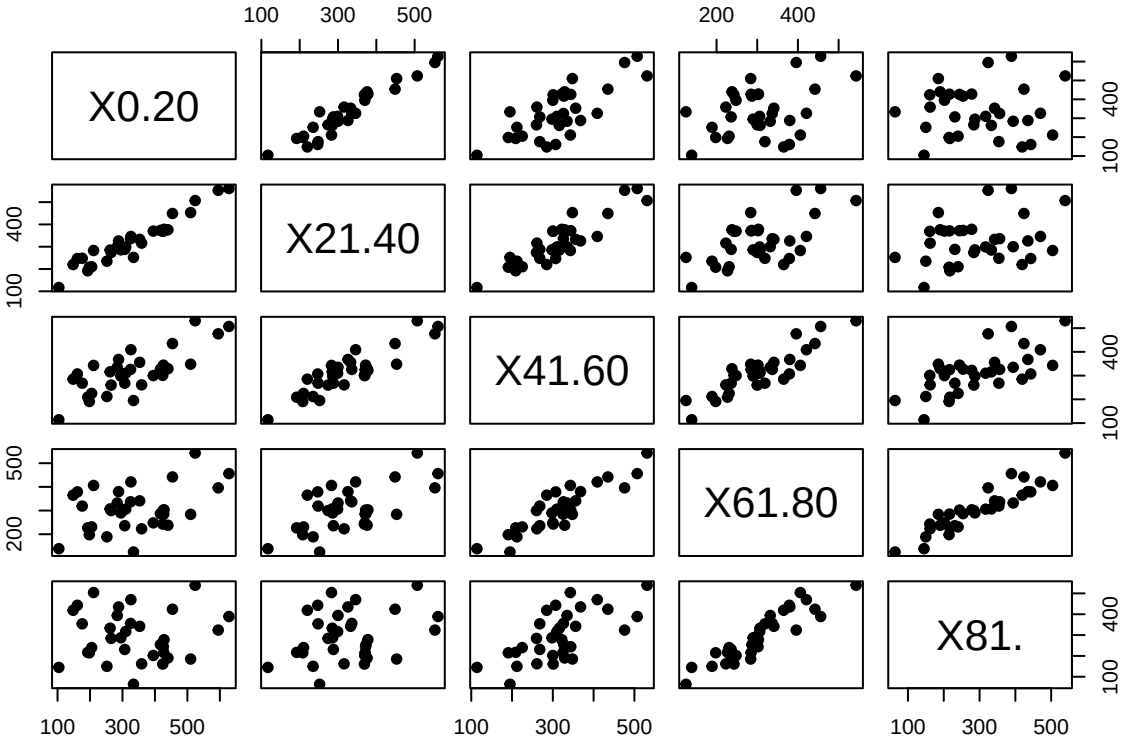
```
##### Example:
##### Question (c) of Problem 3 of the 29/06/2010 exam
## The file scotland.txt collects the number of residents in Scotland, according
## to the last census of 2001, divided by age and county. Assume the data
## associated with different counties to be independent and identically distributed,
## and assume the data corresponding to different age ranges to be dependent.
## Perform a dimensionality reduction of the dataset through a principal component
## analysis and interpret the obtained components

age = read.table('Lab03/scotland.txt', header=T)
head(age)
```

```
##                    X0.20 X21.40 X41.60 X61.80 X81.
## Caithness            193    192    209    227  217
## Sutherland           628    561    507    456  389
## Ross and Cromarty    424    369    301    243  161
## Inverness-shire      307    288    268    236  231
## Nairnshire           176    248    268    319  354
## County of Moray      334    252    195    125   64
```

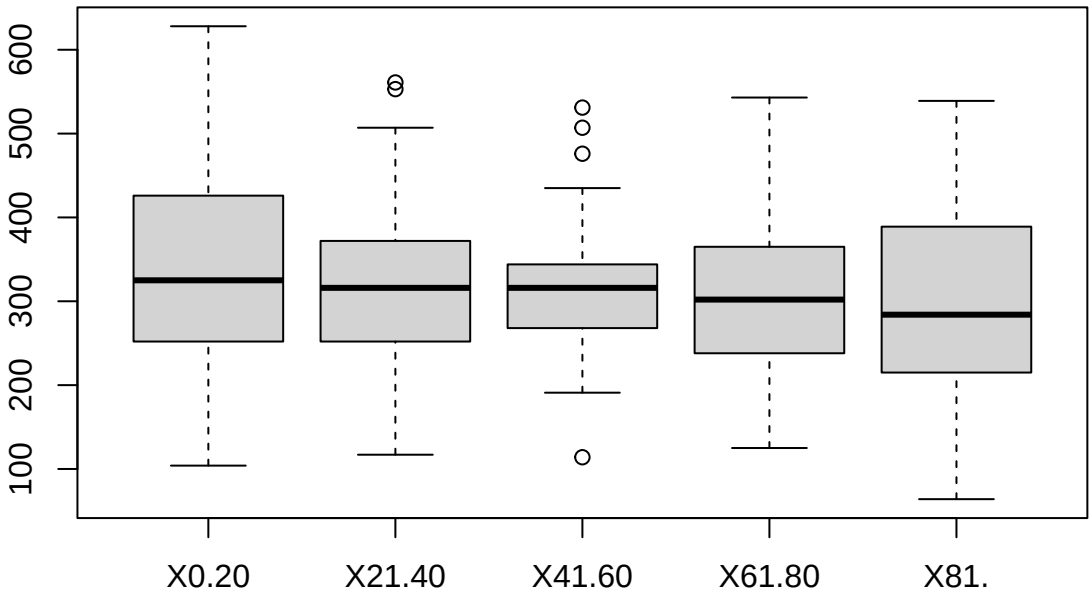
```
dim(age)
```

```
## [1] 33  5
```

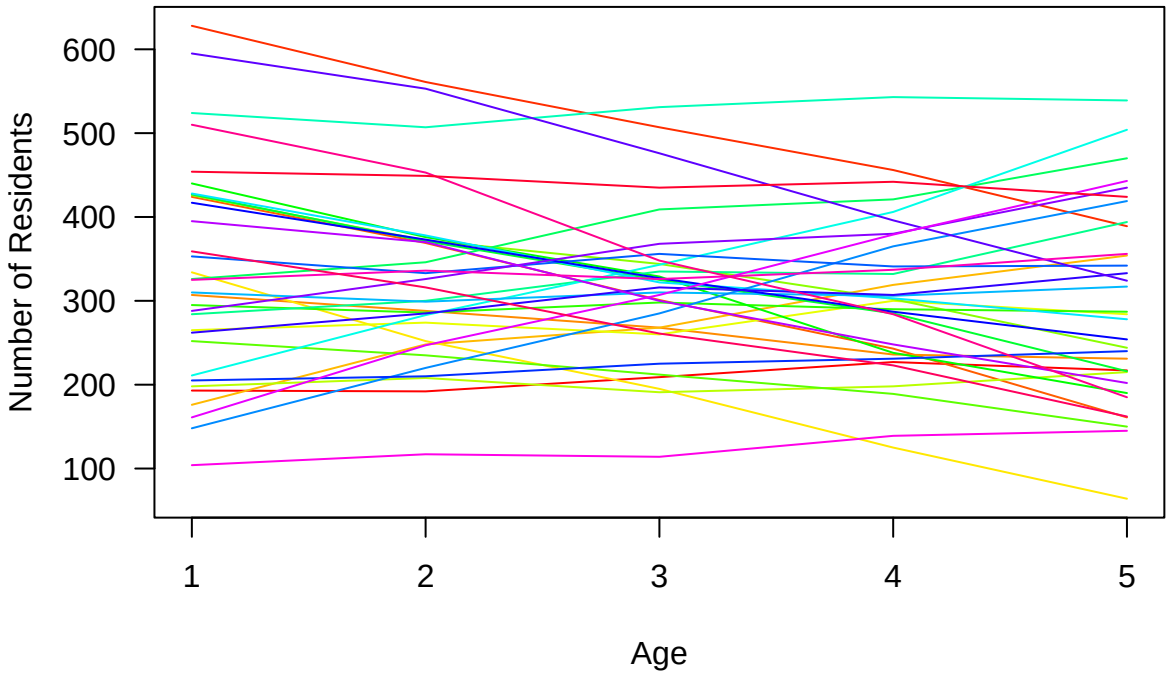
```
pairs(age, pch=19)
```


X0.20
X21.40
X41.60
X61.80
X81.
100
300
500
200
400

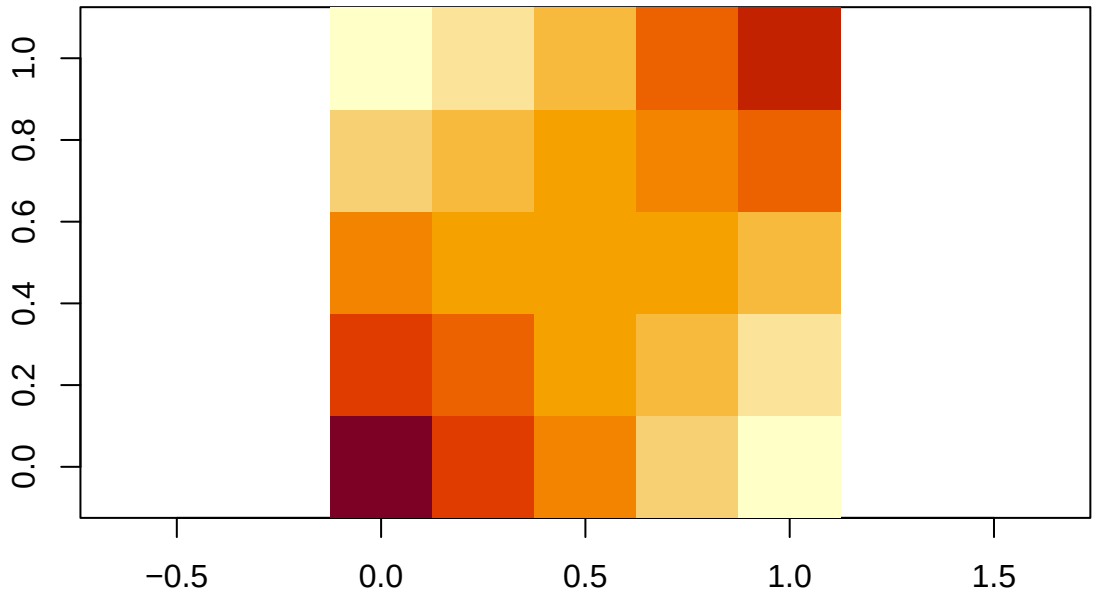
```
boxplot(age)
```


100
200
300
400
500
600
X0.20
X21.40
X41.60
X61.80
X81.

```r
matplot(t(age), type='l', xlab='Age', ylab='Number of Residents', lty=1, col=rainbow(33), las=1)
```




```r
S = cov(age)
image(S, asp=1)
```



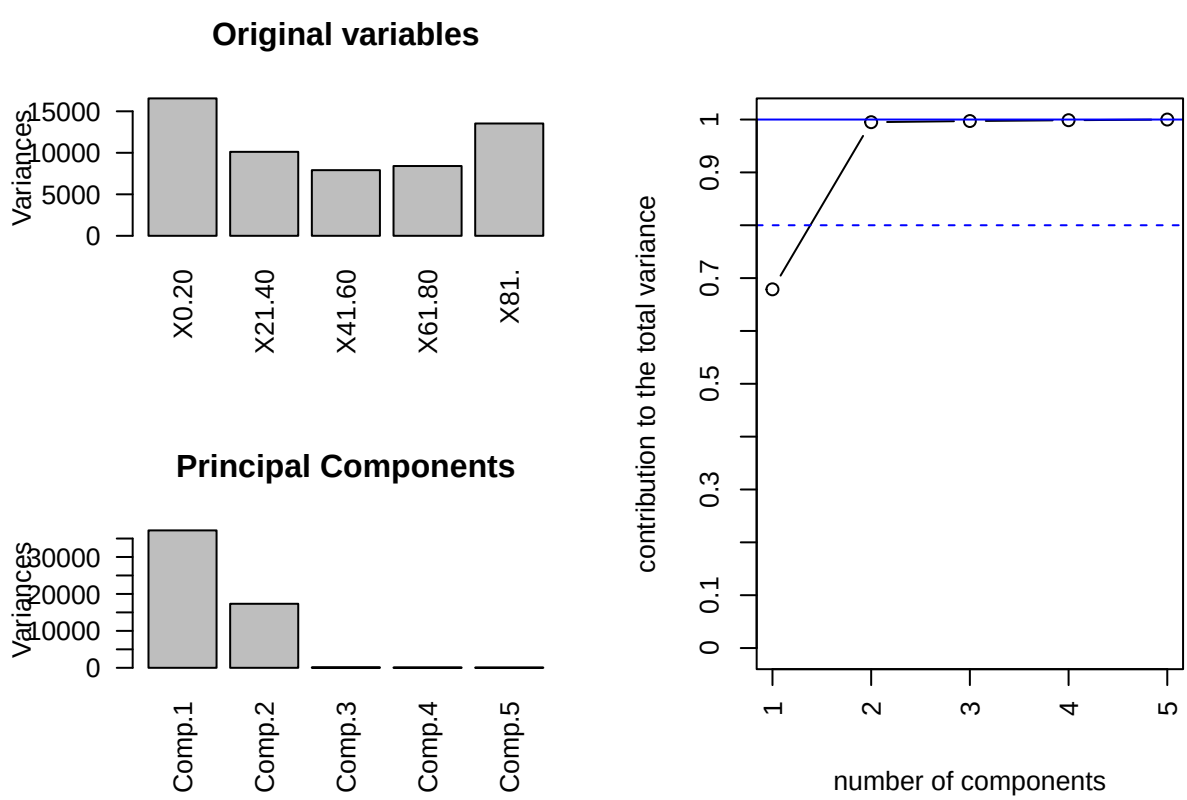
```r
var.gen = det(S)
var.tot = sum( diag(S) )

# PCA (on the covariance matrix)
pc.age = princomp(age, scores=T)
pc.age
```

```
## Call:
## princomp(x = age, scores = T)
##
## Standard deviations:
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5
## 192.869739 131.655545  10.978951   9.277125   8.226957
##
##  5  variables and  33 observations.
```

```r
summary(pc.age)
```

```
## Importance of components:
##                             Comp.1      Comp.2       Comp.3      Comp.4
## Standard deviation     192.8697390 131.6555445 10.978950563 9.277125047
## Proportion of Variance   0.6787322   0.3162631  0.002199338 0.001570352
## Cumulative Proportion    0.6787322   0.9949954  0.997194699 0.998765052
##                             Comp.5
## Standard deviation     8.226957182
## Proportion of Variance 0.001234948
## Cumulative Proportion  1.000000000
```

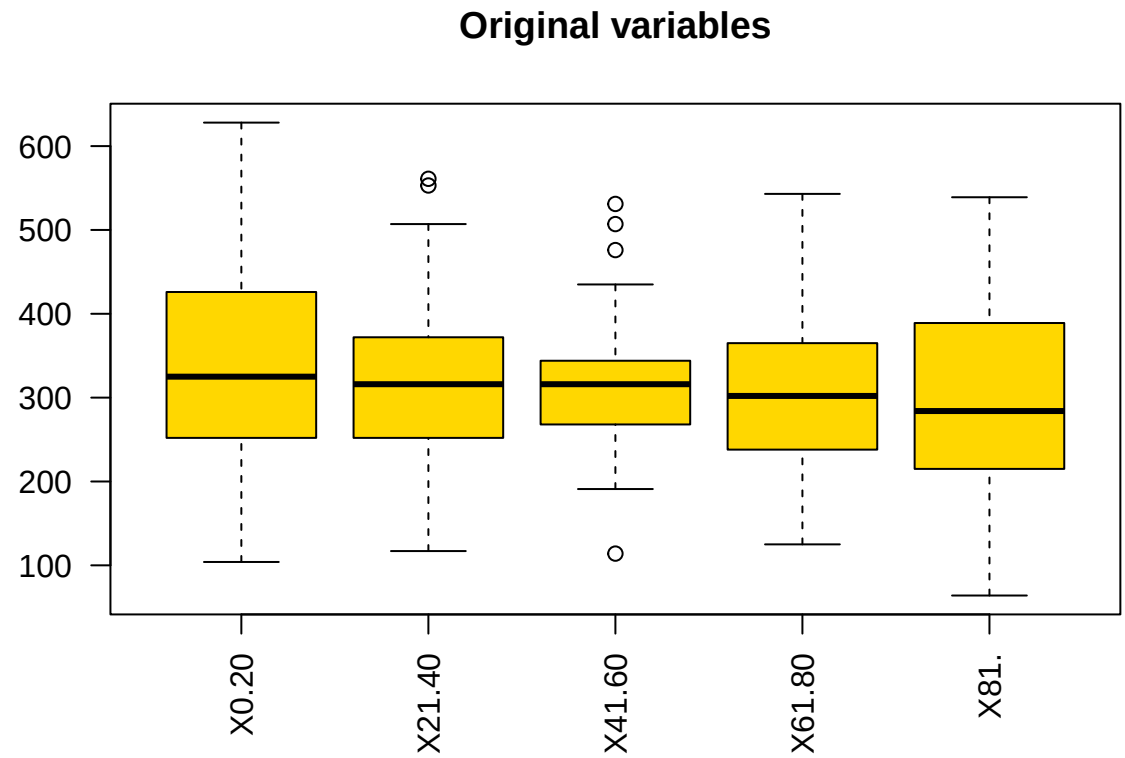
```r
# Explained variance
layout(matrix(c(2,3,1,3),2,byrow=T))
barplot(pc.age$sdev^2, las=2, main='Principal Components', ylab='Variances')
barplot(sapply(age,sd)^2, las=2, main='Original variables', ylab='Variances')
plot(cumsum(pc.age$sdev^2)/sum(pc.age$sde^2), type='b', axes=F, xlab='number of components',
     ylab='contribution to the total variance', ylim=c(0,1))
abline(h=1, col='blue')
abline(h=0.8, lty=2, col='blue')
box()
axis(2,at=0:10/10,labels=0:10/10)
axis(1,at=1:ncol(age),labels=1:ncol(age),las=2)
```



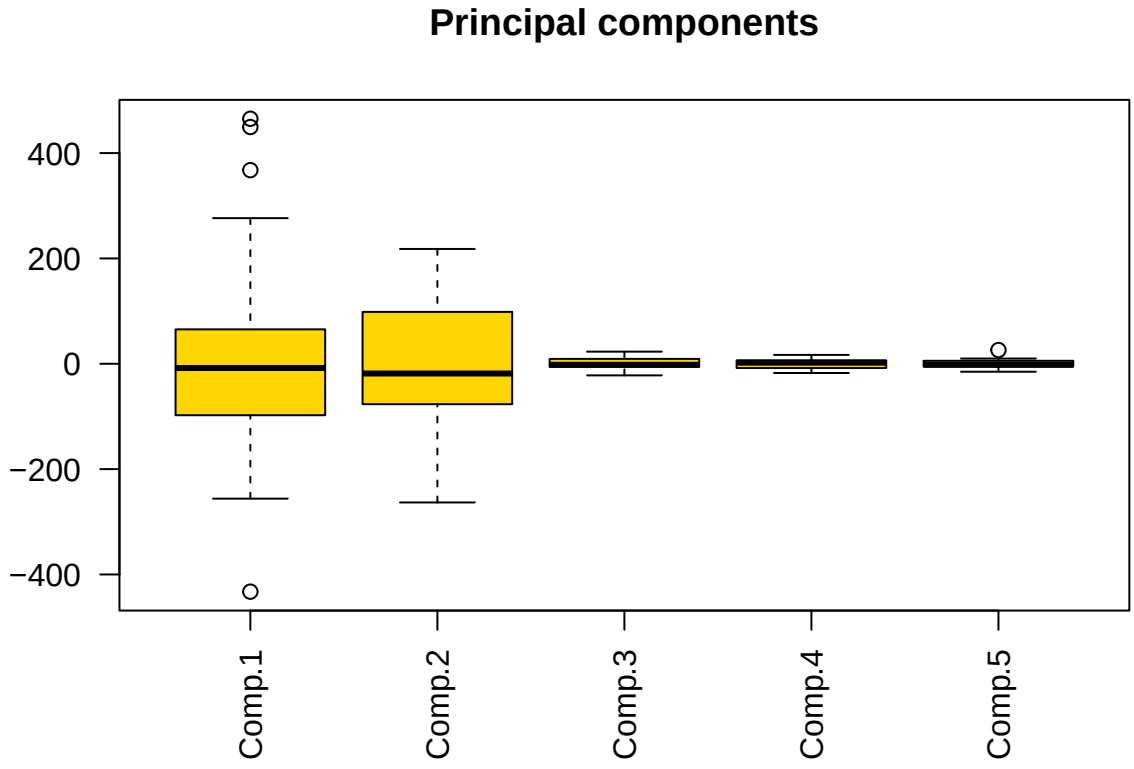
```r
# Scores
scores.age = pc.age$scores
scores.age
```

```
##                        Comp.1       Comp.2       Comp.3     Comp.4      Comp.5
## Caithness          -246.159343  -20.5957665  -4.20309469  16.470275  -2.2983798
## Sutherland          449.508331   89.3364984  -3.39131324   5.517131   5.1684126
## Ross and Cromarty    -8.760094  179.2869377  -7.24695787   2.132629  -6.7331927
## Inverness-shire    -103.769656   49.6231108  10.41990901  -6.790403   0.7991642
## Nairnshire         -117.634818 -149.5014339 -18.34583075 -11.155232  -8.6693862
## County of Moray    -241.551600  218.0233286  14.51053149   6.345341   6.4755616
## Banffshire          -92.871283  -37.0161953 -22.01655762   6.409919   9.6016199
## Aberdeenshire      -256.019922   -0.6817060   0.38915645 -11.790690   9.3818723
## Kincardineshire      65.345624   98.5364577   1.37537862  13.793910  -3.6717541
## Angus              -230.809891   83.5599774  -3.80514820   4.917916  -9.6713017
## Perthshire          -56.815616  -18.4633171   5.71748859   7.895423  -3.8666960
## Argyll               23.917668  169.4180025  19.97341971  -4.604279  -8.5860719
## County of Bute       40.377070  125.9658388  -5.12101513  10.935380  -0.5371047
## Ayrshire            154.214362 -169.7568369  16.01267253   7.322787  -5.3833761
## Renfrewshire         14.566460 -113.9675652  23.05118063  -9.819831   4.3946975
## Dunbartonshire      465.159460 -120.4280512  -1.89684697  16.847771   5.6773894
## Stirlingshire        38.405695 -263.3205605   4.36179018 -13.051649   9.2563518
## Clackmannanshire     71.662632   78.3882758   0.02315265  -4.860474  26.0827681
## Kinross-shire       -20.326873  -34.5731226   9.27402965   4.177613   4.8840019
## Fife                -97.667527 -235.5538628 -13.91845469   7.512622  -3.5511518
## East Lothian         62.360331  -35.5495693  12.66546028  12.191910  -2.6752381
## Midlothian         -214.253923  -28.1224534   3.89663850   5.129224  -1.8884755
## West Lothian         50.911538   93.3508316   4.06959594  -4.100371   7.8739086
## Lanarkshire         -44.150555  -76.7116193  10.72497778  -2.821180 -15.0832451
## Peeblesshire        367.639899  138.5402987  -5.29835417 -15.191489 -14.4035601
## Selkirkshire         77.349095 -153.6243499  10.90374926  -8.049234  -5.6544359
## Berwickshire         -8.015839  132.9063399  -4.72031994 -17.510765  -4.4883246
## Roxburghshire       -53.887531 -244.9842817 -12.76962477  -3.669683  -8.2432087
## Dumfriesshire      -432.649248    0.1093365  -5.83984316   1.495007   5.7562162
## Kirkcudbrightshire   38.596320  -56.6561052  -2.12810188 -10.557615   9.9102939
## Wigtownshire        123.565963  213.1967937 -17.82200877  -9.270367  -2.7236670
## Zetland (Shetland)  -94.716772  139.1773251  -6.43972269   2.083317  -0.6492465
## Orkney              276.480043  -49.9125565 -12.40593676   2.065085   3.5155585
```

```r
# layout(matrix(c(1,2),2))
boxplot(age, las=2, col='gold', main='Original variables')
```



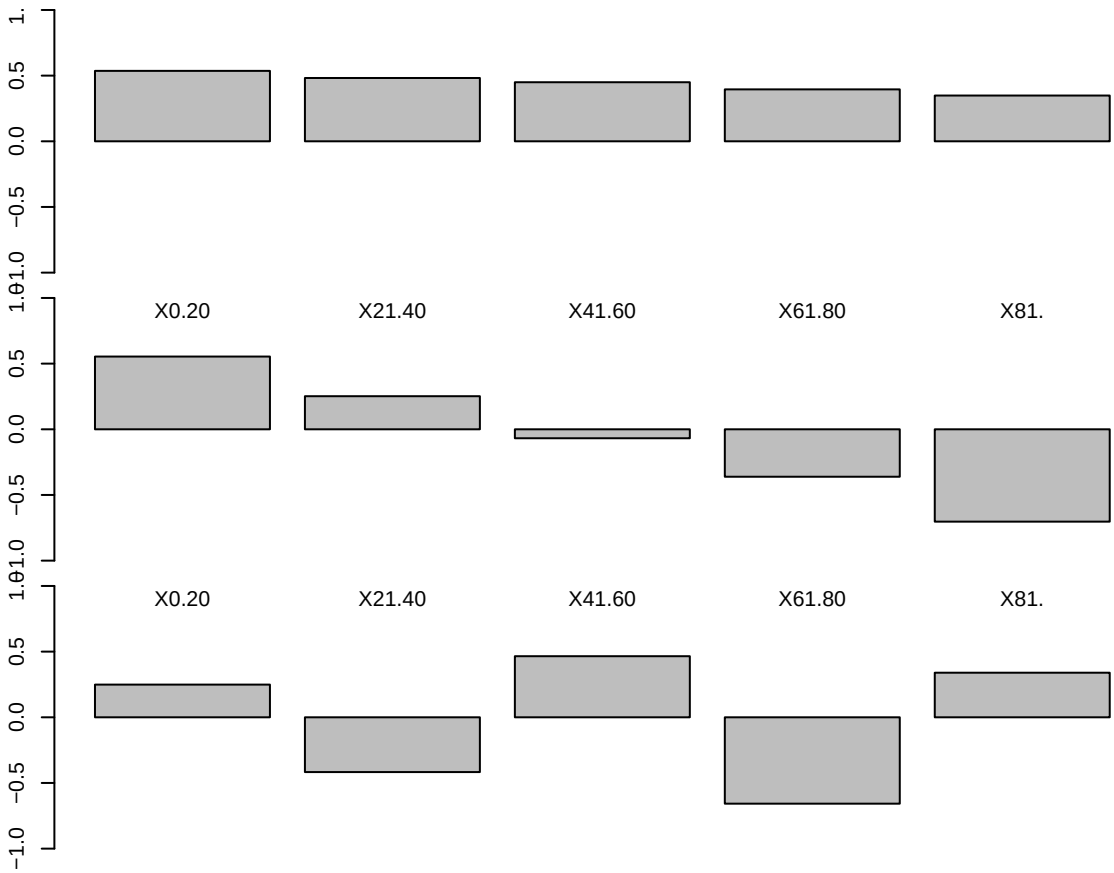

```
scores.age = data.frame(scores.age)
boxplot(scores.age, las=2, col='gold', main='Principal components')
```



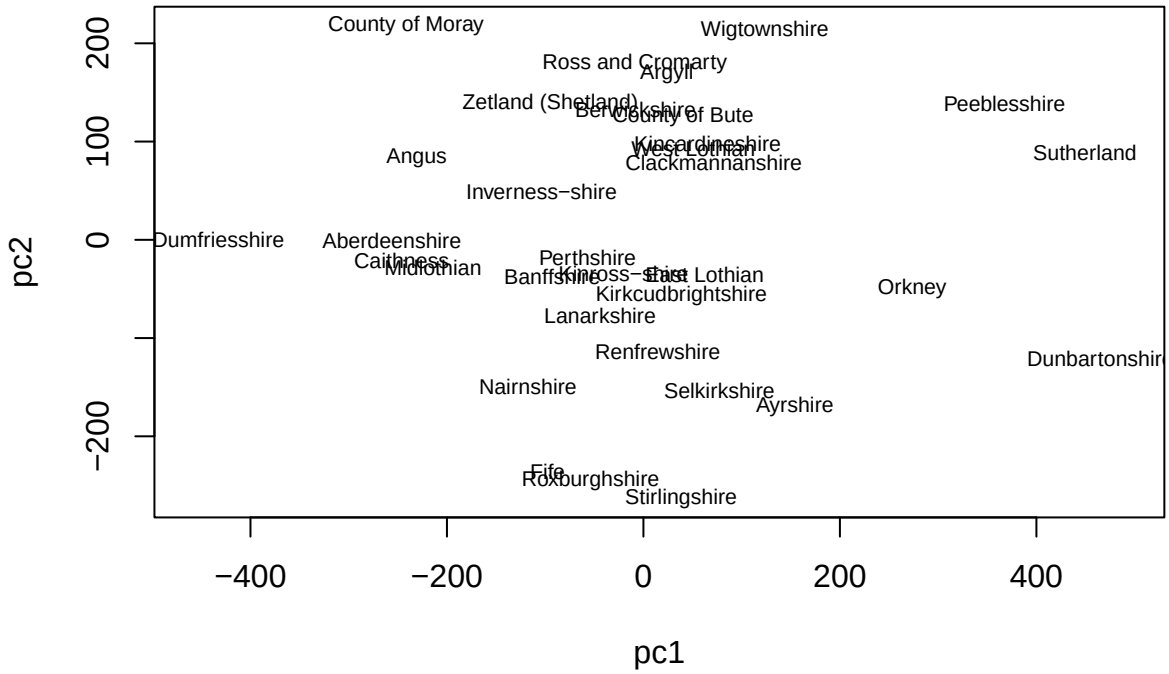
```r
load.age = pc.age$loadings
load.age
```

```
##
## Loadings:
##        Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## X0.20   0.536  0.554  0.249  0.280  0.515
## X21.40  0.482  0.252 -0.417 -0.703 -0.190
## X41.60  0.450         0.465  0.205 -0.731
## X61.80  0.395 -0.361 -0.658  0.530
## X81.    0.349 -0.703  0.339 -0.324  0.405
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## SS loadings       1.0    1.0    1.0    1.0    1.0
## Proportion Var    0.2    0.2    0.2    0.2    0.2
## Cumulative Var    0.2    0.4    0.6    0.8    1.0
```

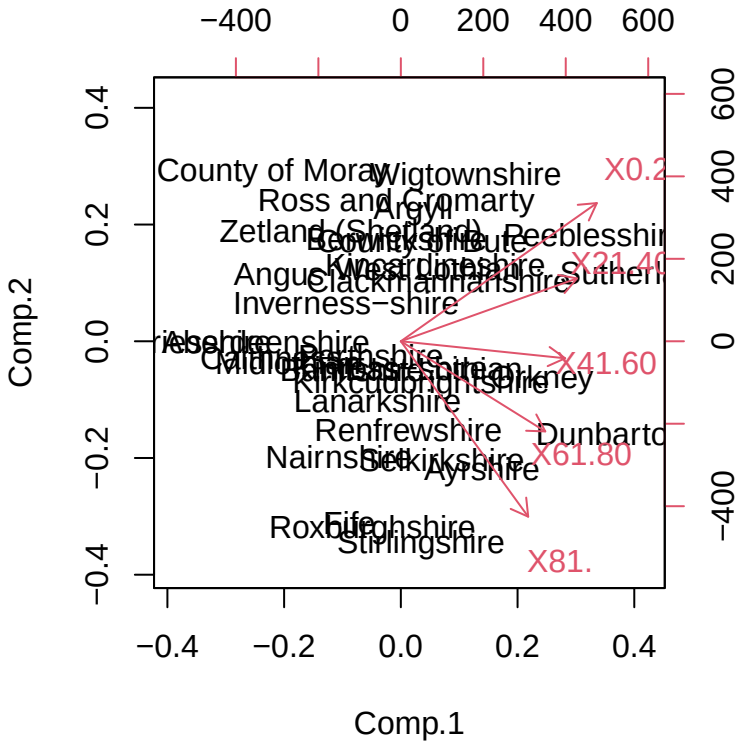
```r
par(mar = c(1,4,0,2), mfrow = c(3,1))
for(i in 1:3)barplot(load.age[,i], ylim = c(-1, 1))
```



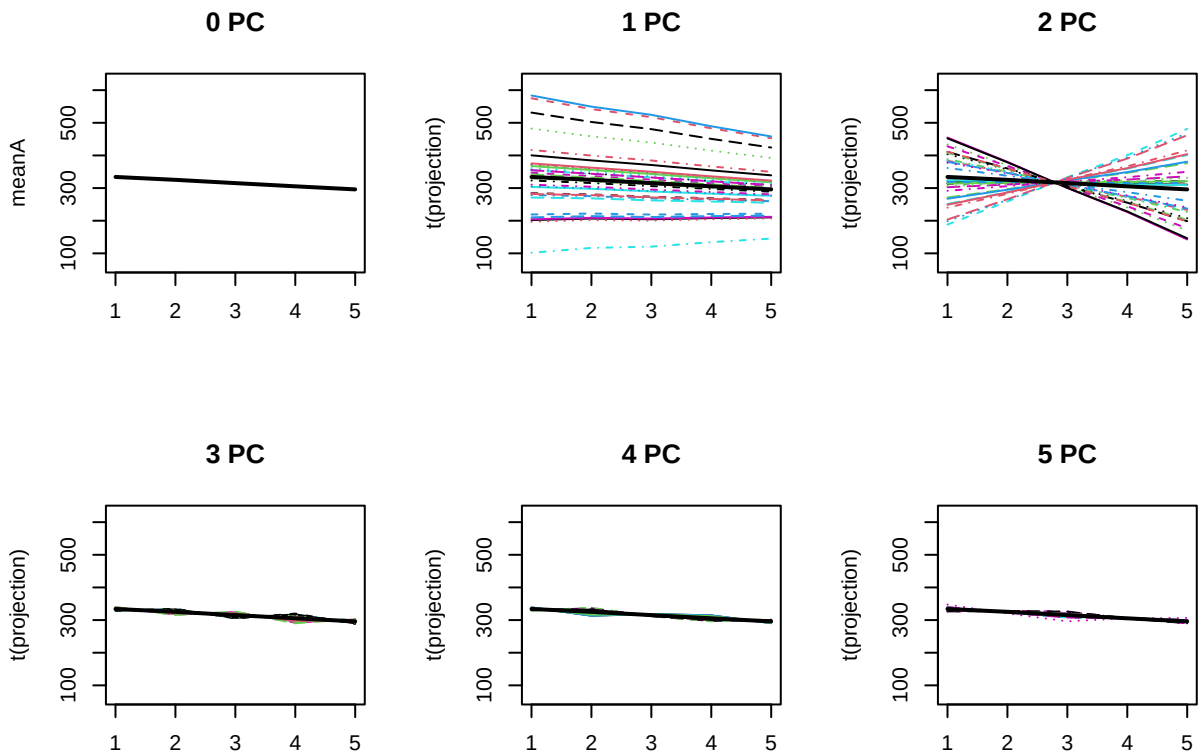

```r
plot(scores.age[,1],scores.age[,2],type="n",xlab="pc1",ylab="pc2", asp=1)
text(scores.age[,1],scores.age[,2],dimnames(age)[[1]], cex=0.7)
```



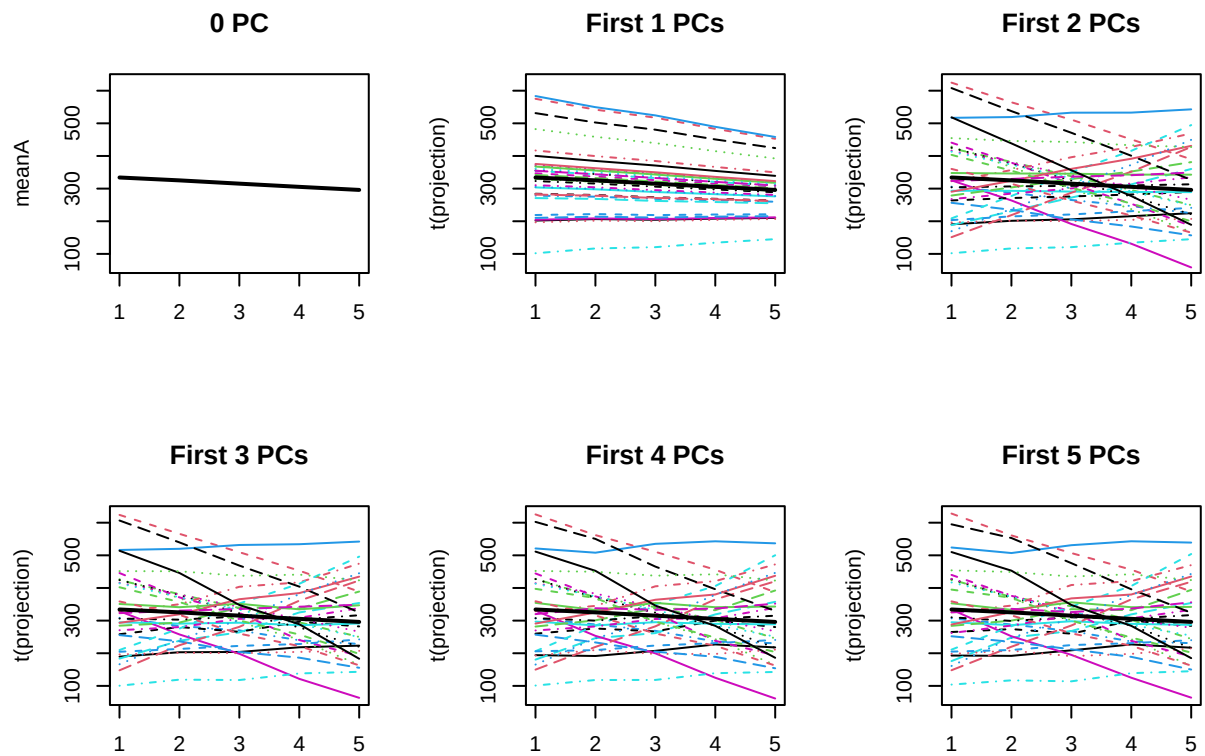

```
biplot(pc.age)
```



```r
# Projection on the space generated by the k-th principal component
par(mfrow=c(2,3))
#matplot(t(age), type='l', main = 'Data', ylim=range(age))
meanA = colMeans(age)
matplot(meanA, type='l', main = '0 PC', lwd=2, ylim=range(age))
for(i in 1:5)
{
  projection = matrix(meanA, dim(age)[[1]], dim(age)[[2]], byrow=T) +
                      scores.age[,i] %*% t(load.age[,i])
  matplot(t(projection), type='l', main = paste(i, 'PC'), ylim=range(age))
  matplot(meanA, type='l', lwd=2, add=T)
}
```

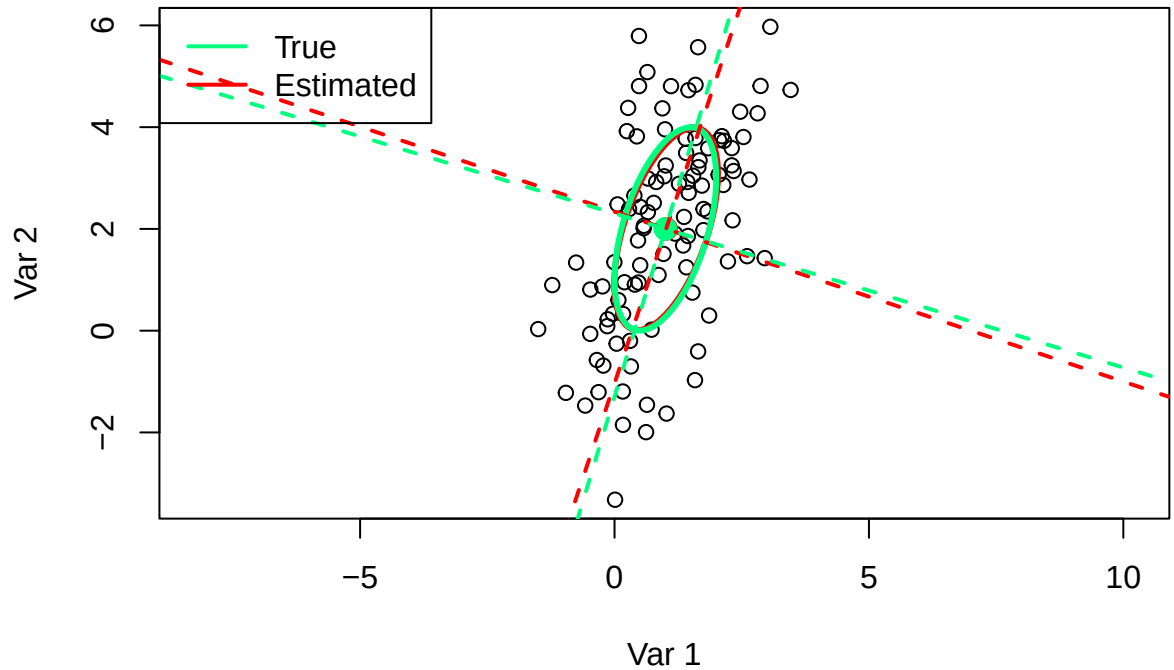



```r
# Projection on the space generated by the first k principal components
par(mfrow=c(2,3))
#matplot(t(age), type='l', main = 'Data', ylim=range(age))
meanA = colMeans(age)
matplot(meanA, type='l', main = '0 PC', lwd=2, ylim=range(age))
projection = matrix(meanA, dim(age)[[1]], dim(age)[[2]], byrow=T)
for(i in 1:5)
{
  projection = projection + scores.age[,i] %*% t(load.age[,i])
  matplot(t(projection), type='l', main = paste('First', i, 'PCs'), ylim=range(age))
  matplot(meanA, type='l', lwd=2, add=T)
}
```

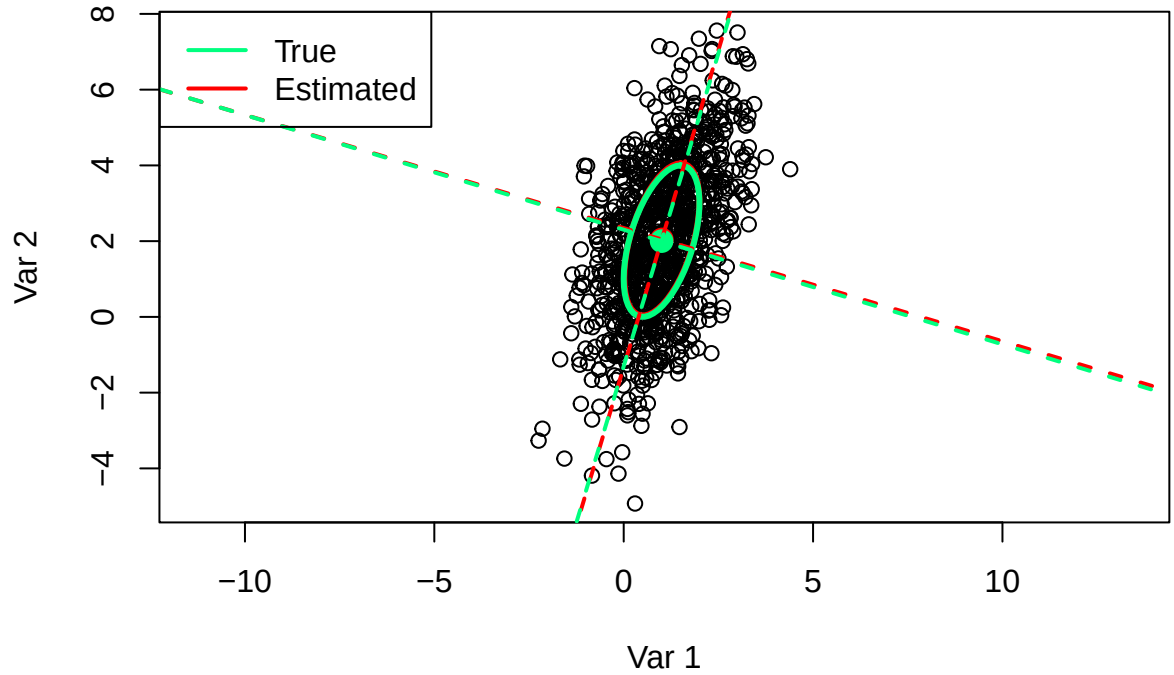



```
#_____________________________________________________________________________
##### Additional execises:

# Problem 1.
# Along the ringroads of Milan four control units measure the concentration of the
# pollutant NO in the air.
# The measures collected during the last year are reported in the file NO.txt.
# Perform a principal component analysis of the available data. In particular:
# (a) Compute the loadings
# (b) Compute the variances along the PCs
# (c) Comment and interpret the results at points (a) and (b)
# (d) On the 3rd July the control unites registered the values (13, 10, 11, 13).
#     Compute the corresponding scores

#_____________________________________________________________________________
##### Additional material: Theoretical & sample PCA

# Let's generate a sample of size 100 from a Gaussian
set.seed(14032022)
mu  =  c(1,2)
sig =  cbind(c(1,1), c(1,4))
n   =  100
X = rmvnorm(n, mu, sig)

M = colMeans(X)
S = cov(X)

plot(X, asp=1, xlab='Var 1', ylab='Var 2',pch=1)
ellipse(M, S, 1, add=T,lwd=3, col = 'red')
ellipse(mu, sig, 1, add=T,lwd=3, col='springgreen')

abline(a = M[2] - eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1]*M[1],
       b = eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1], lty = 2, col = 'red', lwd = 2)
abline(a = M[2] - eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2]*M[1],
       b = eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2], lty = 2, col = 'red', lwd = 2)
abline(a = mu[2] - eigen(sig)$vectors[2,1]/eigen(sig)$vectors[1,1]*mu[1],
       b = eigen(sig)$vectors[2,1]/eigen(sig)$vectors[1,1], lty = 2, col = 'springgreen', lwd = 2)
abline(a = mu[2] - eigen(sig)$vectors[2,2]/eigen(sig)$vectors[1,2]*mu[1],
       b = eigen(sig)$vectors[2,2]/eigen(sig)$vectors[1,2], lty = 2, col = 'springgreen', lwd = 2)
```

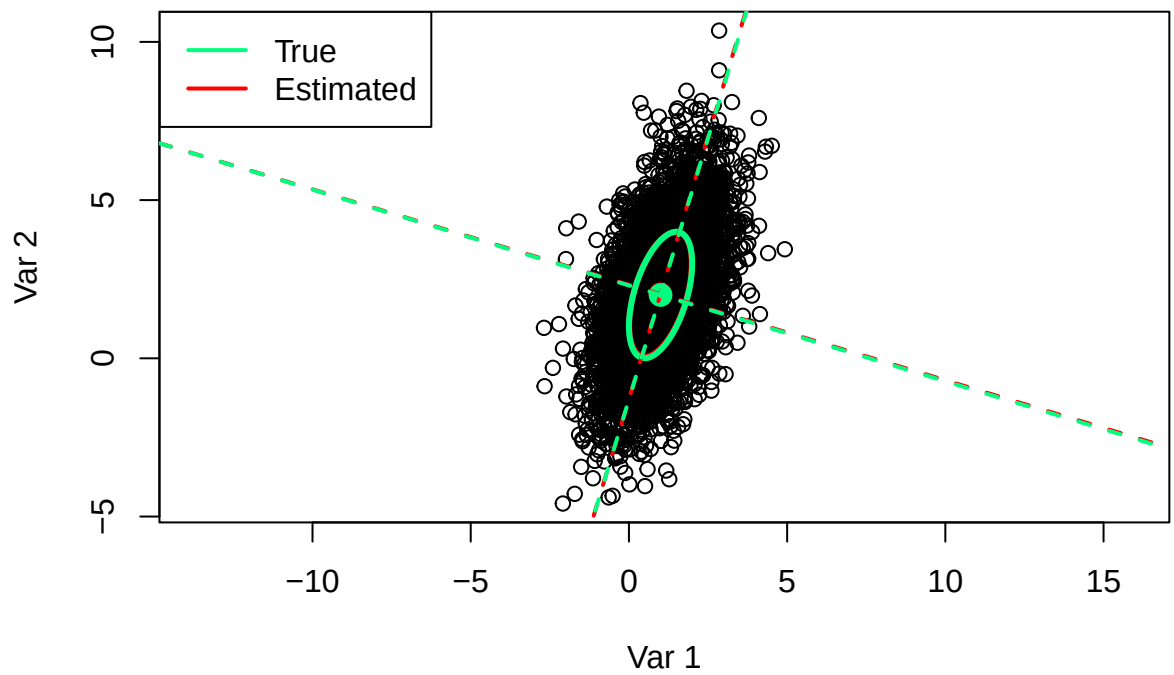
```
legend('topleft',c('True','Estimated'),col=c('springgreen','red'),lty=c(1,1),lwd=2)
```




```
### let's now increase n...
n = 1000

X = rmvnorm(n, mu, sig)
M = colMeans(X)
S = cov(X)

plot(X, asp=1, xlab='Var 1', ylab='Var 2',pch=1)
ellipse(M, S, 1, add=T,lwd=3, col = 'red')
ellipse(mu, sig, 1, add=T,lwd=3, col='springgreen')

abline(a = M[2] - eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1]*M[1],
       b = eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1], lty = 2, col = 'red', lwd = 2)
abline(a = M[2] - eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2]*M[1],
       b = eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2], lty = 2, col = 'red', lwd = 2)
abline(a = mu[2] - eigen(sig)$vectors[2,1]/eigen(sig)$vectors[1,1]*mu[1],
       b = eigen(sig)$vectors[2,1]/eigen(sig)$vectors[1,1], lty = 2, col = 'springgreen', lwd = 2)
abline(a = mu[2] - eigen(sig)$vectors[2,2]/eigen(sig)$vectors[1,2]*mu[1],
       b = eigen(sig)$vectors[2,2]/eigen(sig)$vectors[1,2], lty = 2, col = 'springgreen', lwd = 2)

legend('topleft',c('True','Estimated'),col=c('springgreen','red'),lty=c(1,1),lwd=2)
```




```
###
n = 5000

X = rmvnorm(n, mu, sig)
M = colMeans(X)
S = cov(X)

plot(X, asp=1, xlab='Var 1', ylab='Var 2',pch=1)
ellipse(M, S, 1, add=T,lwd=3, col = 'red')
ellipse(mu, sig, 1, add=T,lwd=3, col='springgreen')

abline(a = M[2] - eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1]*M[1],
       b = eigen(S)$vectors[2,1]/eigen(S)$vectors[1,1], lty = 2, col = 'red', lwd = 2)
abline(a = M[2] - eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2]*M[1],
       b = eigen(S)$vectors[2,2]/eigen(S)$vectors[1,2], lty = 2, col = 'red', lwd = 2)
abline(a = mu[2] - eigen(sig)$vectors[2,1]/eigen(sig)$vectors[1,1]*mu[1],
       b = eigen(sig)$vectors[2,1]/eigen(sig)$vectors[1,1], lty = 2, col = 'springgreen', lwd = 2)
abline(a = mu[2] - eigen(sig)$vectors[2,2]/eigen(sig)$vectors[1,2]*mu[1],
       b = eigen(sig)$vectors[2,2]/eigen(sig)$vectors[1,2], lty = 2, col = 'springgreen', lwd = 2)

legend('topleft',c('True','Estimated'),col=c('springgreen','red'),lty=c(1,1),lwd=2)
```



4 Lab 04 - About normality assumption

```
###--------------------###
### LAB 4 (20/03/2023) ###
###--------------------###

### TOPIC:
### Multivariate normality tests
### Box-Cox transformations
### Tests and confidence regions for the mean of a multivariate Gaussian
#_______________________________________________________________________________


#! MVN stands proprio per multivariate normal
library(MVN) # Can need separate installation of "gsl" software library
library(car)
library(mvtnorm)
library(mvnormtest)
```

4.1 On simulated data

```
### Test for multivariate normality
# We generate a sample of n=150 observation from bivariate Gaussian

mu = c(1,2)
mu
```

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

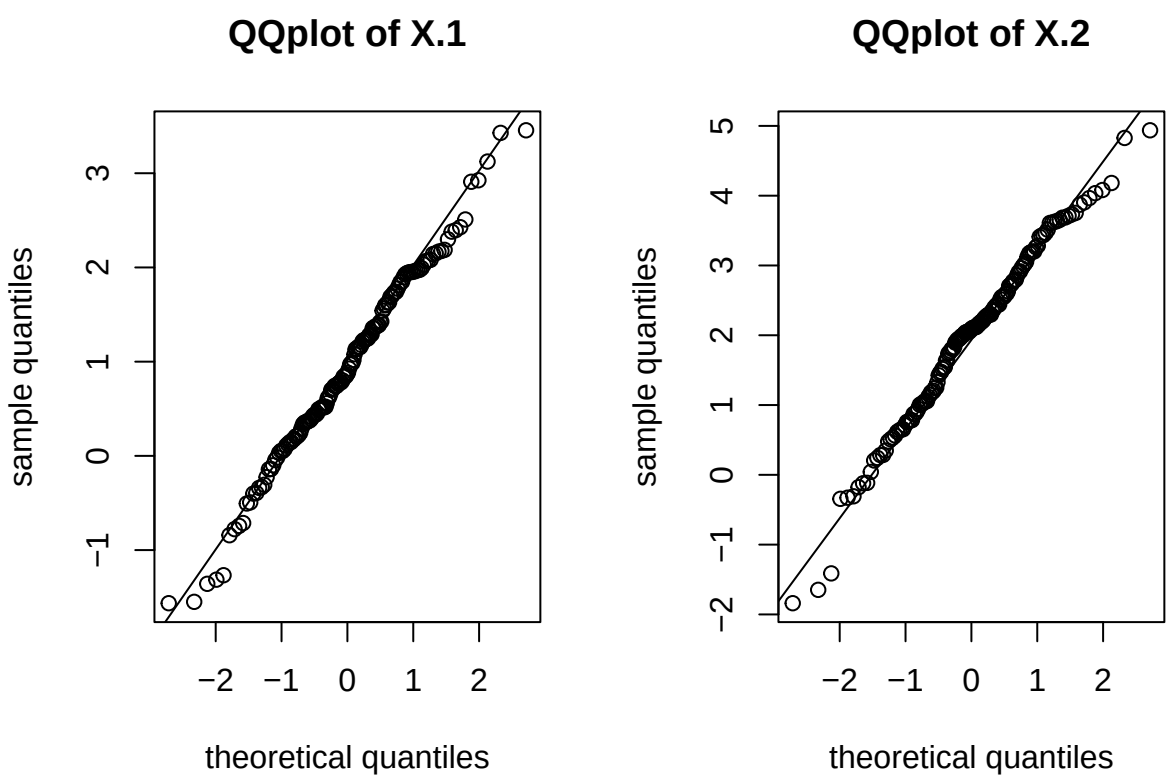
```
sig = matrix(c(1,1,1,2), 2)
sig
```

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

```r
n = 150

set.seed(20230320)
#! to uniform the results between different people doing this simulation
X = rmvnorm(n, mu, sig)

### mvnormtest package ###

# Multivariate version of the Shapiro-Wilk test
# To be used with sample sizes *between 3 and 5000*
# Takes a matrix where each observation is a column
mshapiro.test(t(X)) # Don't forget to transpose your matrix before !!
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.9925, p-value = 0.6213
```

```r
#! mshapiro for multivariate shapiro
#! high pvalue => we accept H0, our sample is gaussian

# Univariate Q-Q plots

par(mfrow=c(1,2))
qqnorm(X[,1], main='QQplot of X.1',xlab='theoretical quantiles',
       ylab='sample quantiles')
qqline(X[,1])
qqnorm(X[,2], main='QQplot of X.2',xlab='theoretical quantiles',
       ylab='sample quantiles')
qqline(X[,2])
```



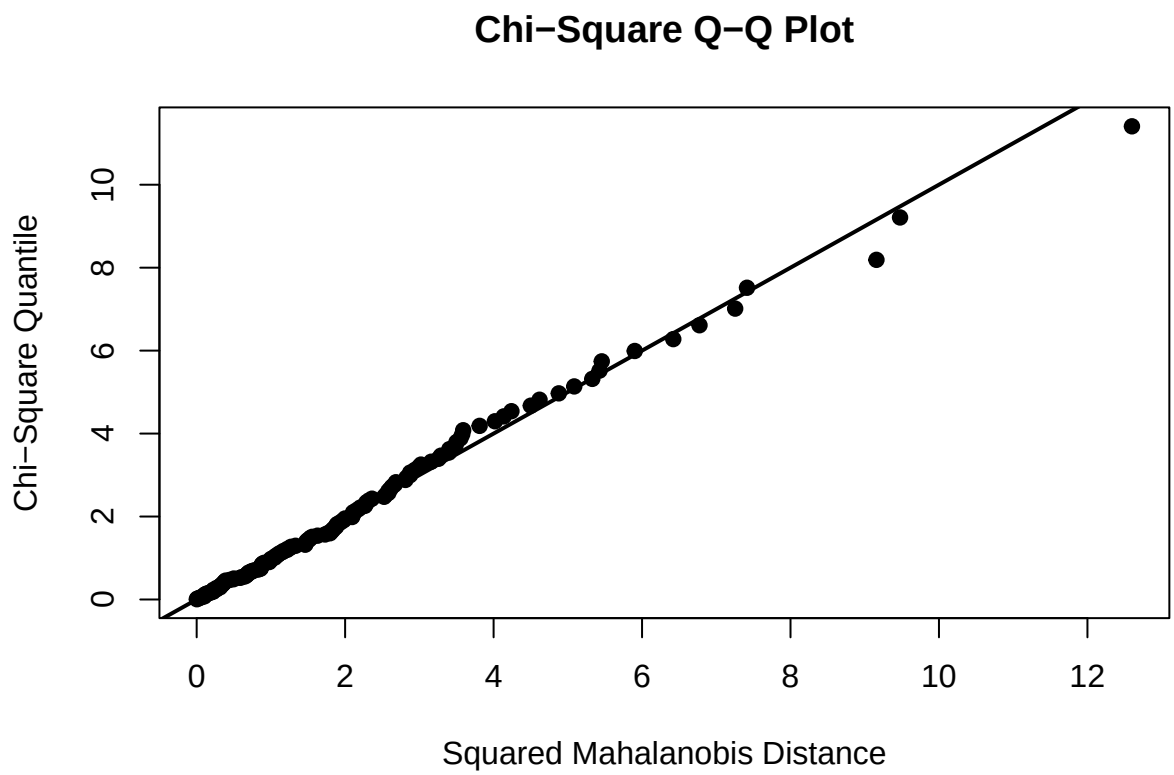

```r
### MVN package ###

# Henze-Zirkler's test
result = mvn(data = X, mvnTest = "hz") # Test performed by default
result$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.4389862 0.8033284 YES
```

```
# Royston's test (Shapiro-Wilk test under the hood)
result = mvn(data = X, mvnTest = "royston")
result$multivariateNormality
```

```
##      Test        H   p value MVN
## 1 Royston 2.505027 0.2808669 YES
```

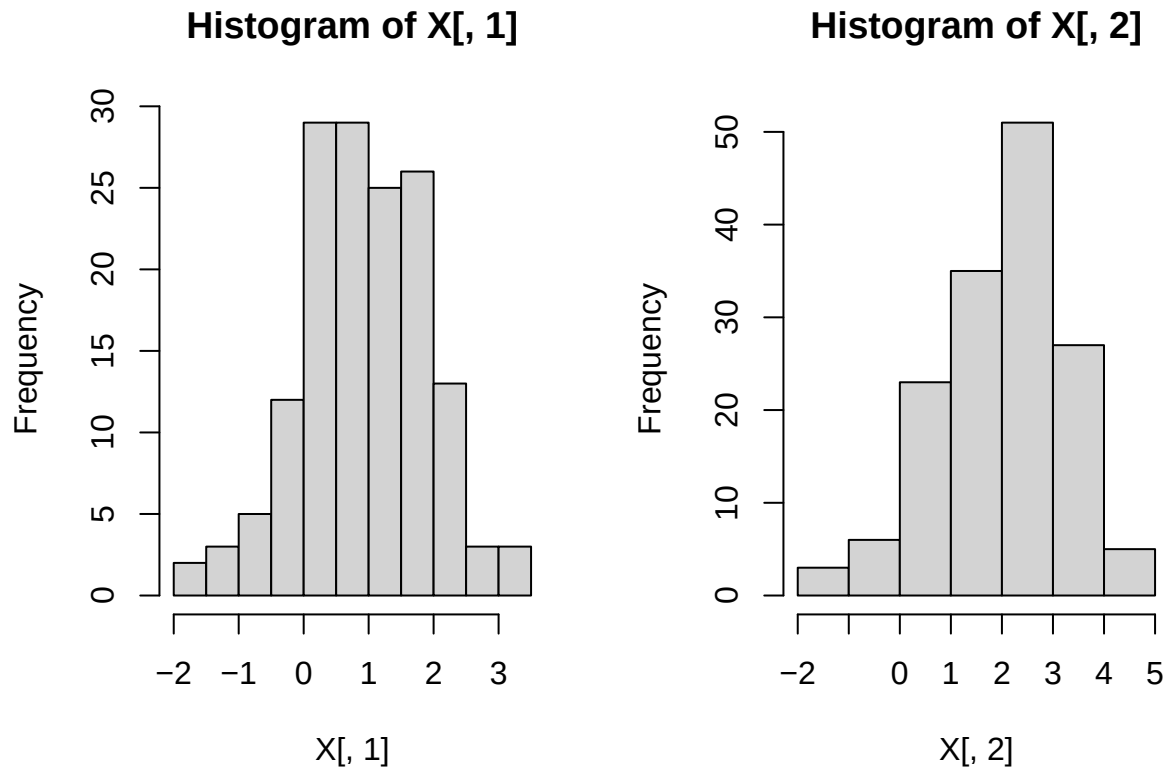
```
# with Q-Q plot of the squared mahalanobis distance over chi-square

#! the last option make us plot the qqplot
result = mvn(data = X, mvnTest = "royston", multivariatePlot = "qq")
```




```
result$multivariateNormality
```

```
##      Test        H   p value MVN
## 1 Royston 2.505027 0.2808669 YES
```

```
par(mfrow=c(1,2))
hist(X[,1])
hist(X[,2])
```

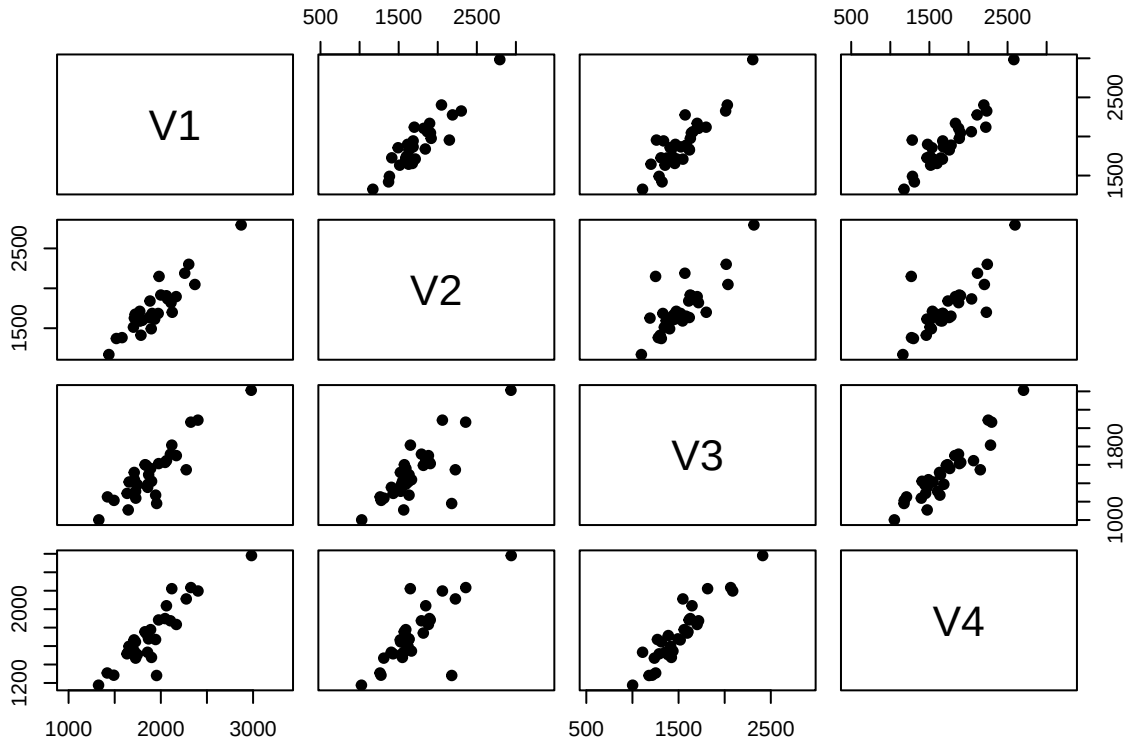


4.2 On dataset stiff

```
# Example: test of normality for the dataset stiff
stiff = read.table('Lab04/stiff.dat')
stiff
```

```
##      V1   V2   V3   V4
## 1  1889 1651 1561 1778
## 2  2403 2048 2087 2197
## 3  2119 1700 1815 2222
## 4  1645 1627 1110 1533
## 5  1976 1916 1614 1883
## 6  1712 1712 1439 1546
## 7  1943 1685 1271 1671
## 8  2104 1820 1717 1874
## 9  2983 2794 2412 2581
## 10 1745 1600 1384 1508
## 11 1710 1591 1518 1667
## 12 2046 1907 1627 1898
## 13 1840 1841 1595 1741
## 14 1867 1685 1493 1678
## 15 1859 1649 1389 1714
## 16 1954 2149 1180 1281
## 17 1325 1170 1002 1176
## 18 1419 1371 1252 1308
## 19 1828 1634 1602 1755
## 20 1725 1594 1313 1646
## 21 2276 2189 1547 2111
## 22 1899 1614 1422 1477
## 23 1633 1513 1290 1516
## 24 2061 1867 1646 2037
## 25 1856 1493 1356 1533
## 26 1727 1412 1238 1469
## 27 2168 1896 1701 1834
## 28 1655 1675 1414 1597
## 29 2326 2301 2065 2234
## 30 1490 1382 1214 1284
```

```r
plot(stiff, asp=1, pch=19)
```

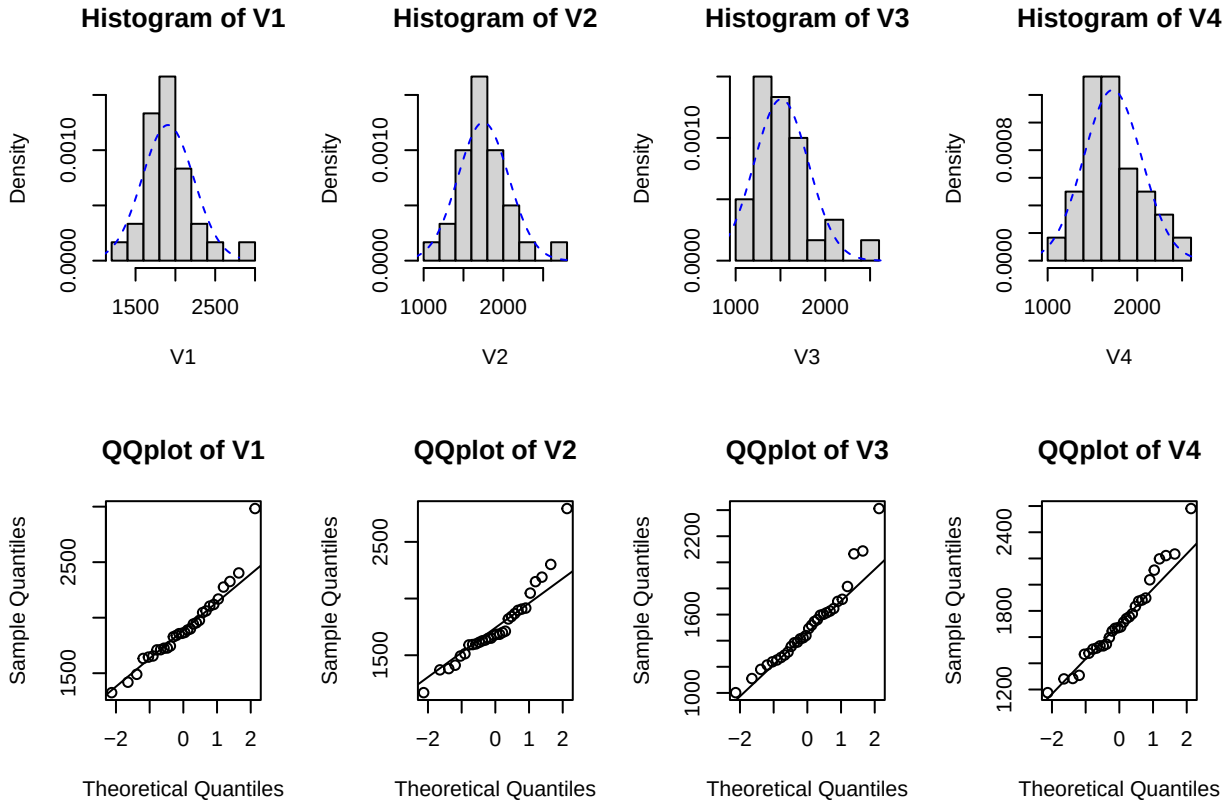



```r
#! we should get a cloud shape, as subset of components of a gaussian vector
#! should still be (and so look) like gaussians, so those "clouds" of the
#! pair looks good, some are compact clouds (good) but some no
```

4.2.1 Univariate normality

```r
# Normality of the components
par(mfcol=c(2,4))
for(i in 1:4)
{
  hist(stiff[,i], prob=T, main=paste('Histogram of V', i, sep=''), xlab=paste('V', i, sep=''))
  lines(900:2800, dnorm(900:2800,mean(stiff[,i]),sd(stiff[,i])), col='blue', lty=2)
  qqnorm(stiff[,i], main=paste('QQplot of V', i, sep=''))
  qqline(stiff[,i])
  print(shapiro.test(stiff[,i])$p)
}
```

```
## [1] 0.05118027

## [1] 0.0174619

## [1] 0.05750619
```

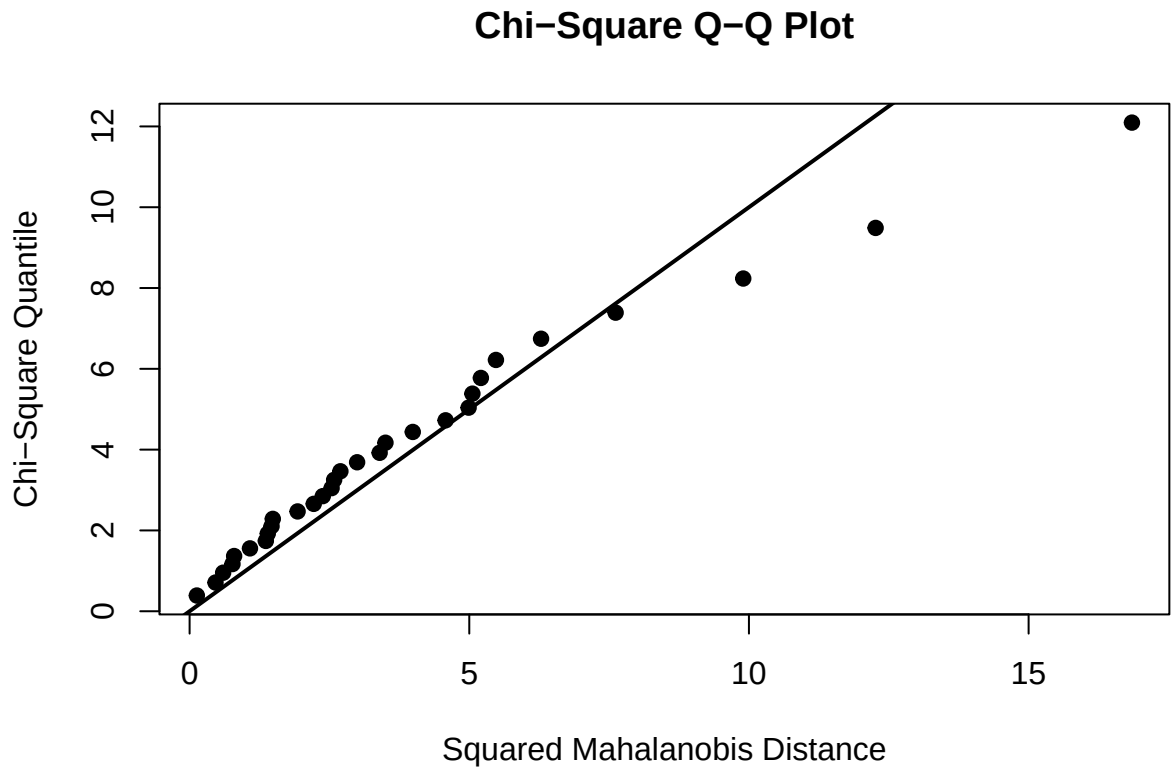



```
## [1] 0.3337193
```

```
#! just V4 looks healthy normal, the others have high right tails, and are skewed
```

4.2.2 Multivariate normality

```
result = mvn(data = stiff, multivariatePlot = "qq", covariance = FALSE)
```




```
result$multivariateNormality
```

```
##            Test      HZ     p value MVN
## 1 Henze-Zirkler 1.06231 0.003790558  NO
```

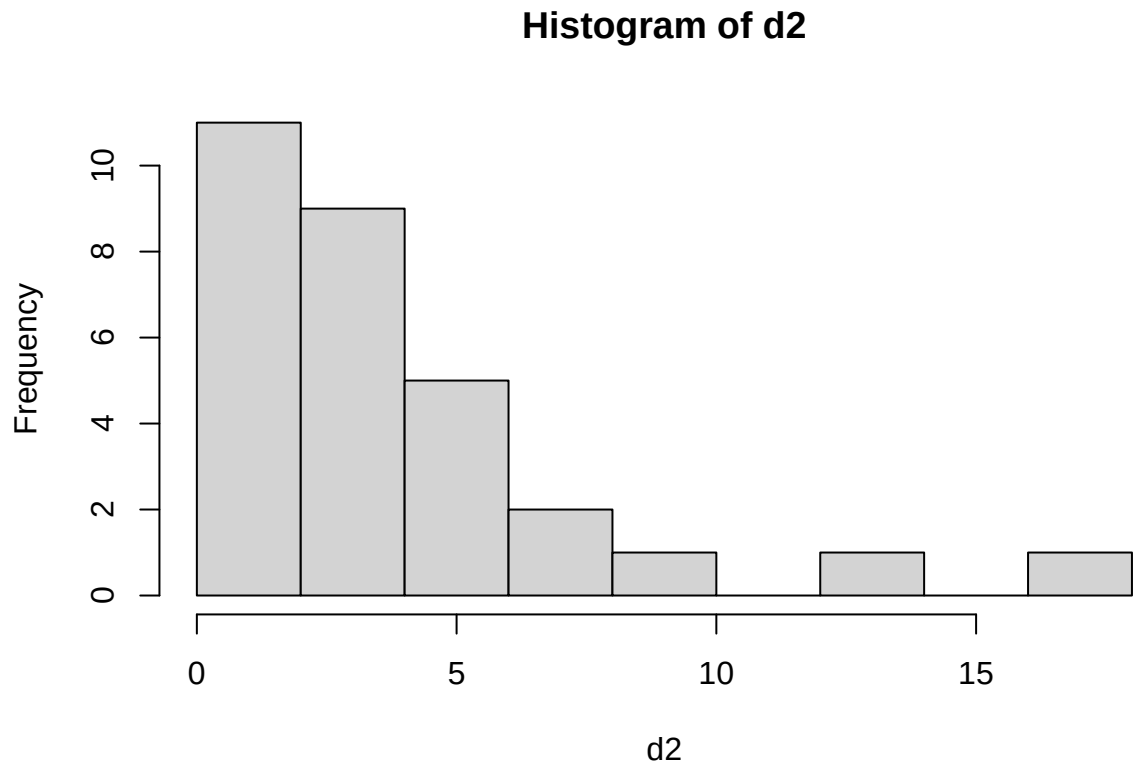
```
#! low pvalue => we reject H0, our sample is not gaussian
```

4.2.3 Recovering gaussianity

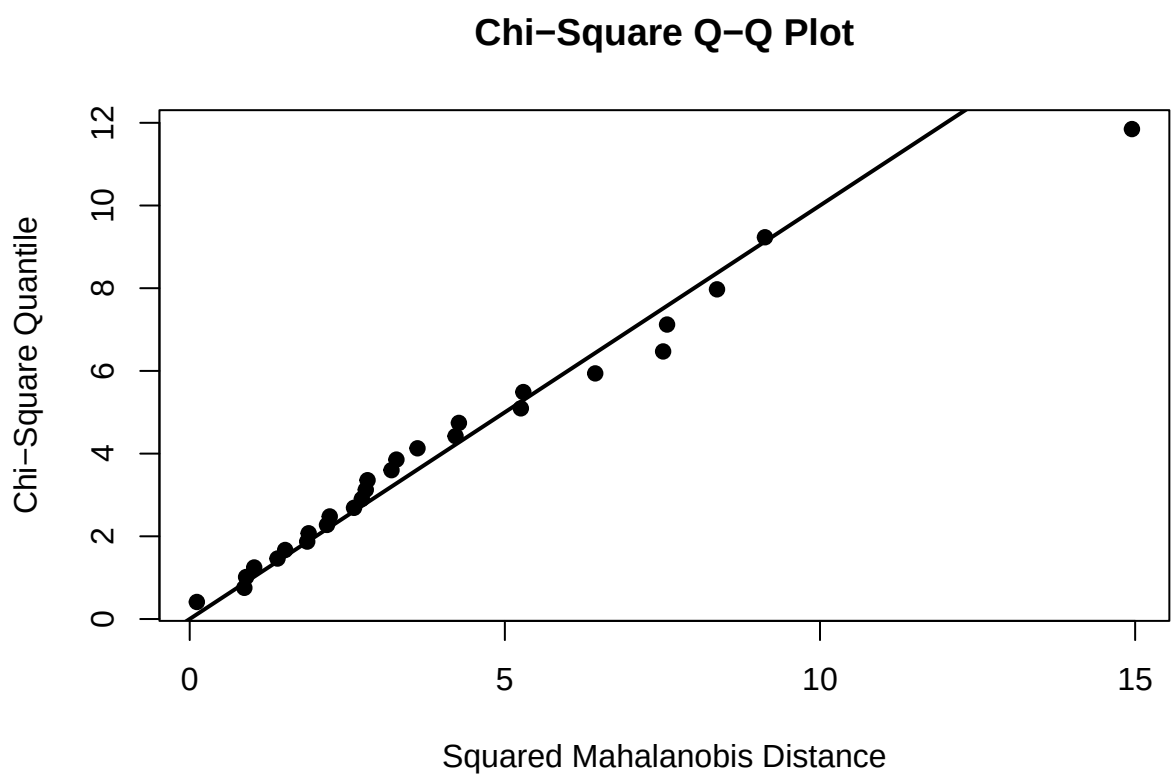
```
### The data don't seem Gaussian. What can we do?
### 1. Identify clusters
### 2. Identify (and possibly remove) outliers
### 3. Transform the data (e.g., Box-Cox transformations)
### 4. Work without the Gaussian assumption (e.g., permutation tests)

#! Transforming is perfectly ok, we dont lose the information that our data carry
#! We alter their form, but we are not changing the dependence between those data
#! But just transform each column, dont mix them (like dont do a*col1+b*col2^2 ecc)

#! Transformations must not be linear (otherwise we would have already had gaussianity)
#! and especially for Box-Cox before doing the transformation remove the outliers
```

```
### Let's try to identify and remove outliers:
M = colMeans(stiff)
S = cov(stiff)

d2 = matrix(mahalanobis(stiff, M, S))
hist(d2)
```

4.2.3.1 Remove outliers



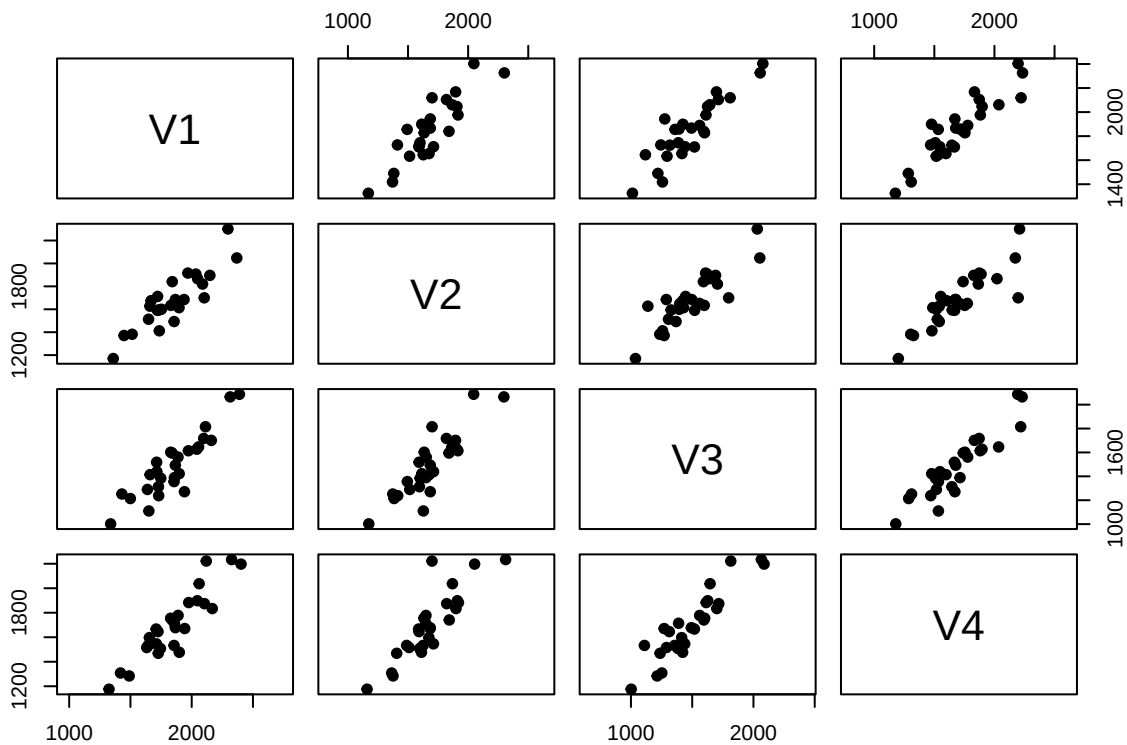

```
#! we can see there are some points too far, like over 8, so:
stiff_wo_outliers = stiff[which(d2 <= 8), ]
#! actually we removed 3 points, from 30 to 27 size

result = mvn(data = stiff_wo_outliers, multivariatePlot = "qq")
```



```r
result$multivariateNormality
```

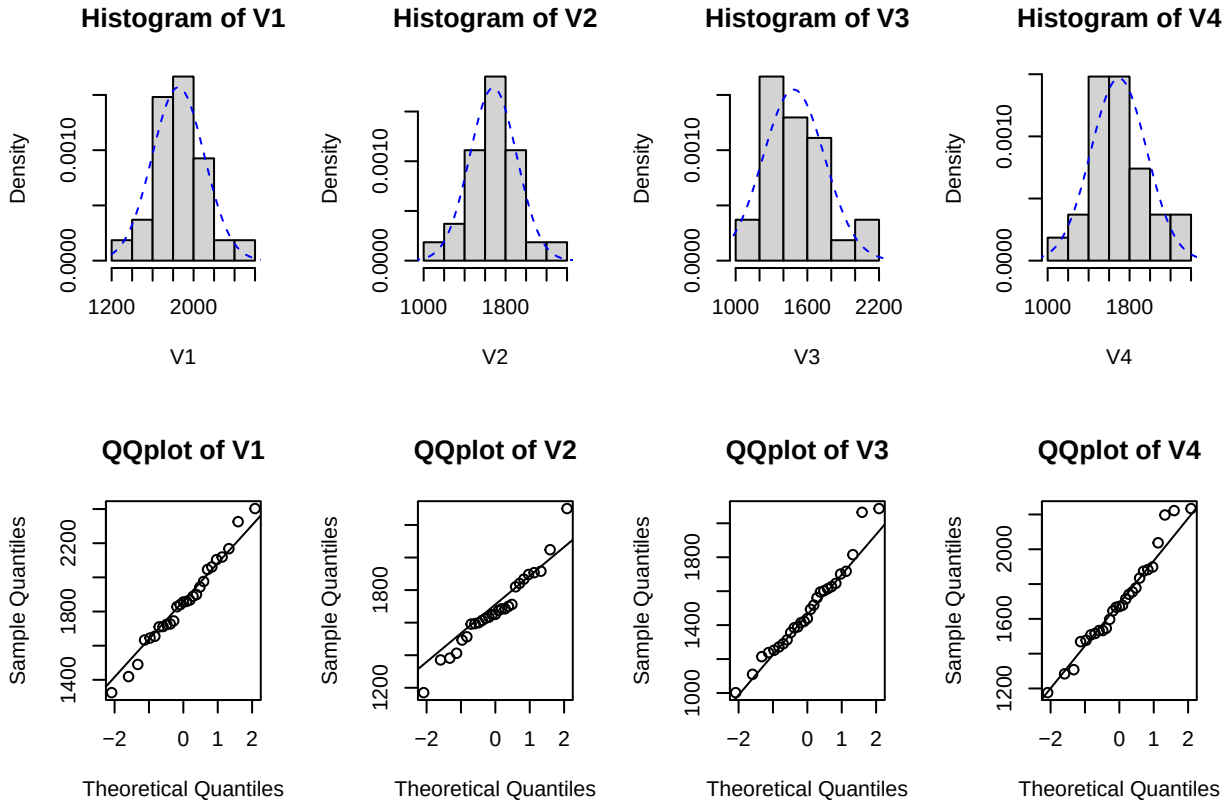
```
##             Test        HZ   p value MVN
## 1 Henze-Zirkler 0.6220241 0.8032532 YES
```

```r
plot(stiff_wo_outliers, asp=1, pch=19)
```




```r
# Normality of the components
par(mfcol=c(2,4))
for(i in 1:4)
{
  hist(stiff_wo_outliers[,i], prob=T, main=paste('Histogram of V', i, sep=''),
     xlab=paste('V', i, sep=''))
  lines(900:2800, dnorm(900:2800,mean(stiff_wo_outliers[,i]),
                        sd(stiff_wo_outliers[,i])), col='blue', lty=2)
  qqnorm(stiff_wo_outliers[,i], main=paste('QQplot of V', i, sep=''))
  qqline(stiff_wo_outliers[,i])
  print(shapiro.test(stiff_wo_outliers[,i])$p)
}
```

```
## [1] 0.9542918

## [1] 0.5009671

## [1] 0.4958939
```

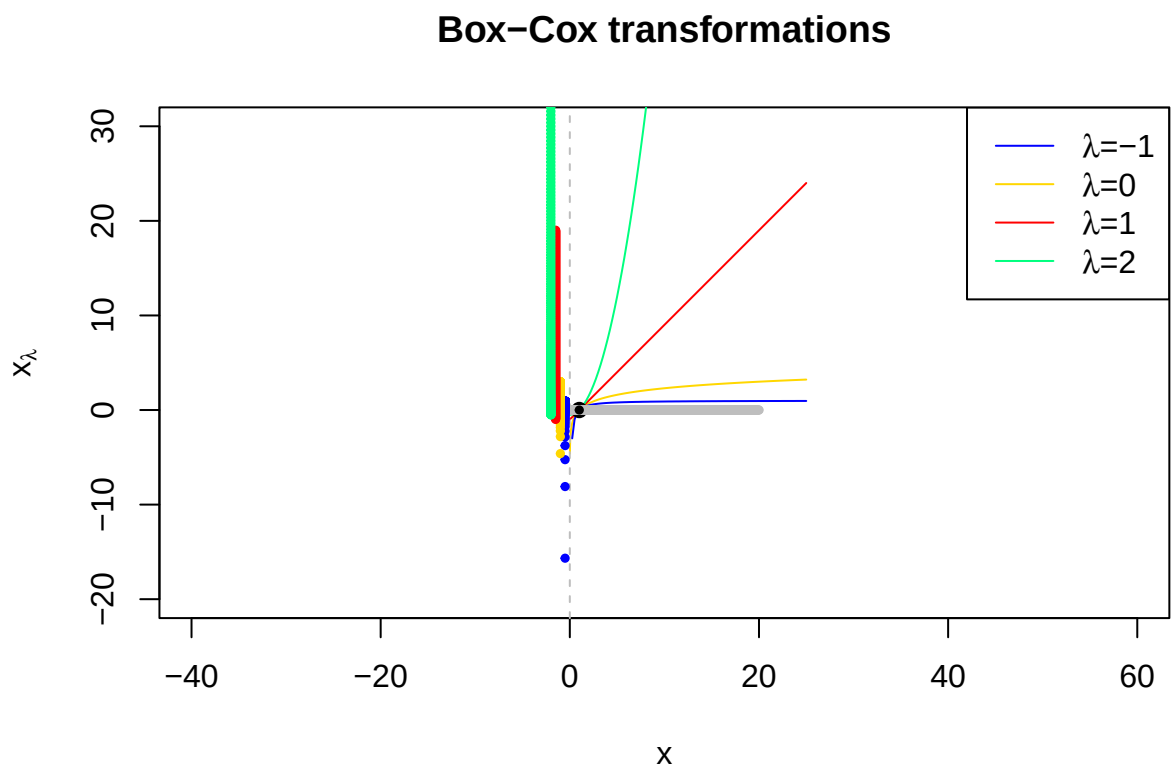



```
## [1] 0.4443887
```

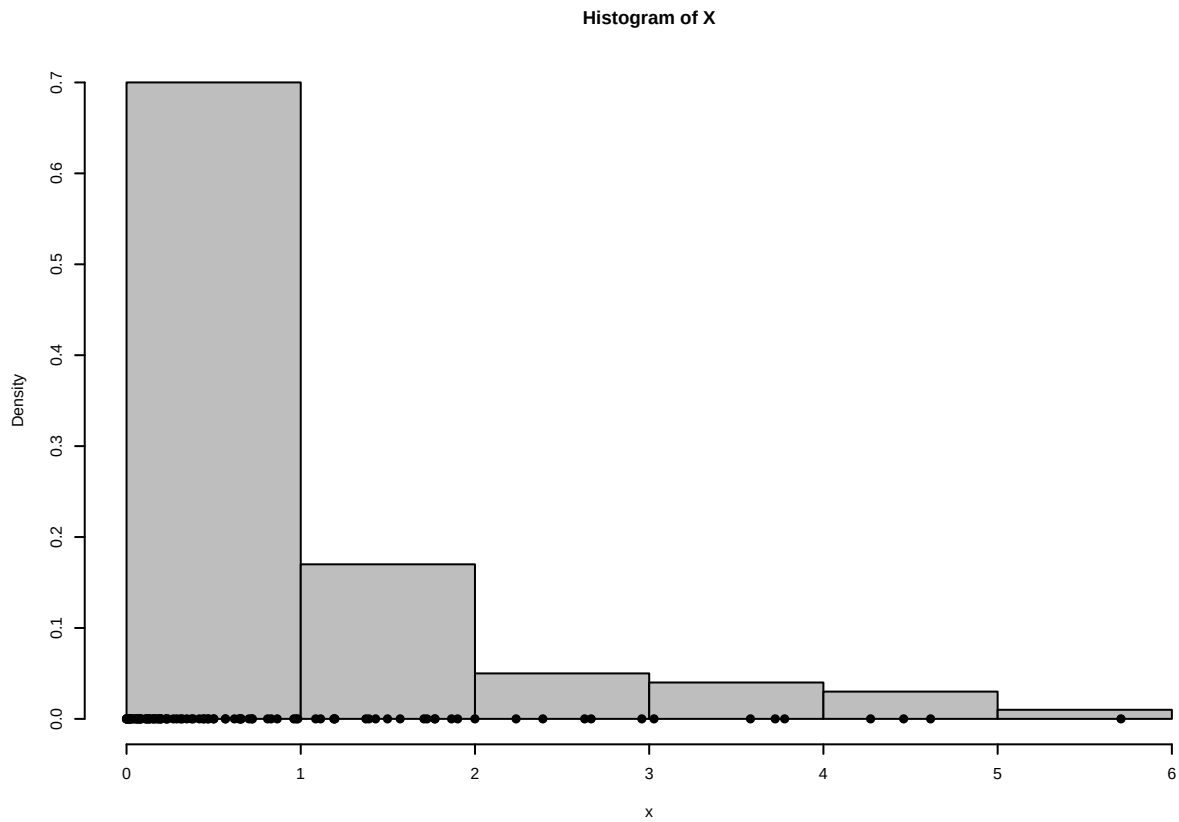
```
### Box-Cox transformations

##############################################################################
### Univariate Box-Cox transformation (Johnson-Wichern Chap.4.8)
# The Box-Cox transformations are based on the family of power transformation
# (for x>0)
# x_lambda = / (x^lambda-1)/lambda if lambda!=0
#            \ ln(x)               if lambda==0
# that is continuous in lambda for x>0.
# The parameter lambda is determined as the maximum likelihood solution
# under the Gaussian assumption (see J-W)

# Let's plot the transformations for some values of lambda

box_cox = function(x,lambda) {
  if(lambda!=0)
    return((x^lambda-1)/lambda)
  return(log(x))
}

x=seq(0, 25, by=0.01)

plot(x, box_cox(x,0), col='gold', type='l', xlab='x', ylab=expression(x[lambda]),
     ylim=c(-20,30), xlim=c(-5,25), asp=1)
title(main='Box-Cox transformations')
curve(box_cox(x,-1),from=0,to=25,col='blue',add=TRUE)
curve(box_cox(x,1),from=0,to=25,col='red',add=TRUE)
curve(box_cox(x,2),from=0,to=25,col='springgreen',add=TRUE)
points(1,0,pch=19,col='black')
abline(v=0,lty=2,col='grey')
legend('topright',c(expression(paste(lambda,'=-1')),expression(paste(lambda,'=0')),
                    expression(paste(lambda,'=1')),expression(paste(lambda,'=2'))),
       col=c('blue','gold','red','springgreen'),lty=c(1,1,1,1,1))

xx=seq(0.01, 20.01, by=.05)
```

```
par(cex=.5)
points(xx, rep(0,length(xx)), col='grey', pch=19)
points(-.5+rep(0,length(xx)), box_cox(xx, -1), col='blue', pch=19)
points(-.5+rep(-.5,length(xx)), log(xx), col='gold', pch=19)
points(-.5+rep(-1,length(xx)), box_cox(xx,1), col='red', pch=19)
points(-.5+rep(-1.5,length(xx)), box_cox(xx,2), col='springgreen', pch=19)
points(1, 0, pch=19, col='black')
```

4.2.3.2 Box-Cox transformations




```
# For lambda<1: observations <1 are "spread", observations >1 are "shrinked"
# For lambda>1: observations <1 are "shrinked", observations >1 are "spread"


##### A trivial example
n = 100
set.seed(20032023)
x = rnorm(n)^2
hist(x, col='grey', prob=T, xlab='x', main ='Histogram of X')
points(x, rep(0,n), pch=19)
```



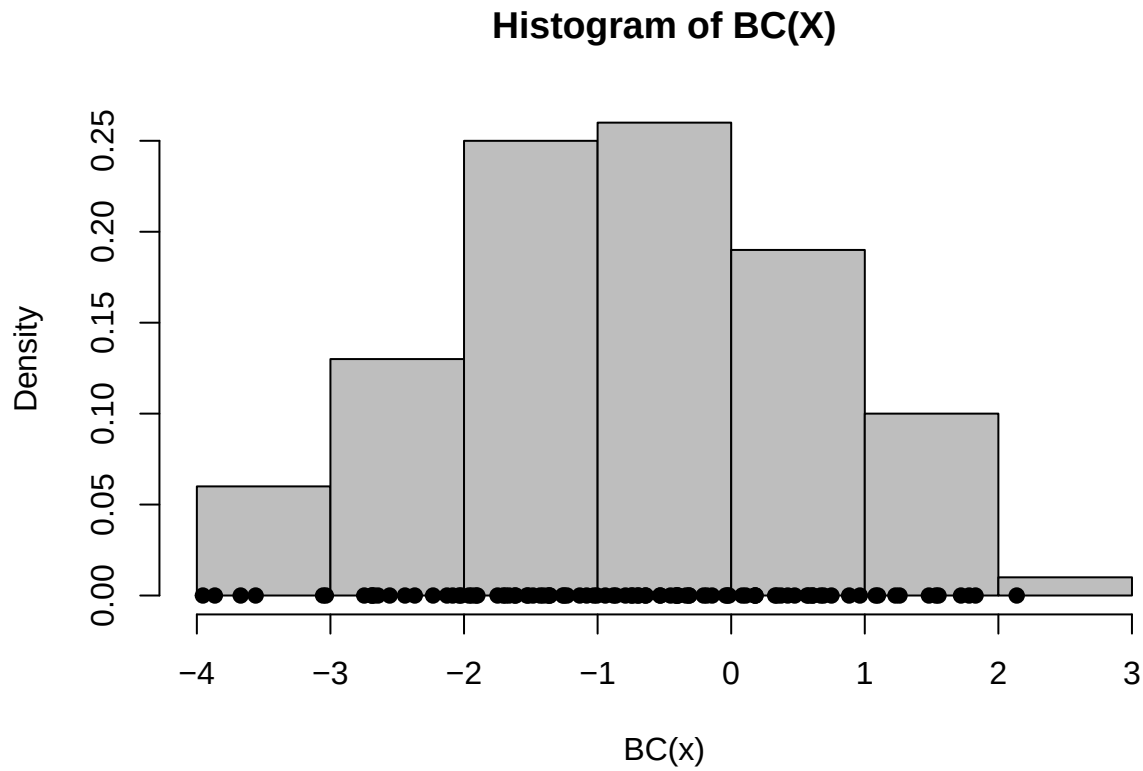

```
# we would need to spread the obs with small values and shrink the obs with
# large values => we expect a small lambda (lambda<1)
```

```
# Univariate Box-Cox transformation
# We compute the optimal lambda of the univariate Box-Cox transformation
# (command powerTransform of library car)
#! it finds the optimal lambda to make our data gaussian
lambda.x = powerTransform(x)
lambda.x
```

4.2.3.3 Univariate transformation

```
## Estimated transformation parameter
##         x
## 0.2274822
```

```
# lambda<1: observations <1 are "spread", observations >1 are "shrinked"

# Transformed sample with the optimal lambda (command bcPower of library car)
bc.x = bcPower(x, lambda.x$lambda)
# it transforms the data of the first argument through the Box-Cox
# transformation with lambda given as second argument

hist(bc.x, col='grey', prob=T, main='Histogram of BC(X)', xlab='BC(x)')
points(bc.x, rep(0,n), pch=19)
```




```
shapiro.test(x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.75316, p-value = 1.136e-11
```

```
shapiro.test(bc.x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bc.x
## W = 0.99021, p-value = 0.6814
```

```
rm(x)
```

```
### Multivariate Box-Cox transformation (Johnson-Wichern Cap.4.8)
# Similar to univariate transformation, but jointly on all the variables

##### Simulated Example
b = read.table('Lab04/data_sim.txt')
head(b)
```

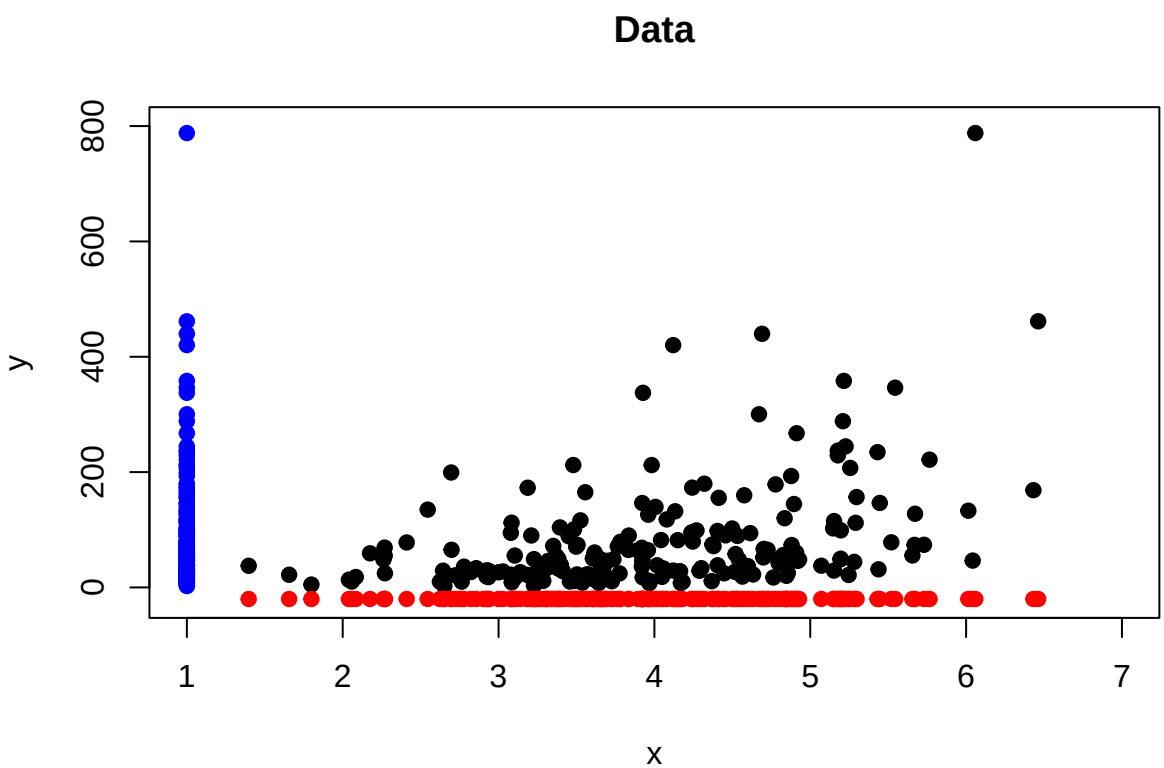
4.2.3.4 Multivariate transformation

```
##          x         y
## 1 5.446189 146.53762
## 2 2.084034  18.09554
## 3 4.238034  95.42054
## 4 4.690763 439.80116
## 5 4.042480  82.17827
## 6 3.573651  22.85204
```

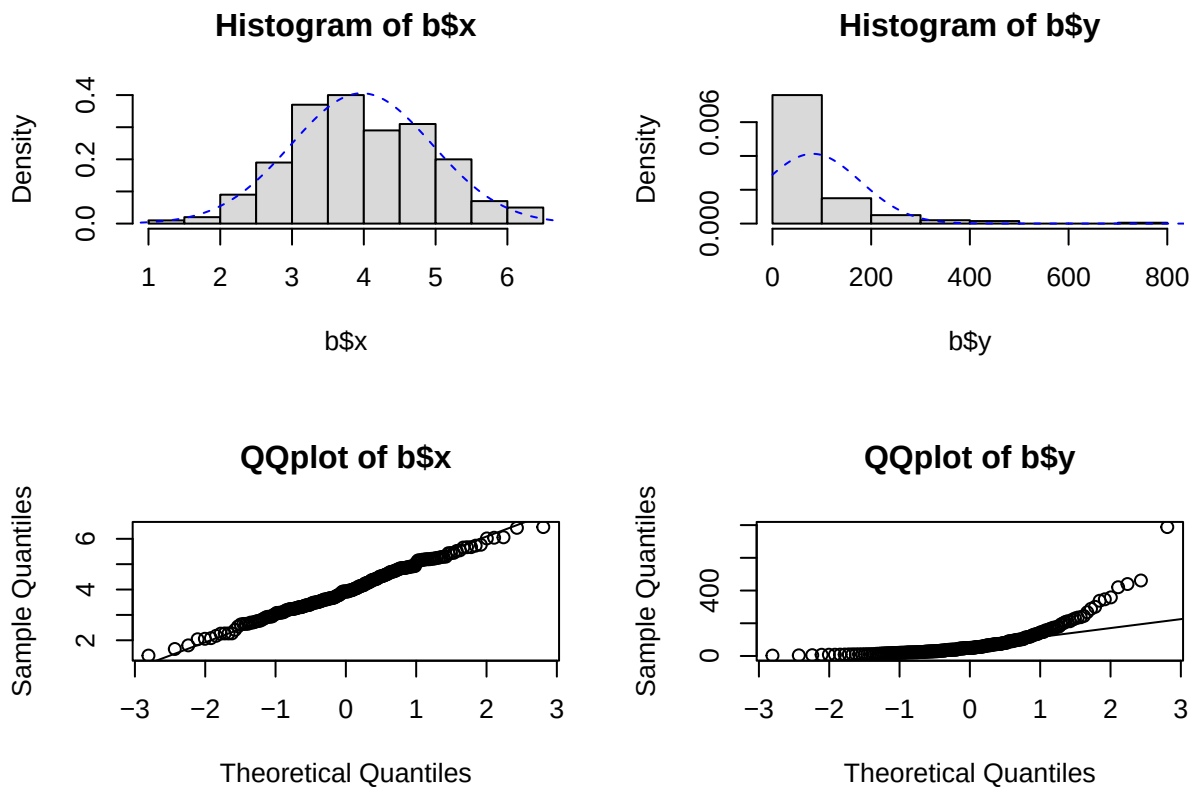
```r
dim(b)
```

```
## [1] 200   2
```

```r
plot(b, pch=19,main='Data', xlim=c(1,7), ylim=c(-20,800))
points(b$x, rep(-20,dim(b)[1]), col='red', pch=19)
#! on the x axis it looks normal, on the y not really
points(rep(1,dim(b)[1]), b$y, col='blue', pch=19)
```




```r
par(mfrow=c(2,2))
hist(b$x, prob=T,col='grey85')
lines(0:1000/100, dnorm(0:1000/100, mean(b$x), sd(b$x)), col='blue', lty=2)
hist(b$y, prob=T,col='grey85')
lines(0:1000, dnorm(0:1000, mean(b$y), sd(b$y)), col='blue', lty=2)

qqnorm(b$x, main='QQplot of b$x')
qqline(b$x)
qqnorm(b$y, main='QQplot of b$y')
qqline(b$y)
```




```
# For y: left tail too light, right tail too heavy

shapiro.test(b$x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  b$x
## W = 0.99443, p-value = 0.6653
```

```
shapiro.test(b$y)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  b$y
## W = 0.67337, p-value < 2.2e-16
```

```
mshapiro.test(t(b))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.67479, p-value < 2.2e-16
```

```
#! non-normality of one component => whole vector not normal
#! while the opposite is not true, we can have a vector jointly gaussian
#! but with components that are not (really?)
```

```
### Bivariate Box-Cox transformation
# Compute the optimal lambda of a bivariate Box-Cox transformation
# (command powerTransform, with a multivariate input)
#! we cant use two times the univariate Box-Cox, that doesnt work

powerTransform(b$x)
```

```
## Estimated transformation parameter
##       b$x
## 0.8749098
```

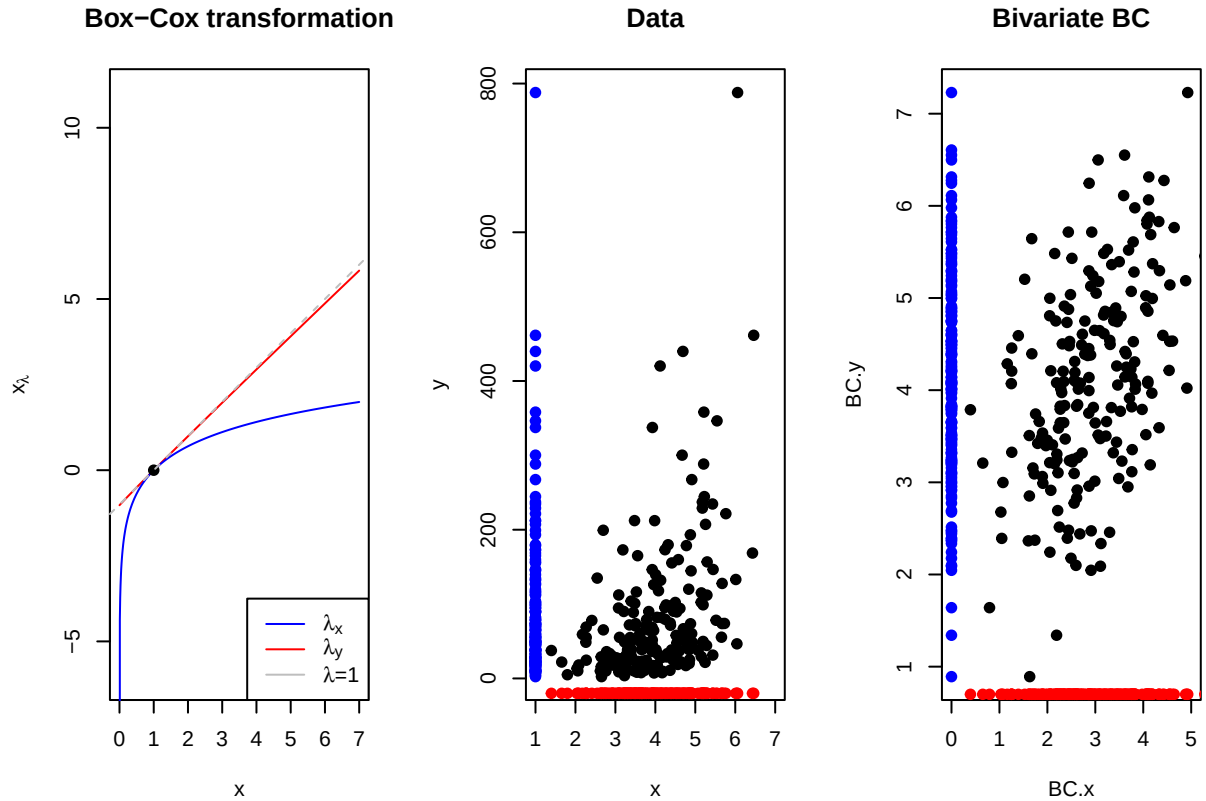
```
powerTransform(b$y)
```

```
## Estimated transformation parameter
##         b$y
## 0.005747775
```

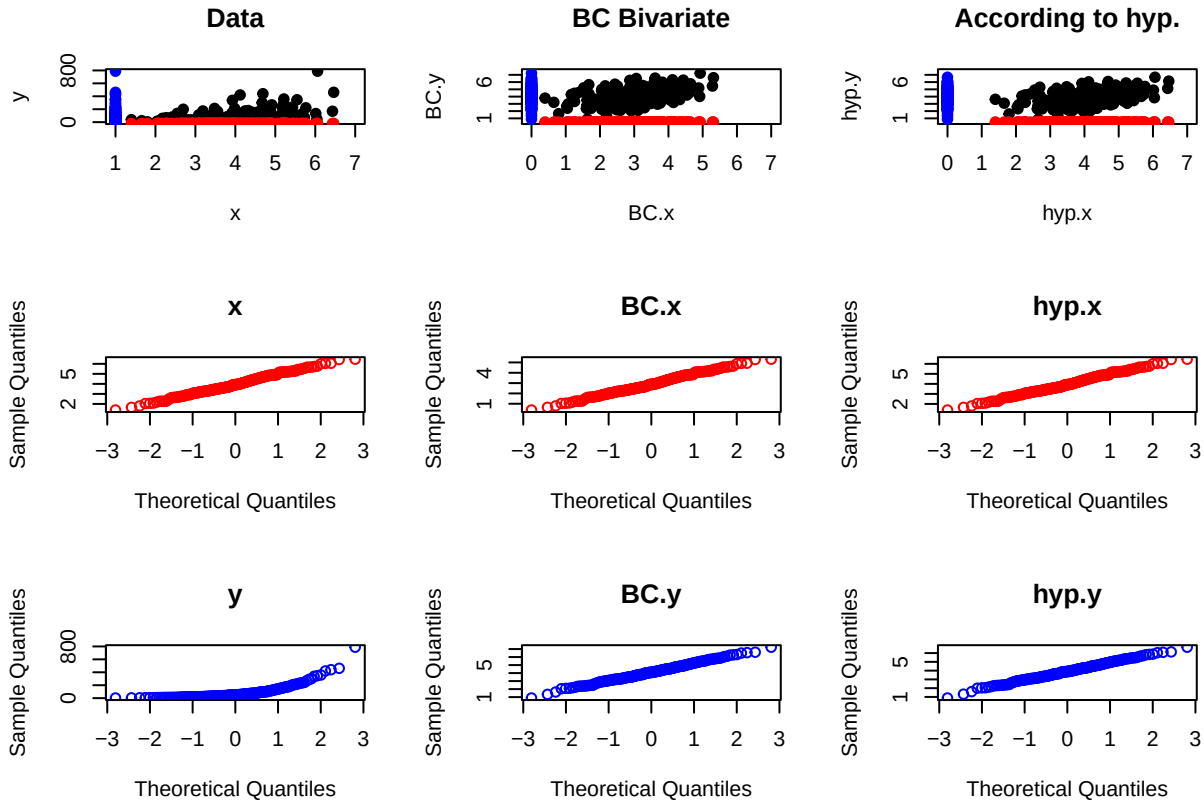
```
#! vs this
lambda = powerTransform(cbind(b$x,b$y))
lambda
```

```
## Estimated transformation parameters
##         Y1         Y2
## 0.97717477 0.02386159
```

```
# lambda[1] close to 1

# lambda[2] close to 0, it is almost a log transform;
#! we can approx lambda=0, maybe with a test to check if it is actually change something
# lambda[2]<1:  obs <1 "spread", obs >1 "shrinked"
# => it adds weight to the left tail, lightens the right tail

# Compute the transformed data with optimal lambda (of the bivariate transf.)
# (command bcPower)
#! We find the lambda with a joint command, but then the trasnformation is done
#! singularly, like there is no bcPower-joint-version command
BC.x = bcPower(b$x, lambda$lambda[1])
BC.y = bcPower(b$y, lambda$lambda[2])

xx = seq(0, 7, by=0.01)

par(mfrow=c(1,3))
plot(xx, box_cox(x=xx, lambda=lambda$lambda[1]), col='red', lty=1, type='l',
     xlab=expression(x), ylab=expression(x[lambda]), ylim=c(-5,10), asp=1)
title(main='Box-Cox transformation')
lines(xx, box_cox(x=xx, lambda=lambda$lambda[2]), col='blue')
points(1, 0, pch=19, col='black')
abline(a=-1, b=1, lty=2, col='grey')
legend('bottomright', c(expression(lambda[x]),expression(lambda[y]),
                        expression(paste(lambda,'=1'))),
       col=c('blue','red','grey'),lty=c(1,1,1))

plot(b, pch=19, main='Data', xlim=c(1,7))
points(rep(1,200), b$y, pch=19, col='blue') # projection on y
points(b$x, rep(-20,200), pch=19, col='red')  # projection on x

plot(BC.x, BC.y, pch=19, main='Bivariate BC', xlim=c(0,5))
points(rep(0,200), BC.y, pch=19, col='blue') # projection on y
points(BC.x, rep(0.7,200), pch=19, col='red')  # projection on x
```




```
# Let's formulate an hypothesis of transformation:
# since we get lambda[1]~1 and lambda[2]~0, we could reasonably consider:
hyp.x = b$x
hyp.y = log(b$y)

# Remark: from the application-oriented viewpoint, explaining a log
# transform could be simpler than introducing Box-Cox transformations

par(mfrow=c(3,3))
plot(b, pch=19, main='Data', xlim=c(1,7))
points(rep(1,200), b$y, pch=19, col='blue')   # projection on y
points(b$x, rep(-20,200), pch=19, col='red')  # projection on x

plot(BC.x, BC.y, pch=19, main='BC Bivariate', xlim=c(0,7), ylim=c(0.5,7.5))
points(rep(0,200), BC.y, pch=19, col='blue')   # projection on y
points(BC.x, rep(0.5,200), pch=19, col='red')  # projection on x

plot(hyp.x, hyp.y, pch=19, main='According to hyp.', xlim=c(0,7), ylim=c(0.5,7.5))
points(rep(0,200), hyp.y, pch=19, col='blue')  # projection on y
points(hyp.x, rep(0.5,200), pch=19, col='red') # projection on x

qqnorm(b$x, main="x", col='red')
qqnorm(BC.x, main="BC.x", col='red')
qqnorm(hyp.x, main="hyp.x", col='red')

qqnorm(b$y, main="y", col='blue')
qqnorm(BC.y, main="BC.y", col='blue')
qqnorm(hyp.y, main="hyp.y", col='blue')
```



4.2.4 Pvalue explanation

```
#! Il pvalue dà la prob di osservare un valore pari a (o più estremo di) quello
#! osservato con la statistica test. Quindi più è alto più vuol dire che quel
#! valore era coerente coi dati, cioè era un valore molto tranquillo, si può fare
#! "molto peggio di lui".
#! Mentre se è alto vuol dire che ci insospettiamo, abbiamo osservato un valore che
#! sembra difficile, improbabile da ottenere secondo quella distribuzione

# Univariate Shapiro-Wilk (H0: univariate gaussianity along a given direction)
shapiro.test(b$x)$p
```

```
## [1] 0.6652598
```

```
shapiro.test(BC.x)$p
```

```
## [1] 0.6871995
```

```
shapiro.test(hyp.x)$p
```

```
## [1] 0.6652598
```

```
shapiro.test(b$y)$p
```

```
## [1] 1.916013e-19
```

```
shapiro.test(BC.y)$p
```

```
## [1] 0.9912947
```

```
shapiro.test(hyp.y)$p
```

```
## [1] 0.9939556
```

```r
# MC-Shapiro test (H0: multivariate gaussianity)
mshapiro.test(rbind(b$x,b$y))$p
```

```
## [1] 2.097264e-19
```

```r
mshapiro.test(rbind(BC.x,BC.y))$p
```

```
## [1] 0.8551287
```

```r
mshapiro.test(rbind(hyp.x,hyp.y))$p
```

```
## [1] 0.9932309
```

```r
# High p-value for the transformed data: with the obtained transformation we
# have no evidence of non-normality, ie, we can assume that the transformed
# data are Gaussian
```

4.2.5 Tranforming the data

```r
### Box-Cox transformation on the dataset stiff

stiff = read.table('Lab04/stiff.dat')
head(stiff)
```

```
##     V1   V2   V3   V4
## 1 1889 1651 1561 1778
## 2 2403 2048 2087 2197
## 3 2119 1700 1815 2222
## 4 1645 1627 1110 1533
## 5 1976 1916 1614 1883
## 6 1712 1712 1439 1546
```

```r
dim(stiff)
```

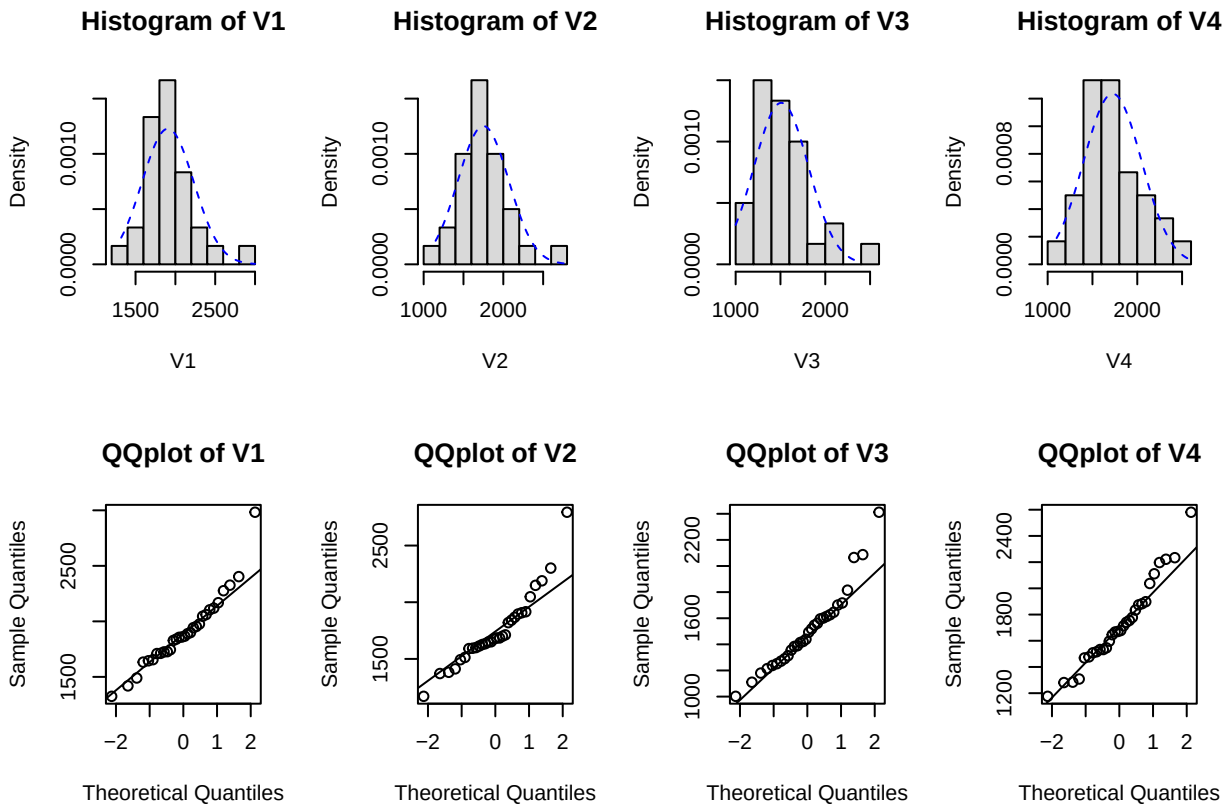
```
## [1] 30  4
```

```r
lambda.mult = powerTransform(stiff)
lambda.mult
```

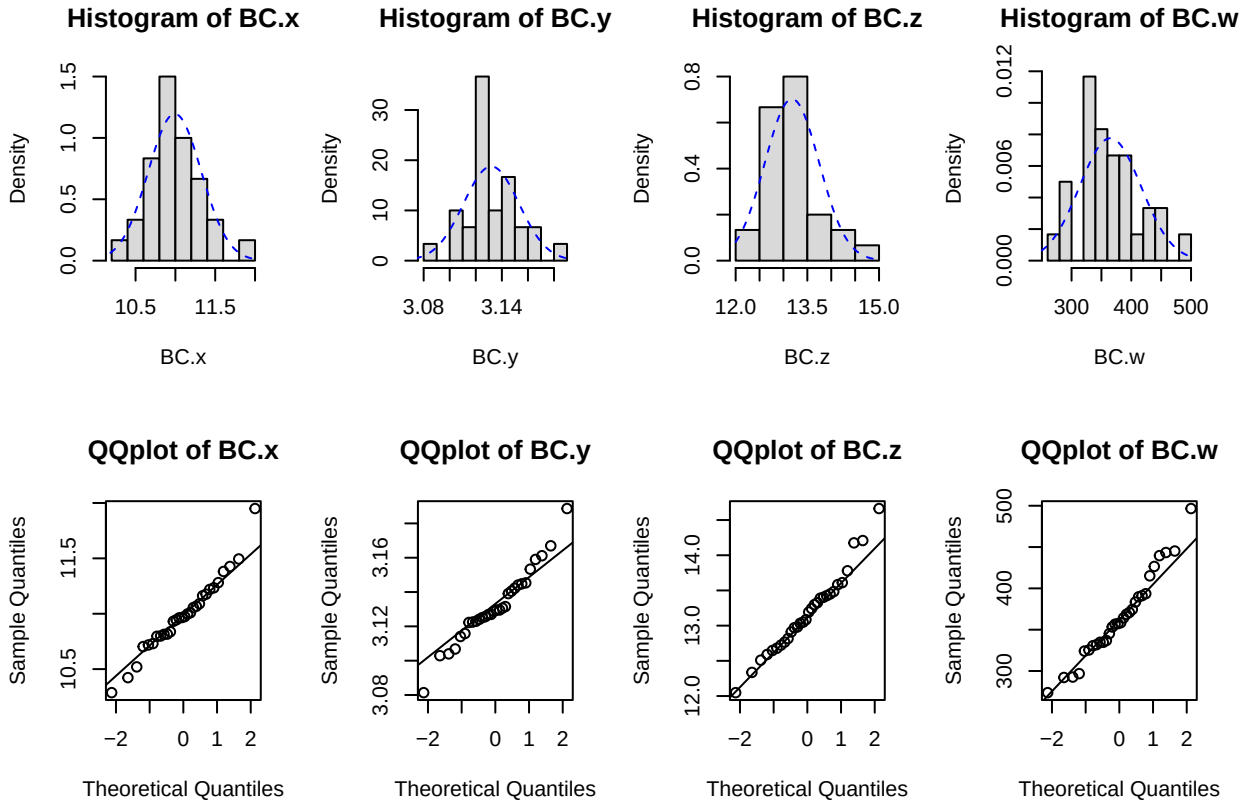
```
## Estimated transformation parameters
##          V1          V2          V3          V4
##  0.09437843 -0.27947125  0.14835794  0.75472854
```

```r
BC.x = bcPower(stiff[,1], lambda.mult$lambda[1])
BC.y = bcPower(stiff[,2], lambda.mult$lambda[2])
BC.z = bcPower(stiff[,3], lambda.mult$lambda[3])
BC.w = bcPower(stiff[,4], lambda.mult$lambda[4])

# Plot of the original variables
attach(stiff)

par(mfrow=c(2,4))
xx=seq(1320, 3000, length=100)
hist(V1, prob=T, breaks=8, col='grey85')
lines(xx, dnorm(xx,mean(V1), sd(V1)), col='blue', lty=2)
yy=seq(1150, 2800, length=100)
hist(V2, prob=T,breaks=8,col='grey85')
lines(yy, dnorm(yy,mean(V2), sd(V2)), col='blue', lty=2)
zz=seq(1000, 2420, length=100)
hist(V3, prob=T, breaks=8, col='grey85')
lines(zz, dnorm(zz,mean(V3), sd(V3)), col='blue', lty=2)
ww=seq(1100, 2600, length=100)
hist(V4, prob=T, breaks=8, col='grey85')
lines(ww, dnorm(ww, mean(V4), sd(V4)), col='blue', lty=2)

qqnorm(V1, main='QQplot of V1')
qqline(V1)
```

```
qqnorm(V2, main='QQplot of V2')
qqline(V2)

qqnorm(V3, main='QQplot of V3')
qqline(V3)

qqnorm(V4, main='QQplot of V4')
qqline(V4)
```




```
detach(stiff)

# Plot of the transformed variables
par(mfrow=c(2,4))
xx=seq(10, 12, length=100)
hist(BC.x, prob=T, breaks=8, col='grey85')
lines(xx, dnorm(xx, mean(BC.x), sd(BC.x)), col='blue', lty=2)
yy=seq(3, 3.2, length=100)
hist(BC.y, prob=T, breaks=8, col='grey85')
lines(yy, dnorm(yy, mean(BC.y), sd(BC.y)), col='blue', lty=2)
zz=seq(12, 15, length=100)
hist(BC.z, prob=T, breaks=8, col='grey85')
lines(zz, dnorm(zz, mean(BC.z), sd(BC.z)), col='blue', lty=2)
ww=seq(250, 500, length=100)
hist(BC.w, prob=T, breaks=8, col='grey85')
lines(ww, dnorm(ww, mean(BC.w), sd(BC.w)), col='blue', lty=2)

qqnorm(BC.x, main='QQplot of BC.x')
qqline(BC.x)

qqnorm(BC.y, main='QQplot of BC.y')
qqline(BC.y)

qqnorm(BC.z, main='QQplot of BC.z')
qqline(BC.z)

qqnorm(BC.w, main='QQplot of BC.w')
qqline(BC.w)
```




```r
# One transformed variable at a time
shapiro.test(BC.x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  BC.x
## W = 0.97172, p-value = 0.5874
```

```r
shapiro.test(BC.y)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  BC.y
## W = 0.96952, p-value = 0.5263
```

```r
shapiro.test(BC.z)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  BC.z
## W = 0.97475, p-value = 0.6753
```

```r
shapiro.test(BC.w)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  BC.w
## W = 0.96898, p-value = 0.5117
```

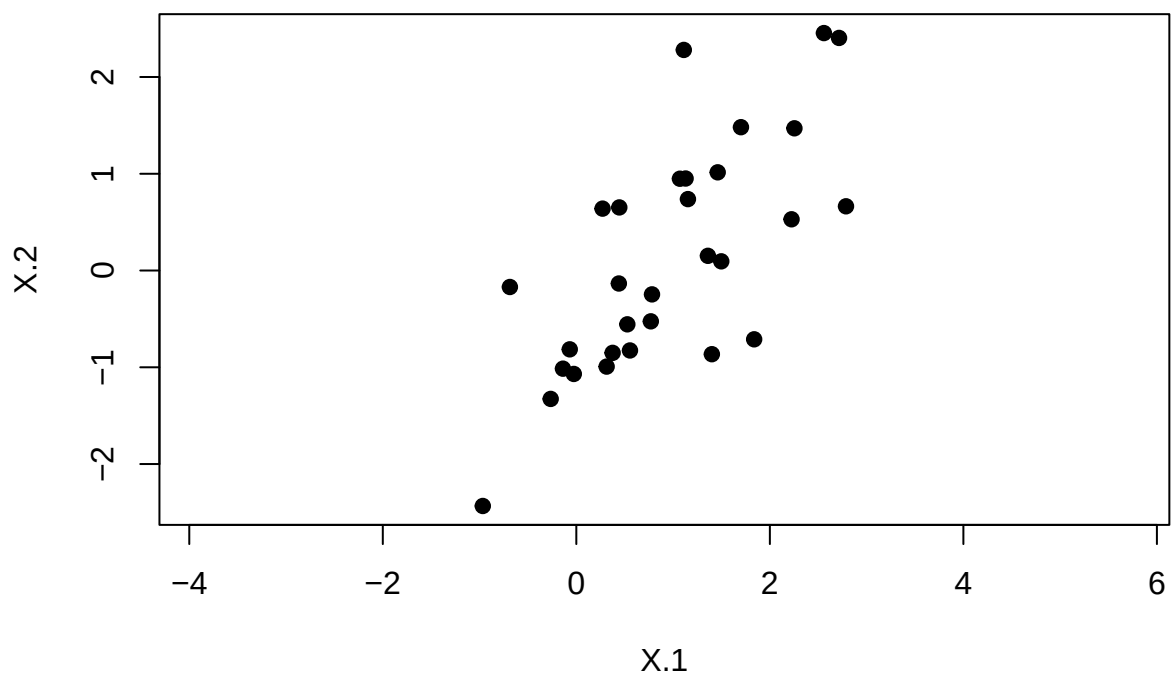
```r
# All together
mshapiro.test(rbind(BC.x, BC.y, BC.z, BC.w))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.73061, p-value = 4.493e-06
```

```
#! we recover gaussianity for components, but not jointly
#! Thats because the Box-Cox wors on components, not on their joint behaviour

# Note: the higher the dimensionality, the more difficult to
# recover the gaussianity

# In this case, Box-Cox transformations are not enough. We saw that removing
# outliers indeed solve the problems of non-Gaussianity.
# Alternative approaches:
# - Asymptotics
# - Non-parametrics (e.g., permutation tests)
```

4.3 On a multivariate Gaussian

```
### Tests and confidence regions for the mean of a multivariate Gaussian
### Example with simulated data from a bivariate Gaussian

mu = c(1, 0)
sig = matrix(c(1, 1, 1, 2), nrow=2) # sigma matrix

set.seed(123)
x = rmvnorm(n=30, mean=mu, sigma=sig)
x = data.frame(X.1=x[,1], X.2=x[,2])

plot(x, asp = 1, pch = 19)
```




```
n = dim(x)[1]
p = dim(x)[2]

x.mean   = sapply(x, mean)
```

```r
x.cov    = cov(x)
x.invcov = solve(x.cov)
```

4.3.1 Standard test for the mean

```r
### Test for the mean of a multivariate Gaussian

################################################################################
# Premise: general rule to perform a test
# 1)  Formulate the test (and test the Gaussian assumption, if needed)
# 2)  Compute the test statistics
# 3a) Having set the level of the test, verify whether the test statistics
#     belongs to the region of rejection (ie, if there is statistical
#     evidence to reject H0)
# 3b) Compute the p-value of the test
################################################################################

#_______________________________________________________________________________
### Test on the mean of level alpha=1%
### H0: mu == mu0 vs H1: mu != mu0
### with mu0 = c(1, 0)
###-----------------------------------
mshapiro.test(t(x))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.96439, p-value = 0.3989
```

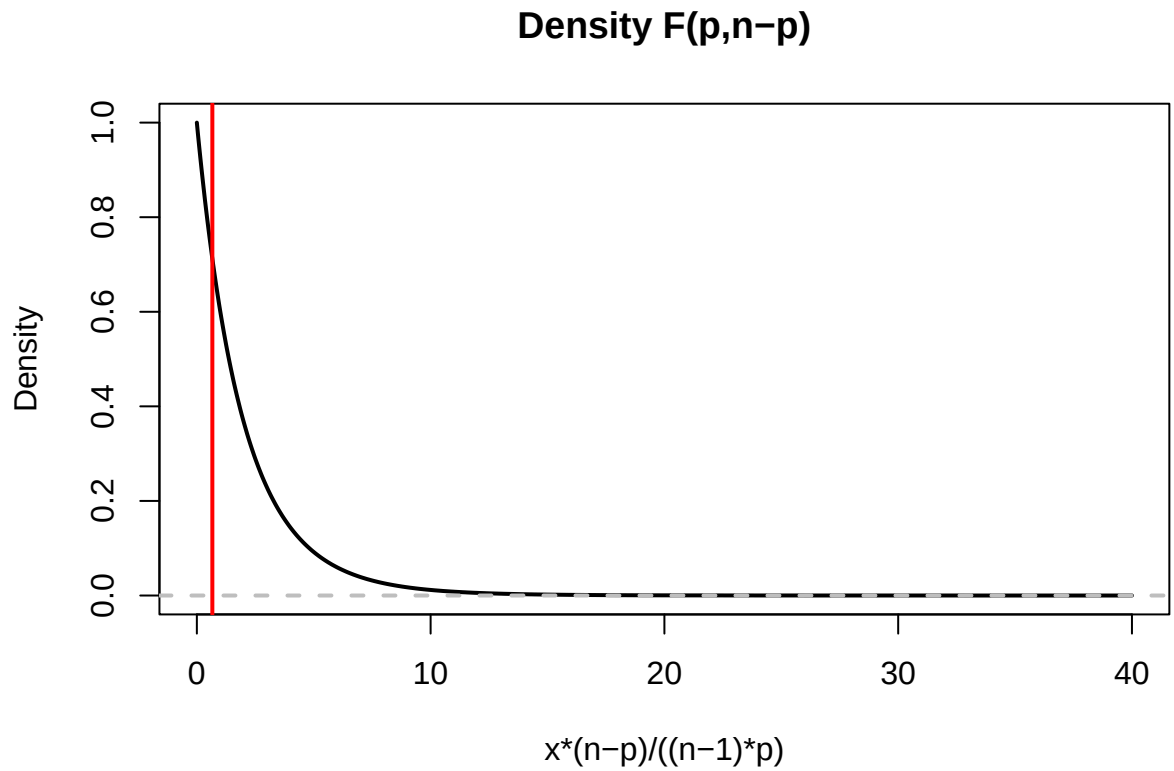
```r
alpha = 0.01
mu0 = c(1,0)

# Hotelling T2 Statistics
x.T2       = n * (x.mean-mu0) %*% x.invcov %*% (x.mean-mu0)
# Radius of the ellipsoid
cfr.fisher = ((n-1)*p/(n-p))*qf(1-alpha,p,n-p)
# Test:
x.T2 < cfr.fisher # no statistical evidence to reject H0 at level alpha
```

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

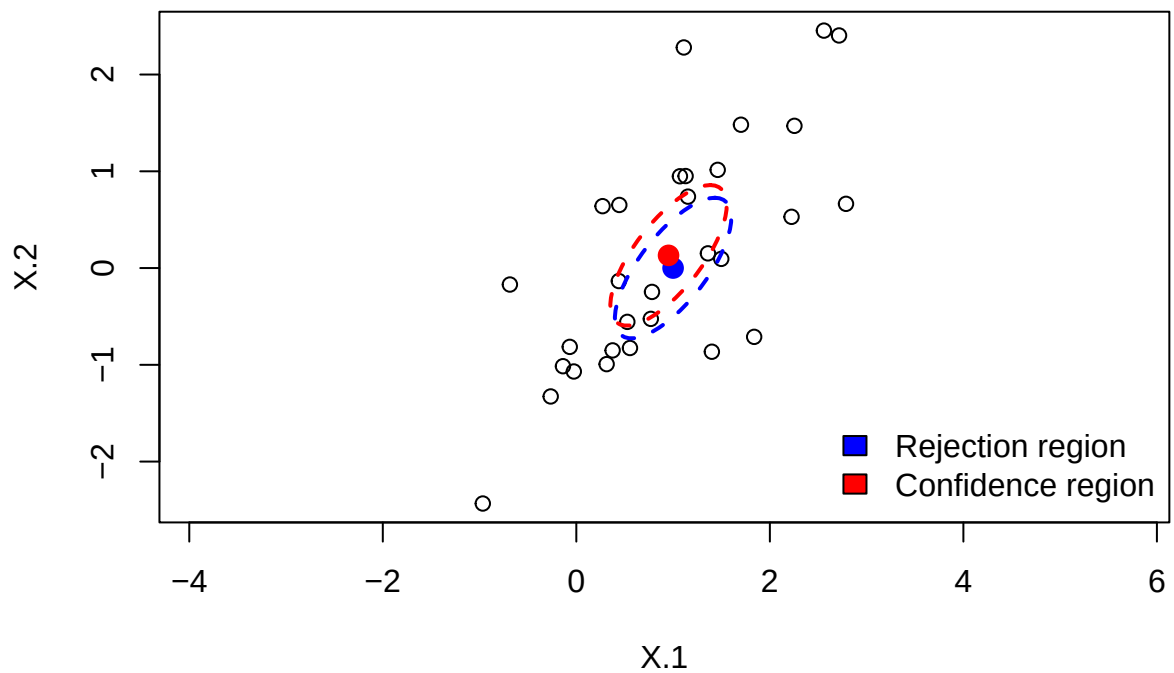
```r
#! that value for mu is inside the ellipsoide
# Rejection region: {x.T2 > cfr.fisher}
# (we reject for large values of the T2 statistics)

# Compute the p-value
P = 1 - pf(x.T2*(n-p) / ((n-1)*p), p, n-p)
P
```

```
##           [,1]
## [1,] 0.5229475
```

```r
xx = seq(0,40,by=0.05)
plot(xx, df(xx*(n-p)/((n-1)*p),p,n-p), type="l", lwd=2, main='Density F(p,n-p)',
     xlab='x*(n-p)/((n-1)*p)', ylab='Density')
abline(h=0, v=x.T2*(n-p)/((n-1)*p), col=c('grey','red'), lwd=2, lty=c(2,1))
```




```
# The P-value is high because the test statistics is central with respect to
# its distribution under H0
# => we cannot reject for any reasonable level (we would reject for a
# level alpha>81.7%)

# Region of rejection (centered in mu0)

plot(x, asp = 1)
ellipse(mu0, shape=x.cov/n, radius=sqrt(cfr.fisher), col = 'blue', lty = 2,
        center.pch = 16)
#! confidence region centered at the "true" mean, this is the acceptance region
#! So the values of xbar for which we accept H0

# We add a red point in correspondence of the sample mean
points(x.mean[1], x.mean[2], pch = 16, col ='red', cex = 1.5)

# Confidence region (centered in x.mean)
# { m \in R^2 s.t. n * (x.mean-m)' %*% (x.cov)^-1 %*% (x.mean-m) < cfr.fisher }

ellipse(x.mean, x.cov/n, sqrt(cfr.fisher), col = 'red', lty = 2, lwd=2, center.cex=1)
#! confidence region centered at the sample mean
#! And the machinery that generates this ellipse contains the true mean 99% of the times
#! So 99% is not the probability of the true mean being inside it, is a different thing

legend("bottomright",c("Rejection region","Confidence region"), fill=c("blue","red"),bty="n")
```




```
# Remark: the radius and the shape of the ellipse are the same, but the center changes:
# - Rejection region: the center is the mean mu0 under H0 (blue ellipse)
# - Confidence region: the center is the sample mean (red ellipse)

# Which relation between the two ellipses?
# - If the rejection region does NOT contain the sample mean (ie, we
#   are in the acceptance region), then we cannot reject H0
#   (ie, if the sample mean falls within the ellipse we accept H0)
# - If the mean under H0 (mu0) is contained in the confidence region
#   of level 1-alpha, then we do not reject H0 at level alpha
# => the confidence region of level 1-alpha contains all the mu0
#    that we would accept at level alpha

# Note: by definition, the confidence region of level 1-alpha
# produces ellipsoidal regions that contain the true mean
# 100(1-alpha)% of the times. If H0 is true (ie, mu0 is
# the true mean), those ellipsoidal regions will contain mu0
# 100(1-alpha)% of the times
```

4.3.2 Asymptotic test for the mean

```
### Asymptotic test on the mean
### H0: mu == mu0 vs H1: mu != mu0
### with mu0 = c(1,0)
###----------------------------------------------

# Note: we don't need to verify the Gaussian assumption!
# Warning: we are going to use an asymptotic test, but we only have n = 30 data!

mu0 = c(1,0)

#! n=30, so we have not a lot of data
#! And in fact we can see that the asymptotic comparison here fails
#! Especially in higher dimensions, we have to have 30*p^2 number of observations
#! where p is the number of columns/variables as usual

x.T2A   = n * (x.mean-mu0) %*%  x.invcov  %*% (x.mean-mu0)
```

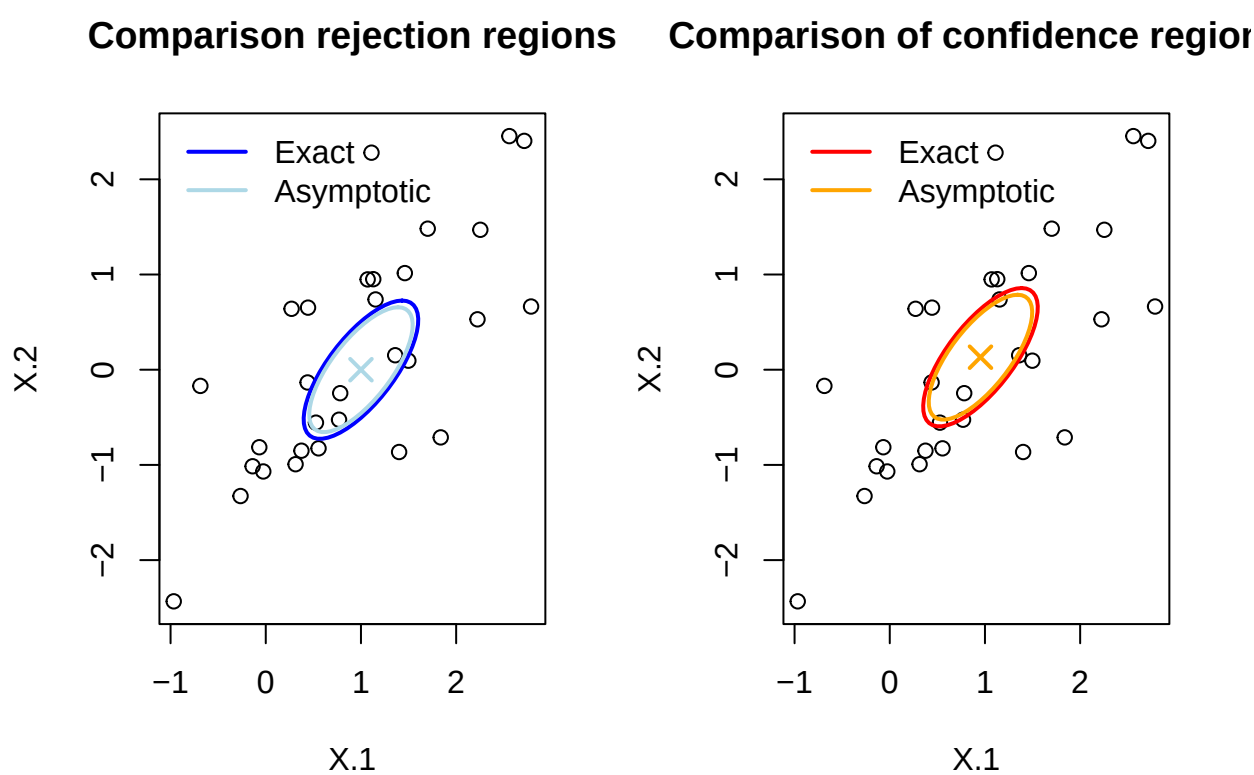
```r
cfr.chisq = qchisq(1-alpha, p)
x.T2A < cfr.chisq # no statistical evidence to reject H0 at level alpha
```

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

```r
# Compute the p-value
PA = 1 - pchisq(x.T2A, p)
PA
```

```
##           [,1]
## [1,] 0.5029749
```

```r
par(mfrow=c(1,2))
plot(x, asp = 1,main='Comparison rejection regions')
ellipse(mu0, shape=x.cov/n, sqrt(cfr.fisher), col = 'blue', lty = 1,
        center.pch = 4, center.cex=1.5, lwd=2)
ellipse(mu0, x.cov/n, sqrt(cfr.chisq), col = 'lightblue', lty = 1,
        center.pch = 4, center.cex=1.5, lwd=2)
points(mu0[1], mu0[2], pch = 4, cex = 1.5, lwd = 2, col ='lightblue')
legend('topleft', c('Exact', 'Asymptotic'), col=c('blue', 'lightblue'),
       lty=c(1), lwd=2, bty="n")

plot(x, asp = 1,main='Comparison of confidence regions')
ellipse(x.mean, x.cov/n, sqrt(cfr.fisher), col = 'red', lty = 1, center.pch = 4,
        center.cex=1.5, lwd=2)
ellipse(x.mean, x.cov/n, sqrt(cfr.chisq), col = 'orange', lty = 1, center.pch = 4,
        center.cex=1.5, lwd=2)
points(x.mean[1], x.mean[2], pch = 4, cex = 1.5, lwd = 2, col ='orange')
legend('topleft', c('Exact', 'Asymptotic'), col=c('red','orange'),
       lty=c(1), lwd=2, bty="n")
```




```r
#! the asymptotic one is smaller, so it can cause some fails/error
#! Ie we reject H0 looking at asymptotic, when actually the true test
#! would have make us accept H0
#! We lose control on the type 1 error (a "false discovery", reject H0 while
#! it was instead true), which is bad
```

```r
#! As when we make a test we want to prove we can reject H0, so we would like
#! to have the most accurate way to say we can reject it

#_____________________________________________________________________________
###  We change the null Hypothesis
###-------------------------------
mu0 = c(1.5,-0.5)
mu0
```

```
## [1]  1.5 -0.5
```

```r
mu
```

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

```r
# Remark: now H0 is false (we will want to reject H0)

### Test of level 1%
### H0: mu == mu0=c(1.5,-0.5) vs H1: mu != mu0=c(1.5,-0.5)
###--------------------------------------------------
```

4.3.3 Ellipsoidal region

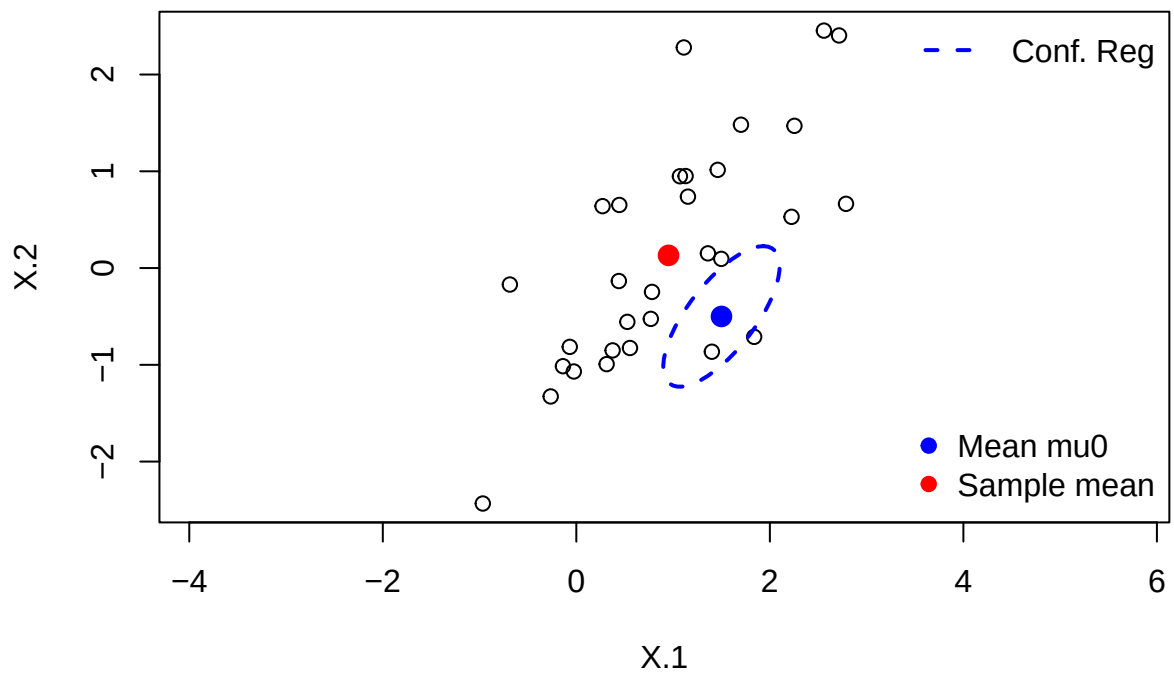
```r
### Ellipsoidal confidence region
x.T2 = n * t(x.mean-mu0) %*% x.invcov %*% (x.mean-mu0)
x.T2 < cfr.fisher
```

```
##       [,1]
## [1,] FALSE
```

```r
# Compute the p-value
P = 1 - pf(x.T2*(n-p)/((n-1)*p), p, n-p)
P
```

```
##              [,1]
## [1,] 9.292415e-08
```

```r
plot(x, asp = 1)
ellipse(mu0, shape=x.cov/n, sqrt(cfr.fisher), col = 'blue', lty = 2,
        center.pch = 16)
points(x.mean[1], x.mean[2], pch = 16, col = 'red', cex=1.5)
legend("bottomright",c("Mean mu0","Sample mean"),col=c("blue","red"),
       pch=c(19,19),bty="n")
legend("topright",c("Conf. Reg"),col="blue",lty=2,bty="n",lwd = 2)
```




```
# How would we communicate the results?
# For instance, we could provide confidence intervals along interesting
# directions (if they don't contain the mean under H0, it means that
# we reject the associated univariate test along that direction)
```

4.3.4 Simultaneous and Bonf CIs

```
# Let's try with simultaneous T2 confidence intervals on the coordinate directions:
# Recall: these are projections of the ellipsoidal confidence region

T2 = cbind(inf = x.mean - sqrt(cfr.fisher*diag(x.cov)/n),
           center = x.mean,
           sup = x.mean + sqrt(cfr.fisher*diag(x.cov)/n))
T2
```

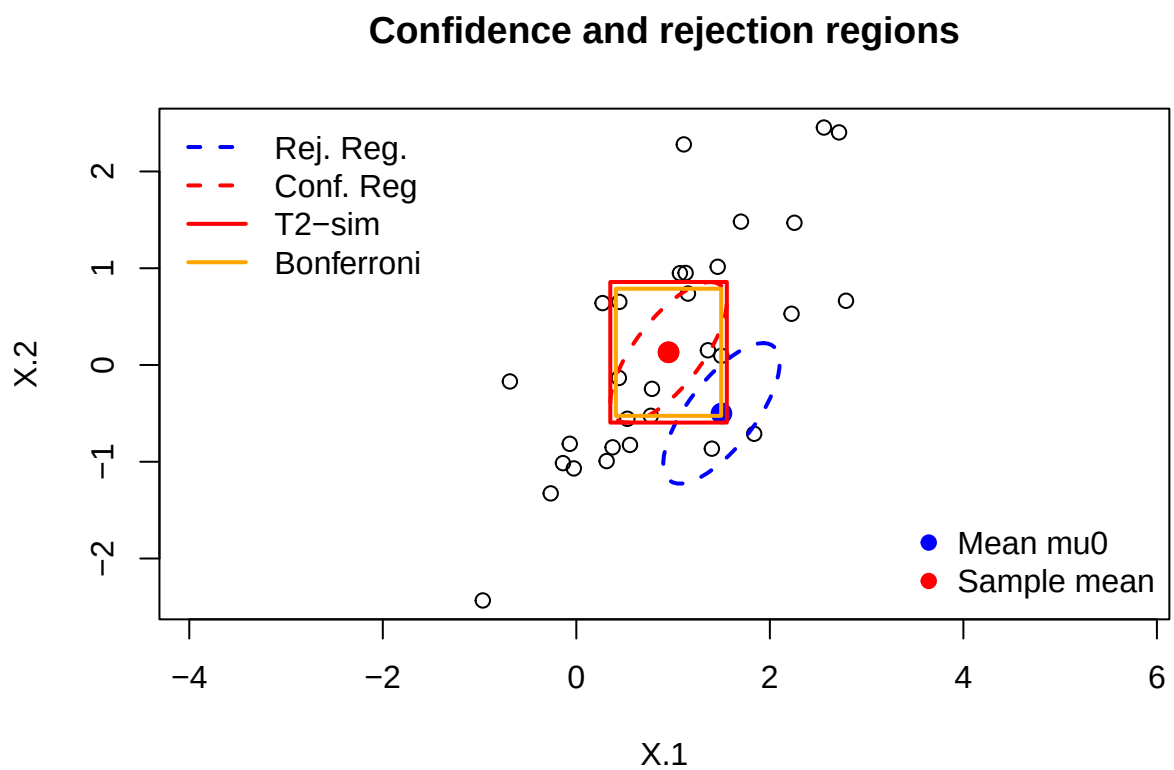
```
##            inf    center       sup
## X.1  0.3509299 0.9528962 1.5548626
## X.2 -0.5952772 0.1312346 0.8577464
```

```
# Both the intervals contain the mean under H0
# (ie, mu0 is contained in the rectangular region determined by
# the projection of the ellipsoid along the coordinate directions)

# Remark: this is not in contrast with the previous findings
# Rejecting the global T2-test means that we reject H0 along at least one
# direction, not necessarily along the coordinate direction

par(mfrow=c(1,1))
plot(x, asp = 1,main='Confidence and rejection regions')
ellipse(mu0, shape=x.cov/n, sqrt(cfr.fisher), col = 'blue', lty = 2,
        center.pch = 16)
points(x.mean[1], x.mean[2], pch = 16, col = 'red', cex=1.5)

ellipse(x.mean, shape=x.cov/n, sqrt(cfr.fisher), col = 'red', lty = 2,
        center.pch = 16)
rect(T2[1,1], T2[2,1], T2[1,3], T2[2,3], border='red', lwd=2)
```

```r
# Let's try with Bonferroni intervals
k = p # number of intervals I want to compute (set in advance)
cfr.t = qt(1-alpha/(2*k), n-1)
Bf = cbind(inf = x.mean - cfr.t*sqrt(diag(x.cov)/n),
           center = x.mean,
           sup = x.mean + cfr.t*sqrt(diag(x.cov)/n))
Bf
```

```
##            inf    center       sup
## X.1  0.4087490 0.9528962 1.4970435
## X.2 -0.5254955 0.1312346 0.7879647
```

```r
# Both the intervals contain the mean under H0
# (ie, mu0 is contained in the rectangular region determined by
# the Bonferroni intervals along the coordinate directions)

# we add the Bonferroni intervals to the plot
rect(Bf[1,1], Bf[2,1], Bf[1,3], Bf[2,3], border='orange', lwd=2)
legend('topleft', c('Rej. Reg.', 'Conf. Reg','T2-sim', 'Bonferroni'),
       col=c('blue','red','red','orange'), lty=c(2,2,1,1), lwd=2, bty="n")
legend("bottomright",c("Mean mu0","Sample mean"),col=c("blue","red"),bty="n",pch=19)
```




```r
# Remark: if we wanted to compute additional Bonferroni intervals
# along other directions, we would need to re-compute all the Bonferroni
# intervals with another correction k
```

4.4 On dataset stiff

```r
### Example: analysis of a real dataset (dataset stiff)

stiff = read.table('Lab04/stiff.dat')
head(stiff)
```

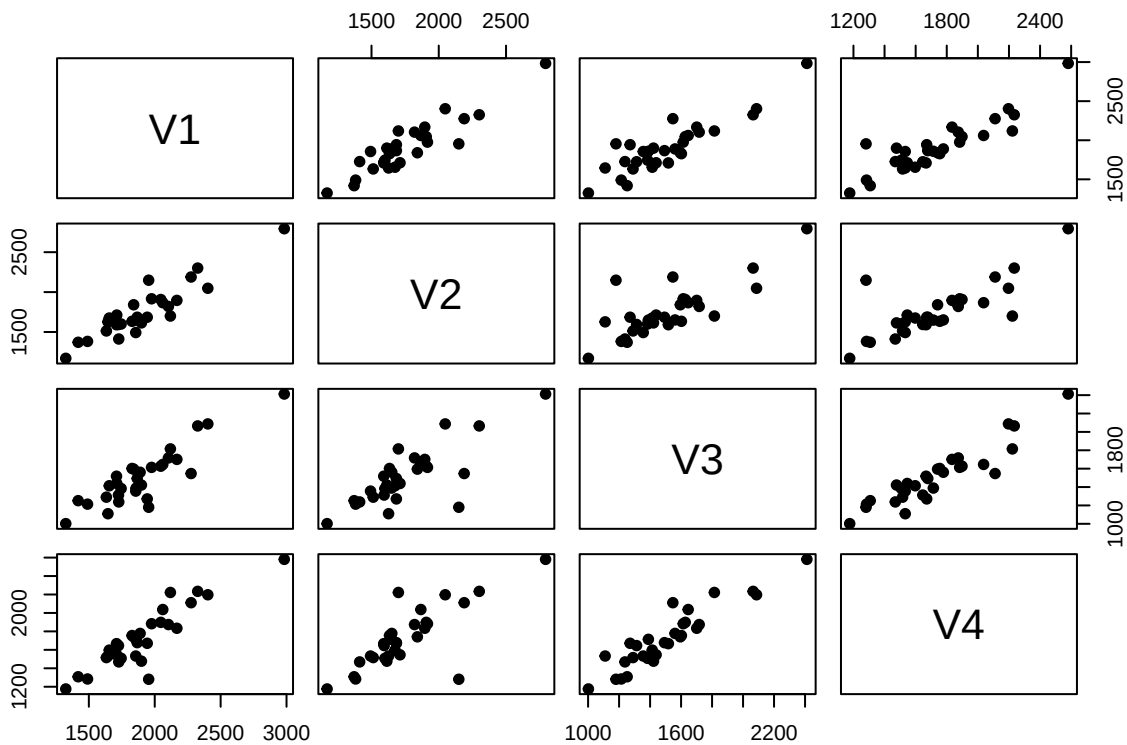
```
##     V1   V2   V3   V4
## 1 1889 1651 1561 1778
## 2 2403 2048 2087 2197
```

```
## 3 2119 1700 1815 2222
## 4 1645 1627 1110 1533
## 5 1976 1916 1614 1883
## 6 1712 1712 1439 1546
```

```
dim(stiff)
```

```
## [1] 30  4
```

```
n = dim(stiff)[1]
p = dim(stiff)[2]

plot(stiff,pch=19)
```




```
### Normality test
mvn(stiff)$multivariateNormality
```

```
##            Test       HZ     p value MVN
## 1 Henze-Zirkler 1.038131 0.005960236  NO
```

```
mvn(stiff_wo_outliers)$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.6220241 0.8032532 YES
```

```
n = dim(stiff_wo_outliers)[1]
p = dim(stiff_wo_outliers)[2]
```

4.4.1 Test for the mean

```
### Test for the mean of level 5%
###--------------------------------
# 1) Formulate the test:
#    H0: mu == mu0 vs H1: mu != mu0
```

```r
#    with mu0=c(1850, 1750, 1500, 1700)

mu0   = c(1850, 1750, 1500, 1700)
alpha = 0.05

# 2) Compute the test statistics
x.mean   = colMeans(stiff_wo_outliers)
x.cov    = cov(stiff_wo_outliers)
x.invcov = solve(x.cov)

x.T2 = n * (x.mean-mu0) %*% x.invcov %*% (x.mean-mu0)
#! the Hotelling statistic

# 3a) Verify if the test statistics belongs to the rejection region
cfr.fisher = ((n-1)*p/(n-p))*qf(1-alpha,p,n-p)
x.T2 < cfr.fisher
```

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

```r
# we accept H0 at 5%

# 3b) Compute the p-value
P = 1 - pf(x.T2*(n-p)/((n-1)*p), p, n-p)
P
```

```
##            [,1]
## [1,] 0.05526597
```

```r
print(paste("We reject H0 at level 5%?", P<0.05))
```

```
## [1] "We reject H0 at level 5%? FALSE"
```

```r
print(paste("We reject H0 at level 6%?", P<0.06))
```

```
## [1] "We reject H0 at level 6%? TRUE"
```

```r
#! la prob di osservare un valore pari a (o più estremo di) T2 più è alta
#! più vuol dire che quel valore osservato era probabile che succedesse,
#! perciò è coerente coi dati e accettiamo H0
```

4.4.2 CR for the mean

```r
### Confidence region for the mean of level 95%
###-------------------------------------------------------------
# 1)  Identify the type of region of interest:
#     CR for the mean (ellipsoidal region)
#     { m \in R^4 t.c. n * (x.mean-m)' %*% x.invcov %*% (x.mean-m)
#                                                    < cfr.fisher }
# 2)  Characterize the region: compute the center, direction of
#     the principal axes, length of the axes

# Center:
x.mean
```

```
##       V1       V2       V3       V4
## 1850.741 1679.778 1486.481 1695.407
```

```r
# Directions of the principal axes:
eigen(x.cov/n)$vectors
```

```
##            [,1]        [,2]       [,3]       [,4]
## [1,] -0.5031044 -0.01315709  0.6877349 -0.5231955
## [2,] -0.4408073 -0.85419934 -0.1797116  0.2091320
## [3,] -0.5093681  0.30440061 -0.6864989 -0.4202425
## [4,] -0.5414047  0.42132013  0.1531134  0.7112851
```

```r
# Length of the semi-axes of the ellipse:
r = sqrt(cfr.fisher)
r*sqrt(eigen(x.cov/n)$values)
```

```
## [1] 333.20113  62.64886  56.12788  49.92648
```

```r
#! these are the axes of the ellipsoide of R4
#! but like one is very long, one intermediate and the other two small and similar

# Warning: Conf Reg => x.cov/n

# Question: how to plot the confidence region?
# (I don't know how to plot in R^4!)
# I can work with 'representations' of the confidence regions
# (e.g., projections along particular directions)

# We plot the projections of the ellipsoid in some directions
# of interest (e.g. the x and y coordinates)
# => We plot the simultaneous T2 confidence intervals in each
#    direction of interest (with global coverage alpha)
```

4.4.3 Simultaneous CIs

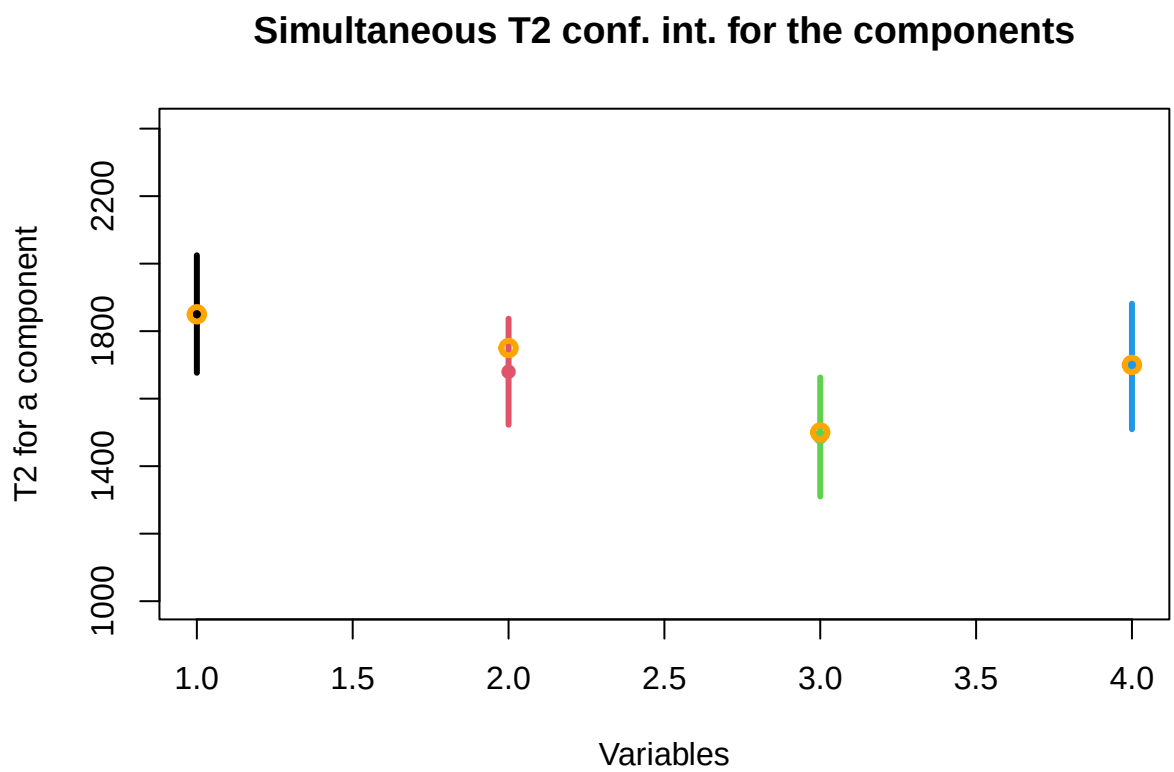
```r
### Simultaneous T2 confidence intervals on the (four) components of the mean
### with global level 95%
###-----------------------------------------------------

T2 = cbind(inf = x.mean - sqrt(cfr.fisher*diag(x.cov)/n),
            center = x.mean,
            sup = x.mean + sqrt(cfr.fisher*diag(x.cov)/n))
T2
```

```
##         inf   center      sup
## V1 1676.745 1850.741 2024.737
## V2 1522.782 1679.778 1836.773
## V3 1310.146 1486.481 1662.817
## V4 1509.465 1695.407 1881.350
```

```r
#! and compare it with
mu0
```

```
## [1] 1850 1750 1500 1700
```

```r
matplot(1:4,1:4, pch='',ylim=range(stiff_wo_outliers), xlab='Variables',
        ylab='T2 for a component',
       main='Simultaneous T2 conf. int. for the components')
for(i in 1:4) segments(i, T2[i,1], i, T2[i,3], lwd=3, col=i)
points(1:4, T2[,2], pch=16, col=1:4)

# Is mu0 inside the rectangular region?
# We add it to the plot
points(1:4, mu0, lwd=3, col='orange')
```




```
# Yes, it is, because it is inside all the T2-intervals,
```

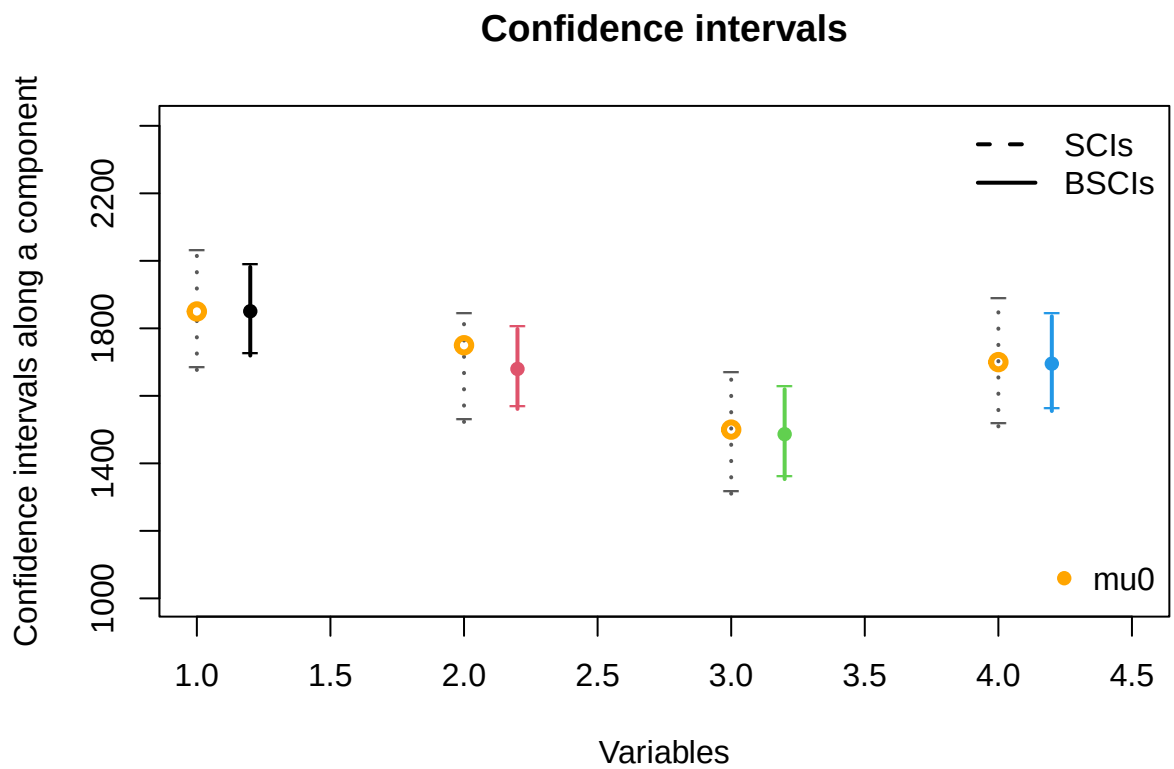
4.4.4 Bonferroni CIs

```
### Bonferroni simultaneous intervals on the components of the mean
### with global level 95%

#! so this is an overallconfidence (before 95% maybe was for each CI)
#! and with Bonferroni we expect a narrower CIs
#! as they depend on the number n of directions (here 4)
#! while the previous one wont change

#! Before is mu0 was inside the ellipsoide, then surely the component will be
#! inside their CI  while with Bonf CI we are not sure of that, some components
#! may be outside
###-----------------------------------------------------
k = p
cfr.t = qt(1 - alpha/(k*2), n-1)

Bf = cbind(inf = x.mean - cfr.t*sqrt(diag(x.cov)/n),
           center = x.mean,
           sup = x.mean + cfr.t*sqrt(diag(x.cov)/n))
Bf
```

```
##         inf   center      sup
## V1 1719.409 1850.741 1982.073
## V2 1561.278 1679.778 1798.278
## V3 1353.384 1486.481 1619.579
## V4 1555.058 1695.407 1835.757
```

```
# Let's do a plot

matplot(1:4, 1:4, pch='', ylim=range(stiff_wo_outliers),xlim=c(1,4.5), xlab='Variables',
        ylab='Confidence intervals along a component', main='Confidence intervals')

for(i in 1:4) segments(i, T2[i,1], i, T2[i,3], lwd=2, col='grey35', lty=3)
points(1:4, T2[,1], pch='-', col='grey35')
```

```
points(1:4, T2[,3], pch='-', col='grey35')

delta=1/5
for(i in 1:4) segments(i+delta, Bf[i,1], i+delta, Bf[i,3], lwd=2, col=i)
points(1+delta:4, Bf[,2], pch=16, col=1+delta:4+delta)
points(1+delta:4, Bf[,1], pch='-', col=1+delta:4+delta)
points(1+delta:4, Bf[,3], pch='-', col=1+delta:4+delta)

# Is mu0 inside the Bonferroni confidence region?
# we add it to the plot
points(1:4, mu0, lwd=3, col='orange')
legend("topright",c("SCIs","BSCIs"),col=c("black","black"),lty=c(2,1),lwd=2,bty="n")
legend("bottomright",c("mu0"),col="orange",pch=16,bty="n")
```




```
# Yes, it is, because it belongs to all the intervals along the components
```

5 Lab 05 - Tests for multivariate gaussians

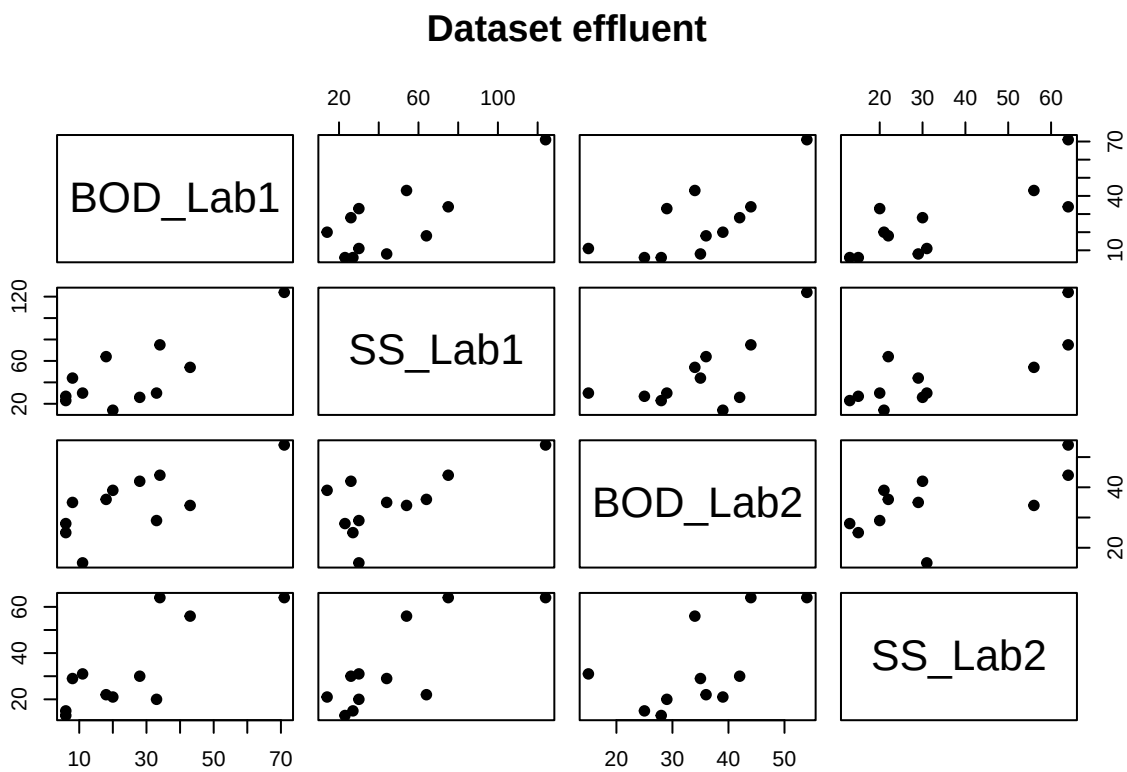
```
###--------------------###
### LAB 5 (24/03/2023) ###
###--------------------###

### TOPICS:
### Test for the mean of paired multivariate Gaussian observations
### Test for repeated measures
### Test for two independent Gaussian populations
### Exercises
#-------------------------------------------------------------------------------

library(car)
library(MVN)
```

5.1 Paired multivariate Gaussians

```r
##### Test for the mean of paired multivariate Gaussian observations
#####-------------------------------------------------------------------

###   Example 6.1 JW + exercise 6.1 JW
###   (n=11, p=2)
###-------------------------------------
# Municipal wastewater treatment plants are required by law to monitor
# their discharges into rivers and streams on a regular basis. Concern
# about the reliability of data from one of these self-monitoring
# programs led to a study in which samples of effluent were divided
# and sent to two laboratories for testing. One-half of each sample
# was sent to the Wisconsin State Laboratory of Hygiene, and one-half
# was sent to a private commercial laboratory routinely used in the
# monitoring program. Measurements of biochemical oxygen demand (BOD)
# and suspended solids (SS) were obtained, for n=11 sample splits,
# from the two laboratories.
# (The experimenter divided each sample by first shaking it and then
# pouring it rapidly into two bottles in order to avoid difference
# in the suspended solids contained in the two half-samples)

# Do the two laboratories' chemical analyses agree?

effluent = read.table('Lab05/effluent.dat')
effluent
```

```
##    V1  V2 V3 V4
## 1   6  27 25 15
## 2   6  23 28 13
## 3  18  64 36 22
## 4   8  44 35 29
## 5  11  30 15 31
## 6  34  75 44 64
## 7  28  26 42 30
## 8  71 124 54 64
## 9  43  54 34 56
## 10 33  30 29 20
## 11 20  14 39 21
```

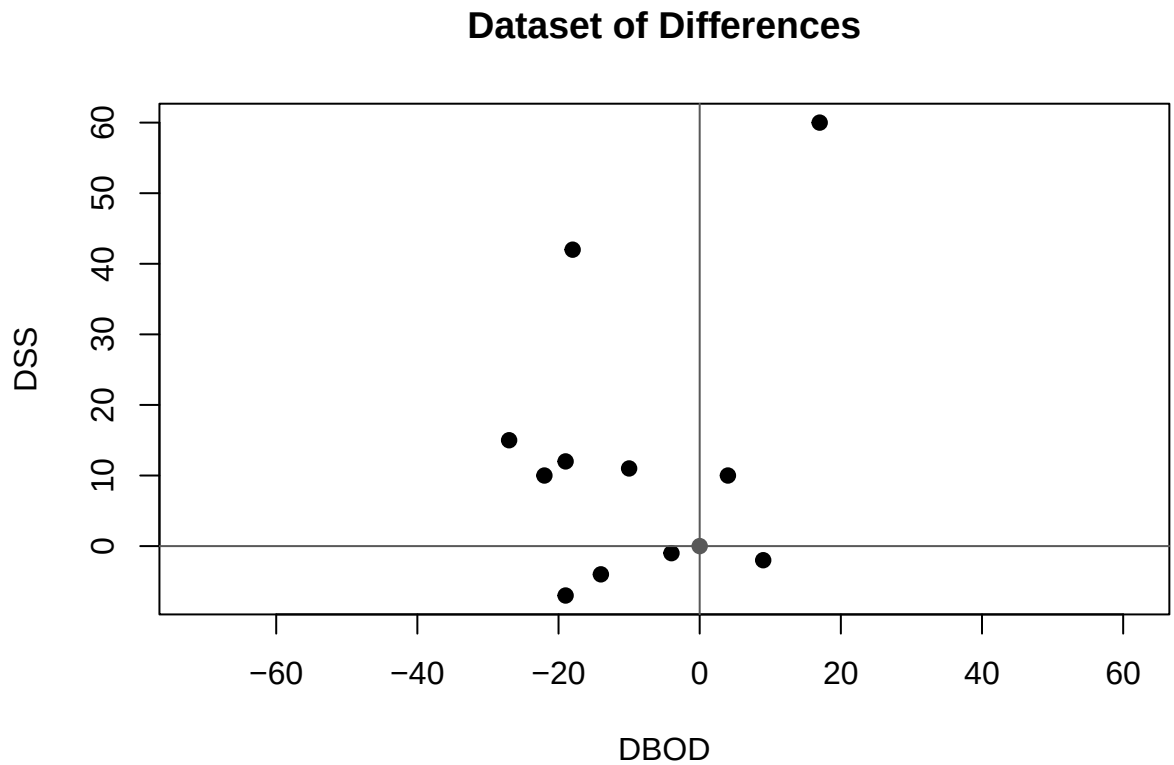
```r
# We change the column names to have the correct label
colnames(effluent) = c('BOD_Lab1', 'SS_Lab1', 'BOD_Lab2', 'SS_Lab2')

pairs(effluent, pch=19, main='Dataset effluent')
```




```r
# we compute the sample of differences
D = data.frame(
  DBOD = effluent$BOD_Lab1 - effluent$BOD_Lab2,
  DSS  = effluent$SS_Lab1 - effluent$SS_Lab2
)
D
```

```
##    DBOD DSS
## 1   -19  12
## 2   -22  10
## 3   -18  42
## 4   -27  15
## 5    -4  -1
## 6   -10  11
## 7   -14  -4
## 8    17  60
## 9     9  -2
## 10    4  10
## 11  -19  -7
```

```r
# Scatter plot of the dataset of differences with the four quadrants
plot(D, asp=1, pch=19, main='Dataset of Differences')
abline(h=0, v=0, col='grey35')
points(0,0, pch=19, col='grey35')
```




```
# DBOD: difference in 'biochemical oxygen demand'
# DSS:  difference in 'suspended solids'
#       measured by the two laboratories
```

5.1.1 Test for the mean on D

```
# Now we can proceed as we already know, but working on D

### T2 Hotelling Test
# H0: delta == delta.0 vs H1: delta != delta.0
# with delta.0 = c(0,0)

# Test the Gaussian assumption (on D!)
result = mvn(data = D)
result$multivariateNormality
```

```
##            Test        HZ    p value MVN
## 1 Henze-Zirkler 0.5825213 0.07940244 YES
```

```
# The p-value isn't very high (but I don't reject for levels 5%, 1%).
# There might be outliers, but we don't remove them because we only have
# very few data

n = dim(D)[1]  # 11
p = dim(D)[2]  #  2

D.mean   = sapply(D,mean)
D.cov    = cov(D)
D.invcov = solve(D.cov)

alpha   = .05
delta.0 = c(0,0)

D.T2 = n * (D.mean-delta.0) %*% D.invcov %*% (D.mean-delta.0)
D.T2
```

```
##          [,1]
## [1,] 13.63931
```

```r
cfr.fisher = ((n-1)*p/(n-p))*qf(1-alpha,p,n-p)
cfr.fisher
```

```
## [1] 9.458877
```

```r
D.T2 < cfr.fisher # FALSE: we reject H0 at level 5%
```

```
##       [,1]
## [1,] FALSE
```

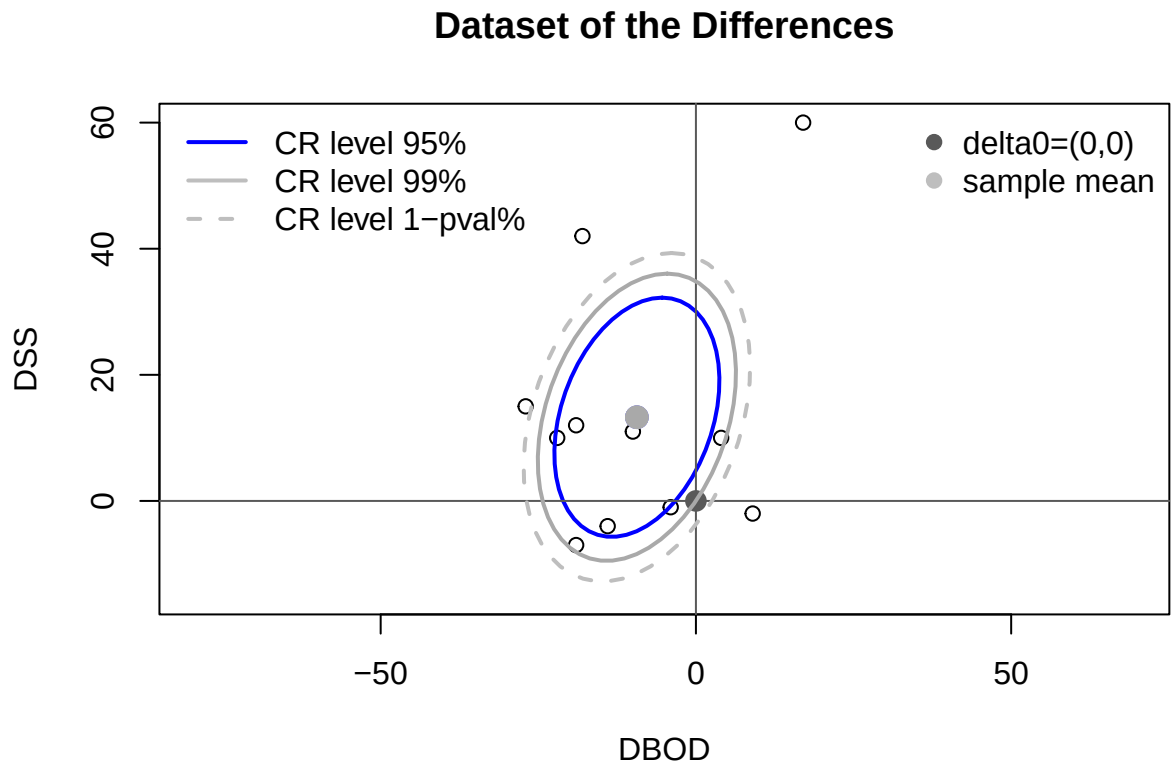
```r
# we compute the p-value
P = 1 - pf(D.T2*(n-p)/(p*(n-1)), p, n-p)
P
```

```
##            [,1]
## [1,] 0.02082779
```

```r
# reject H0 at 5% (don't reject at 1%)
```

5.1.2 CR for the mean of D

```r
# Ellipsoidal confidence region with confidence level 95%

plot(D, asp=1, pch=1, main='Dataset of the Differences',ylim=c(-15,60))
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=2)

# Adding delta.0 and the quadrants
points(delta.0[1], delta.0[2], pch=16, col='grey35', cex=1.5)
abline(h=delta.0[1], v=delta.0[2], col='grey35')

# Ellipsoidal confidence region with confidence level 99%
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt((n-1)*p/(n-p)*qf(1-0.01,p,n-p)),
        lty=2,col='grey',lwd=2)

# What if we set the radius as the quantile of order 1-pval?
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt((n-1)*p/(n-p)*qf(1-as.numeric(P),p,n-p)),
        lty=1, col='dark grey', lwd=2)
legend("topright",c("delta0=(0,0)","sample mean"),col=c("grey35","grey"),bty="n",pch=19)
legend("topleft",c("CR level 95%","CR level 99%","CR level 1-pval%"),
       col=c("blue","gray","gray"),lty=c(1,1,2),lwd=2,bty="n")
```




```
#! Our blue ellipse does not contain the target mena delta0, so we reject H0, we cant
#! say that the two analysis performed are equivalent
```

5.1.3 SCIs for lin comb of components

```
# Now, let's communicate our results to the client (because we need to take our salary!)
# Let's build confidence intervals for linear combination of the
# components of the mean vector

### Simultaneous T2 intervals in the direction of DBOD and DSS
IC.T2.DBOD = c( D.mean[1] - sqrt(cfr.fisher*D.cov[1,1]/n), D.mean[1],
                 D.mean[1] + sqrt(cfr.fisher*D.cov[1,1]/n))
IC.T2.DSS  = c( D.mean[2] - sqrt(cfr.fisher*D.cov[2,2]/n), D.mean[2],
                 D.mean[2] + sqrt(cfr.fisher*D.cov[2,2]/n))

T2 = rbind(IC.T2.DBOD, IC.T2.DSS)
dimnames(T2)[[2]] = c('inf','center','sup')
T2
```

```
##                    inf    center      sup
## IC.T2.DBOD -22.453272 -9.363636  3.72600
## IC.T2.DSS   -5.700119 13.272727 32.24557
```

```
# Plot of the simulatenous T2 intervals

plot(D, asp=1, pch=1, main='Dataset of the Differences',ylim=c(-15,60))
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=2, col='grey')
abline(v = T2[1,1], col='red', lwd=1, lty=2)
abline(v = T2[1,3], col='red', lwd=1, lty=2)
abline(h = T2[2,1], col='red', lwd=1, lty=2)
abline(h = T2[2,3], col='red', lwd=1, lty=2)

points(delta.0[1], delta.0[2], pch=16, col='grey35', cex=1.5)
abline(h=delta.0[1], v=delta.0[2], col='grey35')

segments(IC.T2.DBOD[1], 0, IC.T2.DBOD[3], 0, lty=1, lwd=2, col='red')
segments(0, IC.T2.DSS[1], 0, IC.T2.DSS[3], lty=1,lwd=2, col='red')
```

```r
# We don't have enough evidence to reject H0 in any coordinate axes-direction
# But we can reject the global test of (global) level 5%
# => there exists at least one direction along which we are allowed
#    to reject the univariate test

# Recall:
# We reject the global H0 if in at least one direction we observe a 'high'
# value of the statistics T2 (univariate), ie, we reject the global
# H0 if we reject the univariate test at least in direction (max(T2))

# Hence:
# Worst direction: direction along which the T2 statistics (univariate)
# is maximized

# => From the theory
#    - the maximum is realized (Hotelling T2-statistics)
D.T2
```

```
##          [,1]
## [1,] 13.63931
```

```r
#    - the distribution of the maximum is known
#    - the direction along which the maximum is realized is known
worst = D.invcov %*% (D.mean-delta.0)
worst = worst/sqrt(sum(worst^2)) # Normalization
worst
```

```
##            [,1]
## DBOD -0.8262423
## DSS   0.5633149
```

```r
# Confidence interval along the worst direction:
IC.worst  = c(D.mean %*% worst - sqrt(cfr.fisher*(t(worst) %*% D.cov %*% worst) / n),
              D.mean %*% worst,
              D.mean %*% worst + sqrt(cfr.fisher*(t(worst) %*% D.cov %*% worst) / n))
IC.worst
```

```
## [1]  2.544174 15.213357 27.882540
```

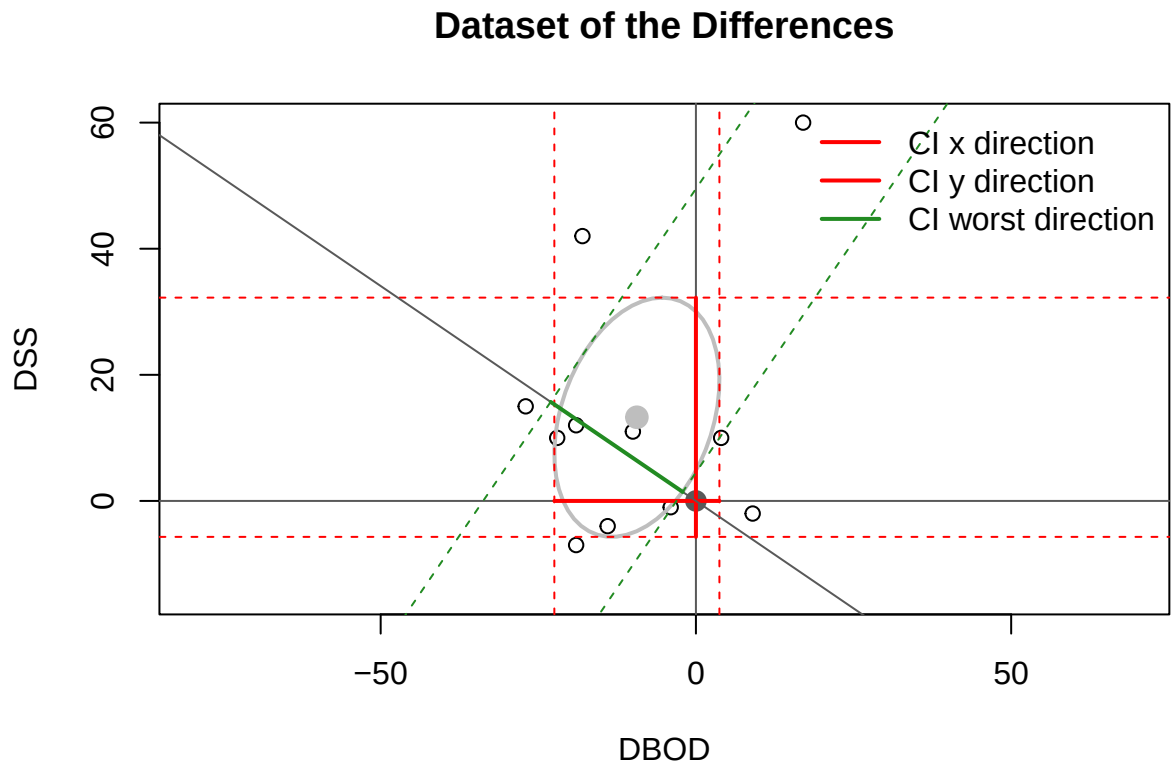
```r
delta.0 %*% worst
```

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

```r
(IC.worst[1] < delta.0 %*% worst) & (delta.0 %*% worst < IC.worst[3])
```

```
##       [,1]
## [1,] FALSE
```

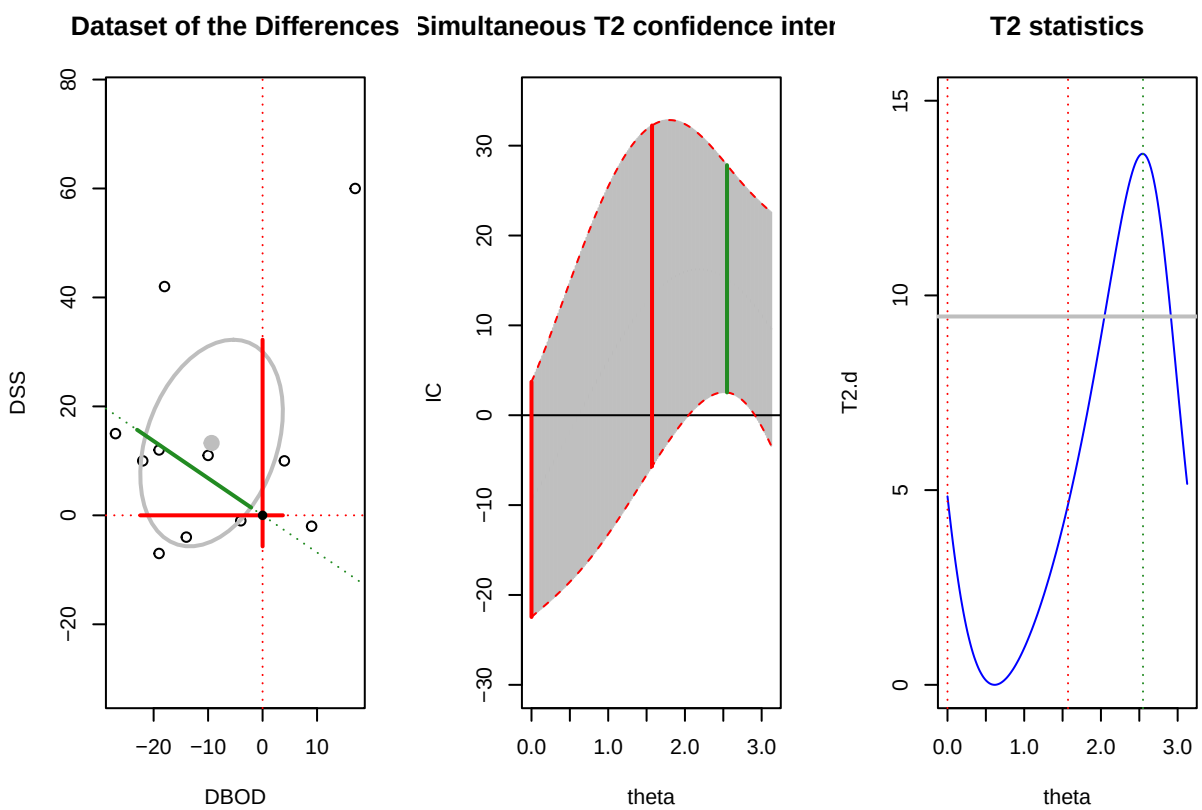
```r
# Reject H0: a'mu == a'delta.0 in direction a = worst

# Extremes of IC.worst in the coordinate system (x,y):
x.min = IC.worst[1] * worst # (x,y) coords of the lower bound of the interval
x.max = IC.worst[3] * worst # (x,y) coords of the upper bound of the interval
m1.ort = - worst[1] / worst[2] # Slope of the line orthogonal to worst
q.min.ort = x.min[2] - m1.ort * x.min[1] # Intercept of line of slope m1.ort
# passing by x.min
q.max.ort = x.max[2] - m1.ort * x.max[1] # Intercept of line of slope m1.ort
# passing by x.max
abline(q.min.ort, m1.ort, col='forestgreen', lty=2,lwd=1)
abline(q.max.ort, m1.ort, col='forestgreen', lty=2,lwd=1)

m1 = worst[2] / worst[1] # worst direction
abline(0, m1, col='grey35') # Intercept 0 because the line has to pass
# by delta.0 which is (0, 0)
segments(x.min[1], x.min[2], x.max[1], x.max[2], lty=1, lwd=2, col='forestgreen')

legend("topright",c("CI x direction","CI y direction","CI worst direction"),
       col=c("red","red","forestgreen"),lty=1,lwd=2,bty="n")
```



5.1.4 SCIs on all directions

```
# If we are not convinced yet, let's look at all the directions:
# we compute confidence intervals for a'x where a varies in all the
# directions between 0 and pi, with step pi/180. For each direction
# we compute the T2 statistic (univariate)
D = as.matrix(D)
theta   = seq(0, pi - pi/180, by = pi/180)
T2.d     = NULL
Centerf  = NULL
Maxf     = NULL
Minf     = NULL

for(i in 1:length(theta))
{
  a   = c(cos(theta[i]), sin(theta[i]))
  t2  = ( mean(D %*% a) - (delta.0 %*% a) )^2 / ( var(D %*% a) / n )
  T2.d  = c(T2.d, t2)
  centerf  = D.mean %*% a
  maxf     = D.mean %*% a + sqrt( t(a) %*% D.cov%*% a / n) * sqrt(cfr.fisher)
  minf     = D.mean %*% a - sqrt( t(a) %*% D.cov%*% a / n) * sqrt(cfr.fisher)
  Centerf  = c(Centerf, centerf)
  Maxf     = c(Maxf, maxf)
  Minf     = c(Minf, minf)
}

par(mfrow=c(1,3))
# Scatter plot of DSS and DBOD
plot(D, asp=1, pch=1, main='Dataset of the Differences', ylim=c(-15,60))
# Adding the quadrants
abline(h=delta.0[1], v=delta.0[2], col='red', lty=3)

# Adding the 95% confidence region for the true mean
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=2, col='grey')

# Adding the simultaneous confidence intervals in the direction of DSS and DBOD
segments(IC.T2.DBOD[1],0,IC.T2.DBOD[3],0,lty=1,lwd=2,col='red')
segments(0,IC.T2.DSS[1],0,IC.T2.DSS[3],lty=1,lwd=2,col='red')
```

```r
# Adding the simultaneous confidence interval in the "worst" direction
x.min = IC.worst[1] * worst
x.max = IC.worst[3] * worst
segments(x.min[1], x.min[2], x.max[1], x.max[2], lty=1, lwd=2, col='forestgreen')
abline(0, m1, col='forestgreen', lty=3)
points(delta.0[1], delta.0[2], pch=16, col='black')

# Second plot with the confidence intervals for each direction tested
plot(theta, Centerf, main = 'Simultaneous T2 confidence intervals', ylim = c(-30,35),
     col = 'grey25', type='l',ylab='IC')
for(i in 1:length(theta))
  lines(c(theta[i], theta[i]), c(Minf[i], Maxf[i]), col = 'grey75')
abline(h=0, col='black')
lines(theta, Minf, col = 'red', lty = 2)
lines(theta, Maxf, col = 'red', lty = 2)

# Highlighting the directions of DSS and DBOD (theta = 0 and theta = pi/2)
lines(c(theta[1], theta[1]), c(Minf[1], Maxf[1]), col = 'red', lwd=2)
lines(c(theta[91], theta[91]), c(Minf[91], Maxf[91]), col = 'red', lwd=2)

# Highlighting the "worst" direction for the rejection of H0
lines(c(theta[which.max(T2.d)], theta[which.max(T2.d)]),
      c(Minf[which.max(T2.d)], Maxf[which.max(T2.d)]), col = 'forestgreen', lwd=2)

# Third plot with the value of T2 statistic
plot(theta, T2.d, main = 'T2 statistics', ylim = c(0,15), col = 'blue', type='l')
abline(v=c(0,pi/2), col = 'red', lty = 3)
abline(v=theta[which.max(T2.d)], col = 'forestgreen', lty = 3)

# Drawing the threshold given by the Fisher quantile at level .95
abline(h=cfr.fisher, col = 'grey', lty = 1, lwd=2)
```



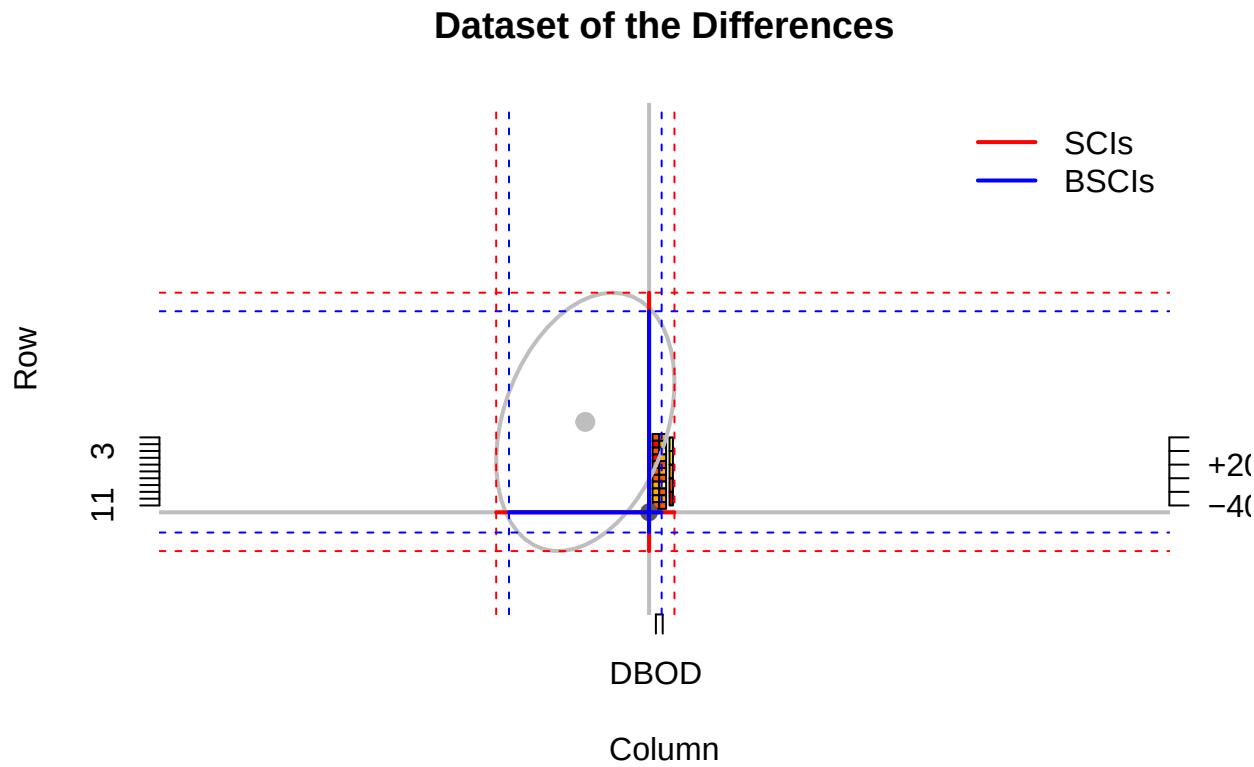
5.1.5 Bonferroni CIs

```r
### Bonferroni intervals
k = p  # Number of Bonferroni intervals
cfr.t = qt(1-alpha/(2*k), n-1) # Student quantile
```

```r
# Bonferroni confidence intervals in the direction of DBOD and DSS
IC.BF.DBOD = c(D.mean[1] - cfr.t*sqrt(D.cov[1,1]/n),
               D.mean[1],
               D.mean[1] + cfr.t*sqrt(D.cov[1,1]/n))
IC.BF.DSS  = c(D.mean[2] - cfr.t*sqrt(D.cov[2,2]/n),
               D.mean[2],
               D.mean[2] + cfr.t*sqrt(D.cov[2,2]/n))
# Bonferroni region defined by the cartesian product of the Bf intervals
Bf = rbind(IC.BF.DBOD, IC.BF.DSS)
dimnames(Bf)[[2]] = c('inf','center','sup')
Bf
```

```
##                    inf    center       sup
## IC.BF.DBOD -20.573107 -9.363636  1.845835
## IC.BF.DSS   -2.974903 13.272727 29.520358
```

```r
# Scatter plot of DSS and DBOD
plot(D, asp=1, pch=1, main='Dataset of the Differences',ylim=c(-15,60))

# Adding the 95% confidence region for the true mean of the differences
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=2, col='grey',
        center.cex=1.25)

# Adding quadrant lines and delta.0
abline(h=0, v=0, col='grey', lty=1, lwd=2)
points(delta.0[1], delta.0[2], pch=16, col='grey35', cex=1.25)

# Simultaneous confidence intervals in the direction of DSS and DBOD
abline(v = T2[1,1], col='red', lwd=1, lty=2)
abline(v = T2[1,3], col='red', lwd=1, lty=2)
abline(h = T2[2,1], col='red', lwd=1, lty=2)
abline(h = T2[2,3], col='red', lwd=1, lty=2)
segments(IC.T2.DBOD[1], 0, IC.T2.DBOD[3], 0, lty=1, lwd=2, col='red')
segments(0, IC.T2.DSS[1], 0, IC.T2.DSS[3], lty=1, lwd=2, col='red')

# Bonferroni intervals in the direction of DSS and DBOD
abline(v = Bf[1,1], col='blue', lwd=1, lty=2)
abline(v = Bf[1,3], col='blue', lwd=1, lty=2)
abline(h = Bf[2,1], col='blue', lwd=1, lty=2)
abline(h = Bf[2,3], col='blue', lwd=1, lty=2)
segments(IC.BF.DBOD[1], 0, IC.BF.DBOD[3], 0, lty=1, lwd=2, col='blue')
segments(0, IC.BF.DSS[1], 0, IC.BF.DSS[3], lty=1, lwd=2, col='blue')

legend("topright",c("SCIs","BSCIs"),col=c("red","blue"),lwd=2,lty=1,bty="n")
```

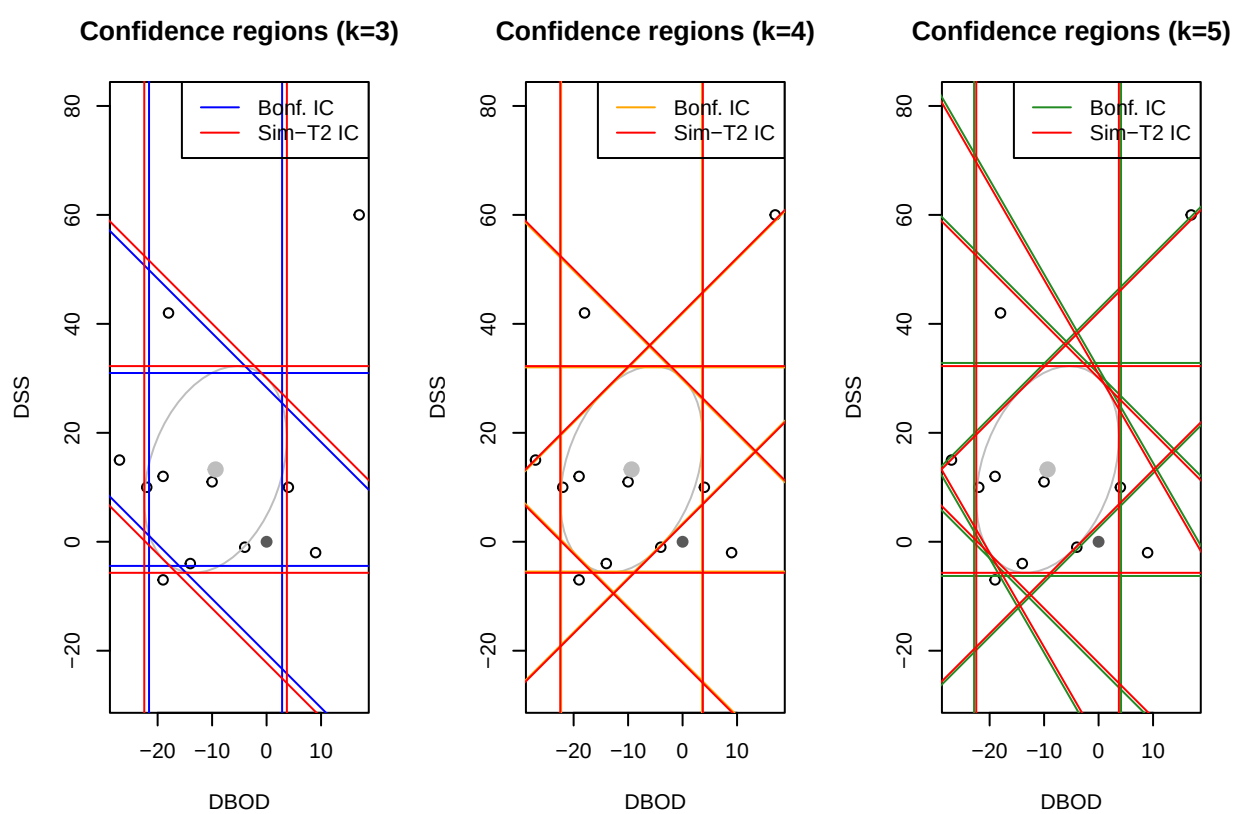


5.1.6 Comparison SCI vs BSCI

```
# Note: The simultaneous confidence intervals and the Bonferroni intervals
# are built through different methods
# => they can lead to different conclusions (although in this particular
#    case the conclusion is the same)

# Note 2: Here we have three confidence regions:
# (1) ellipse, (2) T2 rectangle, (3) Bonferroni rectangle
# In general, univariate confidence intervals for k given directions are
# associated with polygonal regions with 2*k sides.
# In this example, for k=2 the Bonferroni rectangle is smaller than the
# T2 rectangle. For increasing values of k, the Bonferroni region will
# become larger and larger

### For instance, let's consider 3 linear combinations:
k = 3

par(mfrow=c(1,3))

# Scatter plot of DSS and DBOD
plot(D, asp=1, main='Confidence regions (k=3)')

# Adding the 95% confidence region for the true mean and delta.0
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=1, col='grey')
points(delta.0[1], delta.0[2], pch=16, col='grey35', cex=1.25)

# Let's consider the following directions (DSS, DBOD and DSS + DBOD)
theta = c(0, pi/4, pi/2)

# We draw the lines defining the Bonferroni confidence region (which is no
# longer a rectangle but a polygon)
for(i in 1:length(theta)) {
  a       = c( cos(theta[i]), sin(theta[i]))
  a.orth  = c(-sin(theta[i]), cos(theta[i]))
  lines( rbind(D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  qt(1 - alpha/(2*k), n-1)) * a + 100*a.orth,
               D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
```

```
                                qt(1 - alpha/(2*k), n-1)) * a - 100*a.orth),
         col='blue', lty=1,lwd=1)
  lines( rbind(D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    qt(1 - alpha/(2*k), n-1)) * a + 100*a.orth,
               D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                qt(1 - alpha/(2*k), n-1)) * a - 100*a.orth),
         col='blue', lty=1,lwd=1)
}

# Same for T2 simultaneous
for(i in 1:length(theta)) {
  a       = c( cos(theta[i]), sin(theta[i]))
  a.orth  = c(-sin(theta[i]), cos(theta[i]))
  lines( rbind(D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    sqrt(cfr.fisher)) * a + 100*a.orth,
               D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                sqrt(cfr.fisher)) * a - 100*a.orth),
         col='red', lwd=1, lty=1)
  lines( rbind(D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    sqrt(cfr.fisher)) * a + 100*a.orth,
               D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                sqrt(cfr.fisher)) * a - 100*a.orth),
         col='red', lwd=1, lty=1)
}
# We add the legend
legend('topright', c('Bonf. IC', 'Sim-T2 IC'), col=c('blue', 'red'), lty=1)

### let's add another linear combination
k = 4

# Same plot as before with the fourth linear combination added
plot(D, asp=1, main='Confidence regions (k=4)')
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=1, col='grey')
points(delta.0[1], delta.0[2], pch=16, col='grey35', cex=1.25)

theta   = c(theta,3*pi/4)

for(i in 1:length(theta))
{
  a       = c( cos(theta[i]), sin(theta[i]))
  a.orth  = c(-sin(theta[i]), cos(theta[i]))
  lines( rbind(D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    qt(1 - alpha/(2*k), n-1)) * a + 100*a.orth,
               D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                qt(1 - alpha/(2*k), n-1)) * a - 100*a.orth),
         col='orange', lty=1,lwd=1)
  lines( rbind(D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    qt(1 - alpha/(2*k), n-1)) * a + 100*a.orth,
               D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                qt(1 - alpha/(2*k), n-1)) * a - 100*a.orth),
         col='orange', lty=1,lwd=1)
}

for(i in 1:length(theta))
{
  a       = c( cos(theta[i]), sin(theta[i]))
  a.orth  = c(-sin(theta[i]), cos(theta[i]))
  lines( rbind(D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    sqrt(cfr.fisher)) * a + 100*a.orth,
               D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                sqrt(cfr.fisher)) * a - 100*a.orth),
         col='red', lwd=1, lty=1)
  lines( rbind(D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                    sqrt(cfr.fisher)) * a + 100*a.orth,
               D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                sqrt(cfr.fisher)) * a - 100*a.orth),
         col='red', lwd=1, lty=1)
}
legend('topright', c('Bonf. IC', 'Sim-T2 IC'), col=c('orange', 'red'), lty=1)

### let's add another linear combination
k = 5

# Same plot as before with the fifth linear combination added
```

```r
plot(D, asp=1, main='Confidence regions (k=5)')
ellipse(center=D.mean, shape=D.cov/n, radius=sqrt(cfr.fisher), lwd=1, col='grey')
points(delta.0[1], delta.0[2], pch=16, col='grey35', cex=1.25)

theta   = c(theta,pi/6)

for(i in 1:length(theta))
{
  a       = c( cos(theta[i]), sin(theta[i]))
  a.orth  = c(-sin(theta[i]), cos(theta[i]))
  lines( rbind(D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  qt(1 - alpha/(2*k), n-1)) * a + 100*a.orth,
               D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  qt(1 - alpha/(2*k), n-1)) * a - 100*a.orth),
         col='forestgreen', lty=1,lwd=1)
  lines( rbind(D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  qt(1 - alpha/(2*k), n-1)) * a + 100*a.orth,
               D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  qt(1 - alpha/(2*k), n-1)) * a - 100*a.orth),
         col='forestgreen', lty=1,lwd=1)
}

for(i in 1:length(theta))
{
  a       = c( cos(theta[i]), sin(theta[i]))
  a.orth  = c(-sin(theta[i]), cos(theta[i]))
  lines( rbind(D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  sqrt(cfr.fisher)) * a + 100*a.orth,
               D.mean + as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  sqrt(cfr.fisher)) * a - 100*a.orth),
         col='red', lwd=1, lty=1)
  lines( rbind(D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  sqrt(cfr.fisher)) * a + 100*a.orth,
               D.mean - as.vector(sqrt( var(as.matrix(D) %*% a) / n ) *
                                  sqrt(cfr.fisher)) * a - 100*a.orth),
         col='red', lwd=1, lty=1)
}
legend('topright', c('Bonf. IC', 'Sim-T2 IC'), col=c('forestgreen', 'red'), lty=1)
```



```r
# The Bonferroni region is now bigger than the T2 region
```

5.2 Test for repeated measures

```r
##### Test for repeated measures
#####----------------------------

### Exercise similar to Pb 3 of 14/09/2006
###-----------------------------------------
# A pharmaceutical company performed a clinical test on 50 rats to investigate
# the effect of a new drug on the blood pressure.
# The blood pressure was measured to each rat four times: just before
# giving the drug, 8, 16, 24 hours after the drug was given.
# (a) Perform a test at level 5% to prove that the drug has influence on
#     the blood pressure during the 24 hours
# (b) Highlight the effect of the drug on the blood pressure

pressure = read.table('Lab05/pressure.txt',
                      col.names = c('h.0', 'h.8', 'h.16', 'h.24'))
head(pressure)
```

```
##          h.0      h.8       h.16      h.24
## 1  3.4588021 7.224847  0.4596315 3.4849473
## 2  2.1210665 5.592171  1.5727843 3.0106627
## 3  2.5967636 4.728428  0.4975863 2.8403774
## 4  3.2415174 5.746075  1.2622631 3.6995064
## 5 -0.2007665 4.952937 -2.5957236 0.4019944
## 6  1.8091951 6.274101  1.0664080 4.0307254
```

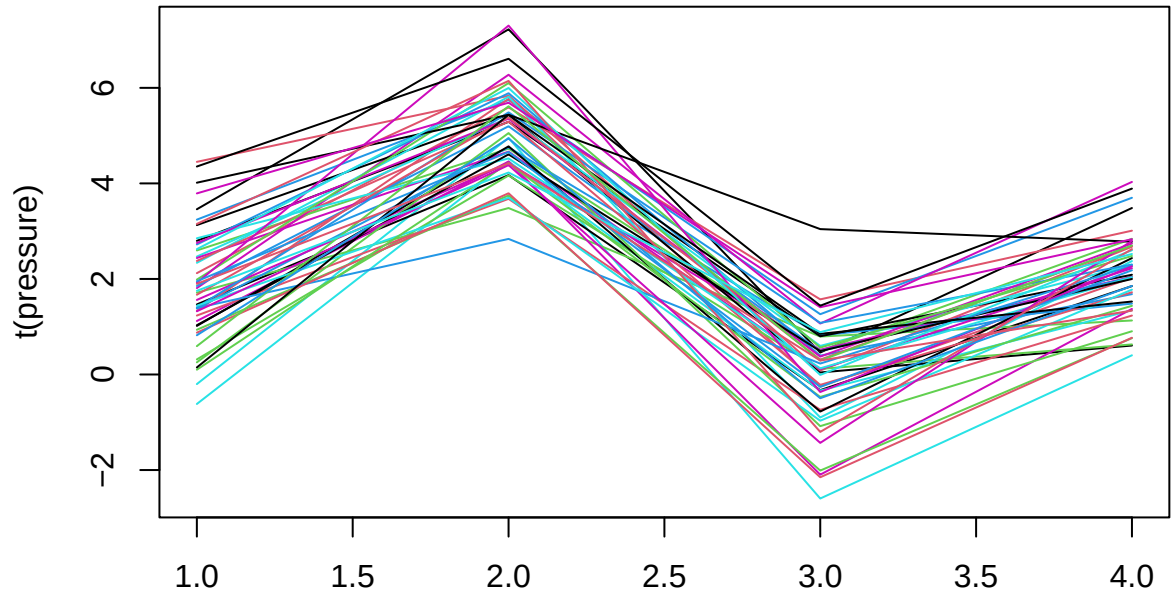
```r
dim(pressure)
```

```
## [1] 50  4
```

```r
# Test for multivariate normality (Henze-Zirkler test by default)
mvn(pressure)$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.5991505 0.9543273 YES
```

```r
# Plotting each observation as a line with 'h.0', 'h.8', 'h.16', 'h.24' on x-axis
matplot(t(pressure), type='l', lty = 1)
```



5.2.1 Qa

```
# (a) Perform a test at level 5% to prove that the drug has influence on
#     the blood pressure during the 24 hours

n = dim(pressure)[1]
q = dim(pressure)[2]

M = sapply(pressure, mean) # sample mean
M
```

```
##        h.0        h.8       h.16       h.24
## 1.86322042 5.06637474 0.04654578 2.09619753
```

```
S = cov(pressure) # covariance matrix
S
```

```
##            h.0       h.8      h.16      h.24
## h.0  1.3223812 0.5260806 0.6070589 0.5725473
## h.8  0.5260806 0.8545744 0.3124937 0.4342793
## h.16 0.6070589 0.3124937 1.1009830 0.4836133
## h.24 0.5725473 0.4342793 0.4836133 0.6995795
```

```
# we build one of the possible contrast matrices to answer
# the question
C = matrix(c(-1, 1, 0, 0,
             -1, 0, 1, 0,
             -1, 0, 0, 1), 3, 4, byrow=T)
C
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -1    1    0    0
## [2,]   -1    0    1    0
## [3,]   -1    0    0    1
```

```r
# here we are looking at the effects on the pressure
# after 8, 16 and 24 hours from the instant the drug was given

# Test: H0: C %*% mu == c(0, 0, 0) vs H1: C %*% mu != c(0, 0, 0)
alpha   = .05
delta.0 = c(0, 0, 0)

Md = C %*% M # Sample mean of the "constrated" observations
Sd = C %*% S %*% t(C) # Sample covariance of the constrasted observations
Sdinv = solve(Sd)

# Hotelling T2 statistics
T2 = n * t(Md - delta.0) %*% Sdinv %*% (Md - delta.0)

# (q-1)*(n-1)/(n-(q-1)) times the 1-alpha Fisher quantile with q-1 and n-q+1 df
cfr.fisher = ((q-1)*(n-1)/(n-(q-1)))*qf(1-alpha, (q-1), n-(q-1))

T2 < cfr.fisher # Testing if we are in the rejection region
```

```
##       [,1]
## [1,] FALSE
```

```r
T2
```

```
##         [,1]
## [1,] 1029.63
```

```r
cfr.fisher
```

```
## [1] 8.764813
```

```r
# T2 is much higher than cfr.fisher => the p-value will be very small
P = 1 - pf(T2*(n-(q-1))/((q-1)*(n-1)), (q-1), n-(q-1))
P
```

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

5.2.2 Qb

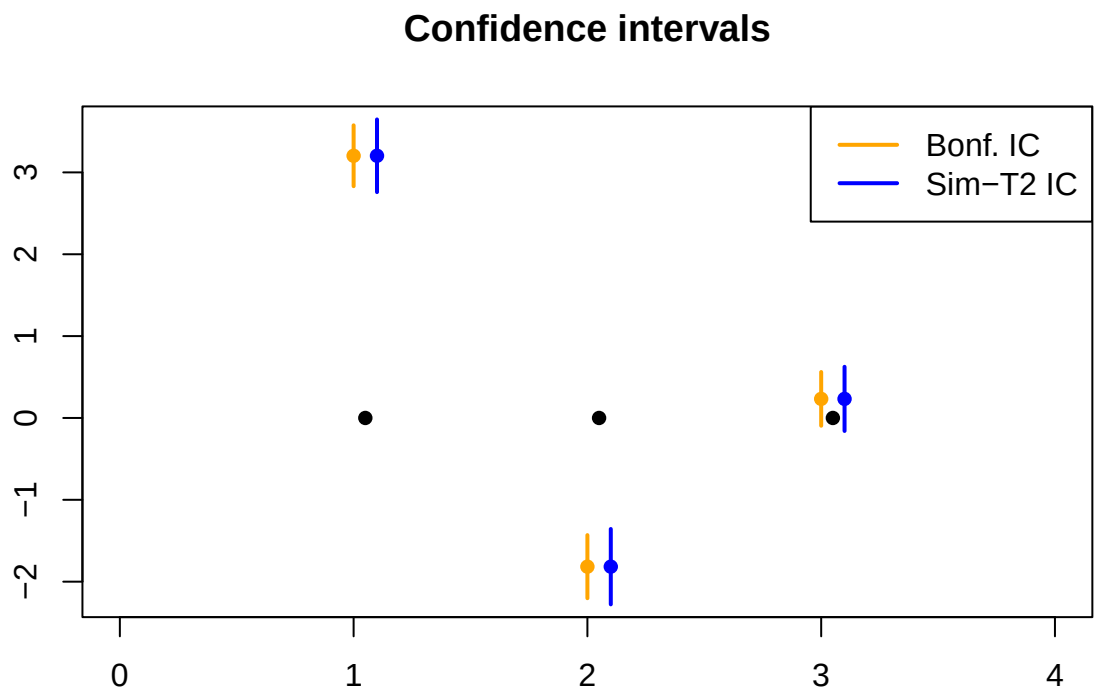
```r
# (b) Highlight the effect of the drug on the blood pressure

# It is implicitly asking for confidence intervals on the components
# (for the mean of the increments after 8 hours, 16 hours and 24 hours)

# Simultaneous T2 intervals in the direction of the contrasts h.* - h.0
IC.T2 = cbind(Md - sqrt(cfr.fisher*diag(Sd)/n),
              Md,
              Md + sqrt(cfr.fisher*diag(Sd)/n))
IC.T2
```

```
##            [,1]       [,2]       [,3]
## [1,]  2.7591135  3.2031543  3.6471952
## [2,] -2.2770835 -1.8166746 -1.3562657
## [3,] -0.1590833  0.2329771  0.6250375
```

```r
# Bonferroni intervals
k = q-1   # number of increments (ie, dim(C)[1])
cfr.t = qt(1-alpha/(2*k), n-1)

IC.BF = cbind(Md - cfr.t*sqrt(diag(Sd)/n),
              Md,
              Md + cfr.t*sqrt(diag(Sd)/n))
IC.BF
```

```
##             [,1]       [,2]       [,3]
## [1,]  2.83134533  3.2031543  3.5749633
## [2,] -2.20218911 -1.8166746 -1.4311602
## [3,] -0.09530707  0.2329771  0.5612613
```

```r
matplot(t(matrix(1:3, 3, 3)), t(IC.BF), type='b', pch='', xlim=c(0,4), xlab='',
        ylab='', main='Confidence intervals')

# Plotting the Bonferroni intervals
segments(matrix(1:3, 3, 1), IC.BF[,1], matrix(1:3, 3, 1), IC.BF[,3],
         col='orange', lwd=2)
points(1:3, IC.BF[,2], col='orange', pch=16)

# Plotting delta.0 under H0 (delta.0 == c(0, 0, 0))
points(1:3+.05, delta.0, col='black', pch=16)

# Plotting the simultaneous T2
segments(matrix(1:3+.1,3,1),IC.T2[,1],matrix(1:3+.1,3,1),IC.T2[,3], col='blue', lwd=2)
points(1:3+.1,IC.T2[,2], col='blue', pch=16)

legend('topright', c('Bonf. IC', 'Sim-T2 IC'), col=c('orange', 'blue'), lty=1, lwd=2)
```




```r
### what happens if we change the constrast matrix?
Cbis = matrix(c(-1, 1, 0, 0,
                0, -1, 1, 0,
                0, 0, -1, 1), 3, 4, byrow=T)
Cbis
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -1    1    0    0
## [2,]    0   -1    1    0
## [3,]    0    0   -1    1
```

```r
# in this way we are looking at the mean increment of the pressure every 8 hours

Mdbis = Cbis %*% M
Sdbis = Cbis %*% S %*% t(Cbis)
Sdinvbis = solve(Sdbis)

T2bis = n * t(Mdbis) %*% Sdinvbis %*% Mdbis

T2bis < cfr.fisher
```

```
##       [,1]
## [1,] FALSE
```

```
# compare the T2 test statistics associated with C and Cbis
T2bis
```

```
##         [,1]
## [1,] 1029.63
```

```
T2
```

```
##         [,1]
## [1,] 1029.63
```

```
# What is changed?
# The confidence intervals on the contrasts
# (because we are looking at different contrasts!)

# Bonferroni intervals for Cbis
IC.BFbis = cbind(Mdbis - cfr.t*sqrt(diag(Sdbis)/n),
                 Mdbis,
                 Mdbis + cfr.t*sqrt(diag(Sdbis)/n))

# Sim. T2 intervals for Cbis
IC.T2bis = cbind(Mdbis - sqrt(cfr.fisher*diag(Sdbis)/n),
                 Mdbis,
                 Mdbis + sqrt(cfr.fisher*diag(Sdbis)/n))

IC.BFbis
```

```
##           [,1]      [,2]      [,3]
## [1,]  2.831345  3.203154  3.574963
## [2,] -5.424221 -5.019829 -4.615437
## [3,]  1.729620  2.049652  2.369684
```

```
IC.BF
```

```
##             [,1]       [,2]       [,3]
## [1,]  2.83134533  3.2031543  3.5749633
## [2,] -2.20218911 -1.8166746 -1.4311602
## [3,] -0.09530707  0.2329771  0.5612613
```

```
IC.T2bis
```

```
##           [,1]      [,2]      [,3]
## [1,]  2.759113  3.203154  3.647195
## [2,] -5.502782 -5.019829 -4.536876
## [3,]  1.667447  2.049652  2.431857
```

```
IC.T2
```

```
##            [,1]       [,2]       [,3]
## [1,]  2.7591135  3.2031543  3.6471952
## [2,] -2.2770835 -1.8166746 -1.3562657
## [3,] -0.1590833  0.2329771  0.6250375
```

5.2.3 Different test

```
### what if we want to verify the following hypothesis:
### "the drug decreases the pressure of two units with respect to
### the baseline at both 8 and 16 hours, and its effect vanishes in 24 hours
### from the drug administration"

C = matrix(c(-1, 1, 0, 0,
             -1, 0, 1, 0,
             -1, 0, 0, 1), 3, 4, byrow=T)

delta.0 = c(-2, -2, 0)

# or
```

```r
C = matrix(c(-1, 1, 0, 0,
             0, -1, 1, 0,
             0, 0, -1, 1), 3, 4, byrow=T)

delta.0 = c(-2, 0, 2)

Md = C %*% M # sample mean for the contrasted observations
Sd = C %*% S %*% t(C) # sample covariance matrix
Sdinv = solve(Sd)

# Hotelling T2 statistic
T2 = n * t(Md - delta.0) %*% Sdinv %*% (Md - delta.0)

# (q-1)*(n-1)/(n-q+1) times the 1-alpha Fisher quantile with q-1 and n-q+1 df
cfr.fisher = ((q-1)*(n-1)/(n-(q-1)))*qf(1-alpha,(q-1),n-(q-1))

T2 < cfr.fisher # Do we accept H0?
```

```
##       [,1]
## [1,] FALSE
```

```r
T2
```

```
##          [,1]
## [1,] 2010.284
```

```r
cfr.fisher
```

```
## [1] 8.764813
```

```r
# p-value
P = 1 - pf(T2*(n-(q-1))/((q-1)*(n-1)), (q-1), n-(q-1))
P
```

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

5.3 Test for two indep Gaussians

```r
##### Test for the mean of two independent Gaussian populations
#####-------------------------------------------------------------

# we build the data
t1 = matrix(c(3,3,1,6,2,3),2)
t1 = data.frame(t(t1))
t2 = matrix(c(2,3,5,1,3,1,2,3),2)
t2 = data.frame(t(t2))

t1
```

```
##   X1 X2
## 1  3  3
## 2  1  6
## 3  2  3
```

```r
t2
```

```
##   X1 X2
## 1  2  3
## 2  5  1
## 3  3  1
## 4  2  3
```

```r
n1 = dim(t1)[1] # n1=3
n2 = dim(t2)[1] # n2=4
p  = dim(t1)[2] # p=2

# we compute the sample mean, covariance matrices and the matrix Spooled
```

```r
t1.mean = sapply(t1,mean)
t2.mean = sapply(t2,mean)
t1.cov  =  cov(t1)
t2.cov  =  cov(t2)
Sp      = ((n1-1)*t1.cov + (n2-1)*t2.cov)/(n1+n2-2)
# we compare the matrices
list(S1=t1.cov, S2=t2.cov, Spooled=Sp)
```

```
## $S1
##      X1   X2
## X1  1.0 -1.5
## X2 -1.5  3.0
##
## $S2
##           X1        X2
## X1  2.000000 -1.333333
## X2 -1.333333  1.333333
##
## $Spooled
##      X1   X2
## X1  1.6 -1.4
## X2 -1.4  2.0
```

```r
# Test H0: mu1 == mu2  vs  H1: mu1 != mu2
# ie,
# Test H0: mu1-mu2 == c(0,0)  vs  H1: mu1-mu2 != c(0,0)

alpha   = .01
delta.0 = c(0,0)
Spinv   = solve(Sp)

T2 = n1*n2/(n1+n2) * (t1.mean-t2.mean-delta.0) %*% Spinv %*% (t1.mean-t2.mean-delta.0)

cfr.fisher = (p*(n1+n2-2)/(n1+n2-1-p))*qf(1-alpha,p,n1+n2-1-p)
T2 < cfr.fisher # TRUE: no statistical evidence to reject H0 at level 1%
```

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

```r
P = 1 - pf(T2/(p*(n1+n2-2)/(n1+n2-1-p)), p, n1+n2-1-p)
P
```

```
##          [,1]
## [1,] 0.317686
```

```r
# P-value high (we don't reject at 1%,5%,10%)

# Simultaneous T2 intervals
IC.T2.X1 = c(t1.mean[1]-t2.mean[1]-sqrt(cfr.fisher*Sp[1,1]*(1/n1+1/n2)),
             t1.mean[1]-t2.mean[1]+sqrt(cfr.fisher*Sp[1,1]*(1/n1+1/n2)) )
IC.T2.X2 = c(t1.mean[2]-t2.mean[2]-sqrt(cfr.fisher*Sp[2,2]*(1/n1+1/n2)),
             t1.mean[2]-t2.mean[2]+sqrt(cfr.fisher*Sp[2,2]*(1/n1+1/n2)) )
IC.T2 = rbind(IC.T2.X1, IC.T2.X2)
dimnames(IC.T2)[[2]] = c('inf','sup')
IC.T2
```

```
##                inf      sup
## IC.T2.X1 -7.480741 5.480741
## IC.T2.X2 -5.245688 9.245688
```

5.4 Exam exercise

```r
##### Pb 2 of 4/07/2013
#####--------------------
# The Chinese Institute of Genomics has reconstructed the genetic map of 200
# Chinese individuals, among which 100 were allergic to almonds and 100 were not
# allergic to almonds. The files hatingalmonds.txt e lovingalmonds.txt collect
# the presence (1) or absence (0) of 520 mutations, for the two groups respectively.
# The geneticist suspects that some of these mutations might be positively associated
# with the allergy to almonds, and for this reason they decide to compare the
```

```r
# incidence of the 520 mutations in the two populations.
# a) Report the significant mutations, imposing a probability of at most 1%
#    that the single mutation is judged as influential if it isn't.
# b) Report the significant mutations, imposing a probability of at most 1%
#    that at least one of the non-influential mutations is judged as influential.
# c) Report the significant mutations, imposing that the expected proportion
#    of mutations erroneously judged to be influential among all those judged to be
#    influential is lower than or equal to 1%

# You'll have to correct the univariate p-values of multiple tests.
# R function to correct p-values: p.adjust

allergy = read.table('Lab05/hatingalmonds.txt')
head(allergy)
```

```
##    M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 M16 M17 M18 M19 M20 M21
## P1  0  0  0  0  1  1  0  1  0   0   0   1   1   0   0   0   1   1   1   1   1
## P2  0  0  0  0  0  1  0  0  0   0   0   1   1   0   1   1   0   0   1   1   0
## P3  0  1  0  0  0  0  1  0  0   0   1   0   0   0   0   1   0   0   0   0   0
## P4  0  0  0  0  0  0  1  1  0   0   1   0   1   1   0   1   0   1   0   0   1
## P5  1  0  1  0  0  0  0  0  0   0   0   0   1   1   0   0   0   1   0   0   0
## P6  1  0  0  0  0  0  0  0  0   1   0   0   0   0   0   1   1   1   1   0   0
##    M22 M23 M24 M25 M26 M27 M28 M29 M30 M31 M32 M33 M34 M35 M36 M37 M38 M39 M40
## P1   1   0   1   1   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0
## P2   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   1   0
## P3   0   0   0   0   0   1   1   0   0   0   0   0   0   1   1   0   0   0   0
## P4   0   0   1   0   1   0   1   0   0   0   0   0   0   0   0   1   0   0   0
## P5   0   0   0   0   0   0   0   0   0   0   1   1   0   1   0   0   0   0   1
## P6   1   1   0   0   0   0   1   0   0   1   0   0   1   0   0   0   0   0   0
##    M41 M42 M43 M44 M45 M46 M47 M48 M49 M50 M51 M52 M53 M54 M55 M56 M57 M58 M59
## P1   0   0   1   1   1   0   0   0   1   0   0   0   0   0   0   0   0   0   1
## P2   0   0   0   0   0   0   0   0   0   1   0   0   1   0   1   0   0   0   1
## P3   0   0   0   0   0   1   0   0   0   0   0   0   0   0   1   1   1   0   1
## P4   0   0   1   0   1   0   0   1   0   0   0   0   1   1   0   1   0   0   1
## P5   0   1   0   1   1   0   0   0   1   0   0   0   1   0   0   1   1   0   0
## P6   1   1   0   1   1   0   0   0   0   1   1   0   0   0   0   0   0   0   0
##    M60 M61 M62 M63 M64 M65 M66 M67 M68 M69 M70 M71 M72 M73 M74 M75 M76 M77 M78
## P1   0   0   1   0   1   0   0   0   0   0   0   0   0   0   1   0   1   0   0
## P2   0   0   0   1   0   1   0   1   0   0   0   0   0   0   1   0   0   0   0
## P3   0   0   1   0   1   0   0   1   1   0   1   0   0   0   0   0   0   0   0
## P4   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   1   0
## P5   0   0   0   0   0   1   0   0   0   0   0   0   0   0   1   0   0   0   1
## P6   1   0   0   0   0   0   1   0   0   0   0   0   0   0   1   0   1   1   0
##    M79 M80 M81 M82 M83 M84 M85 M86 M87 M88 M89 M90 M91 M92 M93 M94 M95 M96 M97
## P1   0   0   0   0   1   1   0   0   1   0   1   1   1   0   0   1   0   0   1
## P2   0   1   1   0   0   1   0   0   0   0   0   0   0   0   0   0   0   1   1
## P3   0   0   0   0   0   0   1   0   0   1   0   0   0   0   0   1   0   0   0
## P4   1   0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   1   0   0
## P5   1   0   0   0   1   0   0   0   1   1   0   1   0   1   0   1   0   0   0
## P6   0   1   0   0   1   1   0   1   1   0   0   0   1   1   0   1   0   0   0
##    M98 M99 M100 M101 M102 M103 M104 M105 M106 M107 M108 M109 M110 M111 M112
## P1   0   0    0    0    0    0    1    1    1    0    0    0    0    1    0
## P2   0   1    1    0    0    0    0    0    0    0    0    0    1    1    0
## P3   0   0    1    1    0    0    1    0    0    0    0    1    1    1    0
## P4   0   0    0    0    1    0    1    0    1    0    0    0    0    0    1
## P5   0   0    0    0    0    0    0    1    0    0    0    0    0    1    0
## P6   0   0    0    0    0    0    1    0    0    0    0    0    1    0    0
##    M113 M114 M115 M116 M117 M118 M119 M120 M121 M122 M123 M124 M125 M126 M127
## P1    0    0    0    0    0    1    0    0    1    0    0    0    0    0    0
## P2    0    0    1    0    0    0    0    1    1    0    0    1    1    0    0
## P3    0    0    0    1    0    0    0    1    0    1    0    0    0    0    0
## P4    0    0    0    1    1    0    0    0    0    0    0    0    1    0    0
## P5    0    1    0    0    0    0    0    1    1    0    0    0    0    0    0
## P6    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
##    M128 M129 M130 M131 M132 M133 M134 M135 M136 M137 M138 M139 M140 M141 M142
## P1    1    0    1    1    0    1    0    1    0    0    0    0    0    0    1
## P2    0    0    0    0    0    0    0    1    0    0    0    0    1    1    0
## P3    0    0    1    0    0    0    1    0    1    0    0    0    0    0    0
## P4    1    0    0    1    0    0    0    1    0    0    1    0    0    1    0
## P5    1    0    0    0    1    0    0    0    0    0    0    0    0    1    0
## P6    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
##    M143 M144 M145 M146 M147 M148 M149 M150 M151 M152 M153 M154 M155 M156 M157
## P1    0    0    0    0    0    1    1    0    0    0    1    0    0    0    0
## P2    0    0    0    0    0    0    0    0    0    1    1    0    0    0    1
```

```
## P3    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0
## P4    0    0    0    0    0    0    0    0    1    0    0    1    0    0    1
## P5    0    1    0    0    0    0    1    1    1    0    1    0    1    0    0
## P6    0    0    0    0    1    0    0    0    1    0    0    0    0    0    0
##    M158 M159 M160 M161 M162 M163 M164 M165 M166 M167 M168 M169 M170 M171 M172
## P1    0    0    0    0    0    0    0    1    0    1    0    0    0    0    0
## P2    0    1    0    0    1    0    0    0    0    0    0    0    0    0    0
## P3    1    0    0    0    0    1    1    0    0    0    0    0    0    0    0
## P4    0    0    1    1    0    0    0    0    1    0    1    0    0    0    0
## P5    0    0    1    0    0    1    0    0    0    0    0    0    0    0    0
## P6    0    0    1    0    0    1    0    0    0    0    1    0    0    0    0
##    M173 M174 M175 M176 M177 M178 M179 M180 M181 M182 M183 M184 M185 M186 M187
## P1    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P2    1    1    0    0    0    0    1    0    0    0    0    0    0    1    0
## P3    0    1    0    1    0    0    0    0    1    1    0    0    0    1    1
## P4    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
## P5    0    0    0    1    0    0    0    0    0    1    0    1    1    0    0
## P6    1    0    0    0    0    1    0    0    1    0    0    0    0    0    0
##    M188 M189 M190 M191 M192 M193 M194 M195 M196 M197 M198 M199 M200 M201 M202
## P1    1    0    1    0    0    0    0    0    0    0    0    0    1    1    1
## P2    0    1    0    1    0    0    1    0    0    0    0    0    0    0    1
## P3    0    0    0    0    0    0    0    1    0    1    1    0    1    0    0
## P4    0    0    0    0    0    0    1    0    1    0    1    0    0    1    0
## P5    1    0    1    0    0    1    0    0    0    0    0    0    0    0    0
## P6    1    0    0    0    0    0    0    0    0    0    0    0    0    1    0
##    M203 M204 M205 M206 M207 M208 M209 M210 M211 M212 M213 M214 M215 M216 M217
## P1    0    1    0    1    0    0    0    1    0    0    1    1    0    0    0
## P2    0    0    0    0    1    1    0    0    0    1    1    0    0    1    1
## P3    0    0    0    0    0    0    0    0    1    0    1    1    0    0    0
## P4    0    0    0    0    0    1    1    0    0    0    1    0    0    1    0
## P5    0    0    0    0    0    1    0    1    0    1    1    0    1    1    0
## P6    1    0    0    0    0    0    1    0    0    0    0    1    0    0    1
##    M218 M219 M220 M221 M222 M223 M224 M225 M226 M227 M228 M229 M230 M231 M232
## P1    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
## P2    1    0    1    0    0    0    1    0    0    0    0    0    0    0    0
## P3    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
## P4    0    1    0    0    0    0    1    0    0    0    0    0    0    0    0
## P5    0    0    0    0    0    0    0    0    1    1    0    0    0    1    0
## P6    0    1    0    0    0    1    0    1    1    1    0    0    0    1    0
##    M233 M234 M235 M236 M237 M238 M239 M240 M241 M242 M243 M244 M245 M246 M247
## P1    0    0    0    1    0    0    0    0    1    0    0    0    0    0    1
## P2    1    1    1    0    0    0    0    0    1    0    0    0    1    1    0
## P3    0    0    1    1    1    0    0    0    0    0    0    0    0    1    1
## P4    1    0    1    1    0    0    0    0    0    0    0    0    0    1    0
## P5    1    0    0    0    0    0    0    0    0    0    1    0    0    0    0
## P6    0    0    0    0    1    0    0    0    0    1    0    0    0    1    1
##    M248 M249 M250 M251 M252 M253 M254 M255 M256 M257 M258 M259 M260 M261 M262
## P1    1    0    0    1    1    0    0    0    0    0    0    0    1    0    1
## P2    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
## P3    1    0    0    0    0    1    0    0    1    0    0    0    0    0    1
## P4    1    0    0    0    1    0    0    0    1    0    0    0    0    1    0
## P5    0    0    0    1    1    0    0    1    0    1    0    0    1    0    1
## P6    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
##    M263 M264 M265 M266 M267 M268 M269 M270 M271 M272 M273 M274 M275 M276 M277
## P1    0    0    0    0    1    0    0    1    0    0    0    0    1    0    0
## P2    0    0    0    1    0    1    0    1    0    1    1    0    0    0    0
## P3    0    0    0    1    0    1    1    1    0    1    0    0    0    0    1
## P4    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
## P5    0    0    1    1    1    0    1    0    0    1    1    0    0    0    0
## P6    0    1    0    0    1    0    0    1    0    1    1    0    0    0    0
##    M278 M279 M280 M281 M282 M283 M284 M285 M286 M287 M288 M289 M290 M291 M292
## P1    0    0    0    0    0    0    1    1    0    0    0    0    1    1    0
## P2    0    1    0    0    0    0    1    1    0    0    0    0    0    0    1
## P3    0    0    1    1    0    0    1    0    0    0    1    0    0    1    0
## P4    1    0    1    0    0    0    0    0    0    0    0    0    0    1    0
## P5    0    0    0    0    0    0    0    0    1    0    0    0    1    0    0
## P6    0    1    1    0    1    0    1    0    0    0    0    1    0    1    0
##    M293 M294 M295 M296 M297 M298 M299 M300 M301 M302 M303 M304 M305 M306 M307
## P1    0    0    0    0    1    0    0    0    0    1    0    0    0    0    1
## P2    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
## P3    0    0    0    1    1    0    0    1    0    1    0    1    0    0    1
## P4    0    0    0    0    0    0    0    0    0    0    0    0    0    1    1
## P5    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P6    0    0    0    0    0    0    0    1    1    0    0    0    1    1    0
##    M308 M309 M310 M311 M312 M313 M314 M315 M316 M317 M318 M319 M320 M321 M322
```

```
## P1    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
## P2    1    0    0    0    1    1    0    0    0    0    0    0    0    0    0
## P3    0    1    1    0    0    0    1    0    0    1    0    0    0    1    1
## P4    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P5    0    0    0    1    0    0    0    0    1    0    0    0    1    0    0
## P6    0    0    0    1    0    1    0    0    0    0    0    0    0    0    0
##    M323 M324 M325 M326 M327 M328 M329 M330 M331 M332 M333 M334 M335 M336 M337
## P1    0    1    0    0    0    1    0    1    0    1    0    0    1    1    1
## P2    0    0    0    0    1    0    0    0    0    1    0    0    0    0    0
## P3    1    0    0    0    0    1    1    0    0    0    1    0    0    0    0
## P4    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
## P5    1    1    0    0    0    0    0    1    0    1    1    1    0    0    0
## P6    0    0    0    0    0    0    1    1    0    0    0    0    1    0    0
##    M338 M339 M340 M341 M342 M343 M344 M345 M346 M347 M348 M349 M350 M351 M352
## P1    0    0    0    0    0    0    0    0    0    0    0    0    0    1    1
## P2    0    0    0    0    1    0    1    0    0    0    0    0    0    0    1
## P3    0    0    0    0    0    0    0    1    1    1    1    0    0    0    0
## P4    1    1    0    0    0    1    0    1    0    0    0    0    1    1    0
## P5    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P6    0    0    1    0    0    0    0    0    0    0    0    1    1    0    0
##    M353 M354 M355 M356 M357 M358 M359 M360 M361 M362 M363 M364 M365 M366 M367
## P1    0    0    0    0    1    1    0    0    1    0    1    0    0    1    0
## P2    0    0    1    0    0    0    1    0    1    0    0    0    0    0    0
## P3    0    1    0    0    0    1    0    0    0    0    0    0    0    1    0
## P4    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P5    0    0    1    0    1    0    0    0    0    1    0    0    0    0    0
## P6    0    0    0    0    0    1    1    0    1    0    1    0    0    0    1
##    M368 M369 M370 M371 M372 M373 M374 M375 M376 M377 M378 M379 M380 M381 M382
## P1    0    1    0    0    1    0    0    0    1    0    0    0    0    0    0
## P2    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
## P3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
## P4    1    0    0    0    1    0    1    0    0    0    0    0    0    0    0
## P5    0    0    0    0    1    1    0    0    0    0    0    0    0    1    0
## P6    0    0    0    0    1    0    0    0    1    0    0    0    0    1    0
##    M383 M384 M385 M386 M387 M388 M389 M390 M391 M392 M393 M394 M395 M396 M397
## P1    0    0    0    0    1    1    0    1    0    0    1    0    1    0    0
## P2    0    0    0    1    1    0    1    1    1    0    0    0    1    1    0
## P3    0    0    0    1    0    1    0    0    0    0    0    0    0    0    0
## P4    0    0    0    1    1    0    0    0    0    0    0    0    1    0    0
## P5    1    0    0    0    0    0    0    0    0    0    0    0    1    0    0
## P6    0    1    0    0    0    0    0    0    0    0    1    1    1    1    0
##    M398 M399 M400 M401 M402 M403 M404 M405 M406 M407 M408 M409 M410 M411 M412
## P1    1    0    0    0    0    0    1    0    0    0    0    0    0    0    0
## P2    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0
## P3    0    0    1    0    0    1    0    0    0    0    0    1    0    1    0
## P4    1    0    1    1    0    0    1    1    0    0    0    0    0    1    0
## P5    0    0    1    0    1    1    0    1    0    0    0    0    0    1    1
## P6    0    1    0    0    0    0    0    0    0    0    0    1    0    1    0
##    M413 M414 M415 M416 M417 M418 M419 M420 M421 M422 M423 M424 M425 M426 M427
## P1    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
## P2    0    0    0    0    0    0    0    0    1    1    0    0    1    0    0
## P3    0    0    0    1    0    0    1    1    0    0    1    0    0    0    0
## P4    0    1    0    0    0    1    1    0    0    0    1    0    1    0    0
## P5    0    0    0    0    1    0    1    1    0    0    0    0    0    0    0
## P6    0    0    0    1    1    0    1    0    0    1    0    0    0    1    0
##    M428 M429 M430 M431 M432 M433 M434 M435 M436 M437 M438 M439 M440 M441 M442
## P1    0    0    0    0    0    0    0    0    0    0    1    0    1    1    1
## P2    0    0    0    0    0    0    0    1    0    0    0    0    1    0    1
## P3    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
## P4    1    0    0    1    1    1    0    0    1    0    0    0    0    1    1
## P5    1    1    0    0    0    1    0    0    0    0    0    0    0    0    0
## P6    0    0    0    0    1    0    0    1    0    1    0    0    0    0    0
##    M443 M444 M445 M446 M447 M448 M449 M450 M451 M452 M453 M454 M455 M456 M457
## P1    1    1    1    1    1    0    0    1    0    0    0    0    1    0    0
## P2    0    1    0    1    0    0    0    0    1    0    1    0    1    0    0
## P3    1    0    1    0    0    0    0    0    0    1    1    0    1    1    0
## P4    1    1    1    1    1    0    0    1    1    0    1    0    0    0    0
## P5    1    0    1    1    0    1    1    1    0    1    0    0    1    0    0
## P6    0    0    0    1    0    1    0    1    0    0    0    0    0    0    0
##    M458 M459 M460 M461 M462 M463 M464 M465 M466 M467 M468 M469 M470 M471 M472
## P1    0    1    0    0    0    0    1    0    0    1    1    0    1    0    0
## P2    1    0    0    0    0    1    0    0    0    0    0    0    1    0    0
## P3    1    0    0    0    0    1    0    0    0    0    0    0    1    0    0
## P4    0    0    0    1    0    1    0    0    1    1    1    0    0    1    0
## P5    0    0    0    0    1    0    0    0    0    0    0    0    0    1    1
```

```
## P6    0    1    0    0    0    0    0    0    0    0    1    0    0    1    1
##    M473 M474 M475 M476 M477 M478 M479 M480 M481 M482 M483 M484 M485 M486 M487
## P1    0    0    1    0    1    0    0    0    1    0    1    0    0    0    0
## P2    0    0    1    0    0    0    0    0    0    0    1    1    1    1    1
## P3    0    1    0    0    0    1    1    0    0    0    0    0    0    1    0
## P4    0    0    0    0    0    1    0    1    1    1    0    0    0    0    0
## P5    0    0    0    1    0    0    0    0    0    1    0    1    0    0    0
## P6    0    1    0    0    0    1    0    0    0    0    0    0    1    1    0
##    M488 M489 M490 M491 M492 M493 M494 M495 M496 M497 M498 M499 M500 M501 M502
## P1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P2    0    0    0    0    0    1    1    1    0    0    0    0    0    0    0
## P3    0    0    0    0    0    1    0    0    0    0    0    1    0    1    0
## P4    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P5    1    0    0    1    0    0    0    0    1    0    1    1    1    1    0
## P6    1    0    0    1    0    0    0    1    0    0    0    1    0    0    0
##    M503 M504 M505 M506 M507 M508 M509 M510 M511 M512 M513 M514 M515 M516 M517
## P1    0    1    0    1    0    1    1    0    0    0    0    1    0    0    0
## P2    0    1    0    0    0    0    0    0    0    0    0    0    0    1    0
## P3    0    0    1    0    0    0    0    0    0    0    0    1    1    0    1
## P4    0    0    1    0    0    0    0    0    1    1    1    0    0    0    0
## P5    1    0    0    0    0    0    0    0    1    0    1    0    0    1    1
## P6    1    1    1    0    0    1    1    1    0    0    1    0    0    0    0
##    M518 M519 M520
## P1    1    0    1
## P2    0    0    0
## P3    0    0    1
## P4    0    0    0
## P5    0    1    0
## P6    1    0    0
```

```
dim(allergy)
```

```
## [1] 100 520
```

```
noallergy = read.table('Lab05/lovingalmonds.txt')
head(noallergy)
```

```
##      M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 M16 M17 M18 M19 M20 M21
## P101  1  0  0  1  1  0  0  1  0   1   1   1   1   0   0   1   1   1   0   0   0
## P102  0  0  0  0  0  0  1  0  0   1   1   1   0   0   0   0   0   0   0   0   0
## P103  0  0  0  0  0  1  0  0  0   0   1   0   0   0   1   0   0   0   0   0   0
## P104  1  1  0  0  1  0  0  0  0   0   1   1   0   1   0   0   0   0   0   0   0
## P105  0  1  1  1  1  1  0  0  0   1   0   1   0   0   0   1   1   0   0   0   1
## P106  0  0  0  0  1  0  0  0  0   1   0   0   0   1   0   0   0   0   1   0   0
##      M22 M23 M24 M25 M26 M27 M28 M29 M30 M31 M32 M33 M34 M35 M36 M37 M38 M39
## P101   0   0   0   1   0   0   1   0   0   0   1   1   0   0   0   1   0   0
## P102   0   0   0   0   1   0   0   0   0   0   1   0   1   0   0   0   1   0
## P103   0   0   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
## P104   0   0   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0   1
## P105   0   0   0   0   1   0   0   1   0   0   0   0   0   0   0   1   0   0
## P106   1   0   1   0   0   1   1   1   0   0   0   0   1   1   0   0   0   0
##      M40 M41 M42 M43 M44 M45 M46 M47 M48 M49 M50 M51 M52 M53 M54 M55 M56 M57
## P101   0   1   0   1   1   0   0   1   1   0   0   0   0   0   0   0   1   0
## P102   1   0   1   1   0   1   0   0   0   1   1   0   1   0   1   0   0   0
## P103   0   0   0   0   0   0   0   0   1   0   0   1   0   0   0   0   1   0
## P104   0   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0   0   0
## P105   1   0   0   1   0   0   0   0   1   0   0   0   0   1   0   0   0   0
## P106   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
##      M58 M59 M60 M61 M62 M63 M64 M65 M66 M67 M68 M69 M70 M71 M72 M73 M74 M75
## P101   0   0   0   0   0   1   1   0   0   0   0   0   0   0   0   1   0   0
## P102   1   0   0   0   0   0   0   1   1   0   0   0   0   0   0   0   0   0
## P103   0   1   0   1   1   1   0   0   0   0   0   0   1   0   0   0   0   0
## P104   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0
## P105   0   1   0   0   0   0   0   0   1   1   0   0   0   1   0   0   0   1
## P106   0   1   0   0   0   0   0   0   1   1   0   0   1   0   0   0   0   0
##      M76 M77 M78 M79 M80 M81 M82 M83 M84 M85 M86 M87 M88 M89 M90 M91 M92 M93
## P101   0   0   1   1   0   1   0   0   1   1   0   0   1   0   0   0   0   0
## P102   0   0   0   0   1   0   0   0   1   0   1   1   0   1   0   0   0   0
## P103   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## P104   1   1   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1
## P105   1   0   0   1   0   1   0   0   1   1   0   0   1   1   1   0   0   0
## P106   0   1   0   0   0   0   0   0   1   0   0   1   1   0   0   0   0   0
##      M94 M95 M96 M97 M98 M99 M100 M101 M102 M103 M104 M105 M106 M107 M108 M109
```

```
## P101   0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   1
## P102   0   0   0   1   0   0   0   0   0   0   1   0   0   0   0   1
## P103   0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   0
## P104   0   0   0   0   1   0   0   0   0   1   1   0   0   0   0   0
## P105   1   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0
## P106   1   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0
##      M110 M111 M112 M113 M114 M115 M116 M117 M118 M119 M120 M121 M122 M123 M124
## P101    1    0    0    1    1    1    1    1    0    1    1    0    0    0    0
## P102    0    0    1    0    1    0    0    0    0    0    0    1    0    0    0
## P103    0    1    0    0    1    0    0    0    0    1    0    0    0    0    0
## P104    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P105    0    0    0    0    0    0    0    1    0    1    0    1    0    0    1
## P106    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##      M125 M126 M127 M128 M129 M130 M131 M132 M133 M134 M135 M136 M137 M138 M139
## P101    0    0    0    0    1    0    1    0    0    1    0    0    0    0    0
## P102    0    1    0    0    1    0    0    0    0    0    0    1    0    0    0
## P103    0    0    0    1    1    0    0    0    1    0    1    0    0    1    0
## P104    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
## P105    0    0    0    1    1    0    1    0    0    0    0    0    0    1    0
## P106    0    0    0    0    0    1    1    0    0    0    1    0    0    0    0
##      M140 M141 M142 M143 M144 M145 M146 M147 M148 M149 M150 M151 M152 M153 M154
## P101    1    1    1    0    0    0    1    0    0    1    0    0    0    0    0
## P102    1    0    0    0    0    1    0    0    0    0    0    0    0    0    0
## P103    0    0    0    1    0    1    0    0    1    0    1    0    1    1    0
## P104    1    0    0    1    0    0    0    0    0    0    0    0    1    1    0
## P105    1    0    0    0    0    0    0    0    0    1    1    1    0    0    0
## P106    0    0    0    1    1    0    0    0    0    0    1    1    0    0    1
##      M155 M156 M157 M158 M159 M160 M161 M162 M163 M164 M165 M166 M167 M168 M169
## P101    0    0    1    0    1    0    0    0    0    1    1    0    0    0    0
## P102    0    0    1    1    0    0    0    0    0    0    0    0    1    0    0
## P103    0    0    0    1    0    1    0    0    1    0    0    0    0    0    1
## P104    0    1    1    1    0    0    0    0    1    0    0    0    0    0    0
## P105    0    1    0    0    0    1    1    0    1    0    0    1    0    1    0
## P106    1    0    0    1    0    0    0    0    0    0    0    0    0    0    0
##      M170 M171 M172 M173 M174 M175 M176 M177 M178 M179 M180 M181 M182 M183 M184
## P101    1    0    0    0    0    0    1    0    0    0    0    1    0    0    0
## P102    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P103    0    0    0    0    0    0    0    0    0    1    0    1    0    0    0
## P104    1    0    1    0    0    0    0    1    0    0    0    0    1    0    0
## P105    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
## P106    0    0    0    0    1    0    0    0    0    1    0    0    0    0    0
##      M185 M186 M187 M188 M189 M190 M191 M192 M193 M194 M195 M196 M197 M198 M199
## P101    0    0    0    0    0    0    0    0    1    0    0    0    0    1    1
## P102    0    0    0    0    0    0    1    0    0    0    1    0    0    0    0
## P103    1    1    0    0    0    0    0    1    1    1    0    0    0    0    0
## P104    0    0    0    0    0    0    0    1    0    0    0    0    0    1    0
## P105    0    0    1    0    0    0    0    0    1    0    0    0    0    1    0
## P106    0    0    0    0    0    0    0    1    1    1    0    0    0    0    0
##      M200 M201 M202 M203 M204 M205 M206 M207 M208 M209 M210 M211 M212 M213 M214
## P101    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
## P102    0    0    0    0    0    0    0    0    0    0    1    1    0    0    0
## P103    0    0    1    0    1    0    0    0    1    0    0    1    0    0    1
## P104    0    1    0    0    0    0    1    0    0    0    0    0    0    0    0
## P105    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
## P106    1    1    0    0    0    0    1    0    0    0    0    1    1    0    0
##      M215 M216 M217 M218 M219 M220 M221 M222 M223 M224 M225 M226 M227 M228 M229
## P101    0    0    0    0    1    1    1    0    0    0    0    0    0    0    0
## P102    0    1    0    0    1    0    0    0    0    0    1    0    0    0    0
## P103    1    1    1    0    1    0    1    0    0    0    0    1    0    0    0
## P104    1    0    0    1    0    1    0    0    0    0    1    0    0    0    0
## P105    0    0    0    0    1    1    0    0    0    0    0    0    1    0    0
## P106    0    0    0    0    1    1    0    0    0    0    0    1    0    0    0
##      M230 M231 M232 M233 M234 M235 M236 M237 M238 M239 M240 M241 M242 M243 M244
## P101    0    0    1    0    1    0    0    0    0    0    0    0    1    0    0
## P102    1    0    0    0    0    0    1    0    0    0    1    0    0    0    0
## P103    0    0    0    0    1    0    1    0    0    0    0    0    1    0    0
## P104    1    0    0    1    0    0    1    0    0    0    0    0    0    0    0
## P105    0    0    0    0    1    0    1    1    0    0    1    1    1    0    0
## P106    1    0    0    1    0    1    0    0    0    1    0    0    0    0    0
##      M245 M246 M247 M248 M249 M250 M251 M252 M253 M254 M255 M256 M257 M258 M259
## P101    0    0    0    0    0    1    0    0    1    1    0    1    1    1    1
## P102    0    0    0    0    1    0    1    0    1    0    1    1    1    1    0
## P103    1    0    0    0    0    1    0    0    0    1    0    0    1    1    0
## P104    0    1    0    0    0    0    0    0    1    0    0    0    0    1    0
## P105    1    1    0    0    1    0    0    0    0    0    0    0    0    1    1
```

```
## P106    1    0    0    0    0    0    1    0    1    0    0    1    1    0    0
##      M260 M261 M262 M263 M264 M265 M266 M267 M268 M269 M270 M271 M272 M273 M274
## P101    1    0    0    0    0    0    0    0    0    1    0    0    0    0    0
## P102    1    1    0    0    1    1    0    1    1    1    0    0    1    0    0
## P103    0    0    1    0    0    0    1    1    1    0    0    0    0    1    0
## P104    0    0    1    0    0    0    0    0    0    0    0    0    1    0    0
## P105    0    0    1    0    0    1    0    1    0    1    1    0    0    1    0
## P106    0    0    1    1    0    0    1    0    0    0    0    0    0    1    0
##      M275 M276 M277 M278 M279 M280 M281 M282 M283 M284 M285 M286 M287 M288 M289
## P101    0    0    0    0    0    0    1    0    1    0    0    1    0    1    0
## P102    1    0    0    0    0    1    0    0    1    0    0    0    0    0    0
## P103    1    1    0    0    0    0    0    1    0    0    0    0    0    1    1
## P104    0    0    0    0    0    1    1    0    1    0    1    0    0    0    1
## P105    0    1    0    0    1    0    0    0    0    0    0    1    0    0    1
## P106    0    0    0    0    0    0    0    0    1    0    0    1    0    0    0
##      M290 M291 M292 M293 M294 M295 M296 M297 M298 M299 M300 M301 M302 M303 M304
## P101    0    0    0    0    0    0    1    0    0    0    1    0    0    0    0
## P102    0    1    0    0    0    0    0    1    0    0    0    0    1    1    1
## P103    0    0    0    0    1    0    0    0    0    1    0    0    0    1    1
## P104    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
## P105    1    0    0    0    0    0    1    0    0    0    0    1    0    0    0
## P106    0    0    0    1    0    1    0    0    0    0    0    0    0    0    0
##      M305 M306 M307 M308 M309 M310 M311 M312 M313 M314 M315 M316 M317 M318 M319
## P101    0    0    0    0    0    1    0    1    0    1    0    0    0    1    1
## P102    1    0    0    0    0    0    0    0    0    0    0    1    0    0    1
## P103    0    0    0    0    0    1    0    0    1    0    0    1    0    0    0
## P104    1    0    1    0    1    1    1    0    1    1    0    1    0    0    0
## P105    0    1    1    0    1    0    0    0    1    1    0    0    0    0    0
## P106    0    0    1    0    0    1    0    0    1    0    0    1    0    0    0
##      M320 M321 M322 M323 M324 M325 M326 M327 M328 M329 M330 M331 M332 M333 M334
## P101    0    0    1    0    1    0    1    0    1    0    1    0    0    0    0
## P102    0    0    0    1    0    0    0    0    0    0    1    0    0    1    0
## P103    0    0    1    1    0    1    1    1    0    0    0    1    1    0    1
## P104    1    0    1    1    0    0    1    0    0    0    0    0    0    1    0
## P105    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
## P106    0    0    0    1    0    1    1    0    0    1    0    0    1    1    0
##      M335 M336 M337 M338 M339 M340 M341 M342 M343 M344 M345 M346 M347 M348 M349
## P101    0    0    0    1    0    0    0    0    0    0    0    0    0    1    0
## P102    0    1    0    0    0    0    0    1    0    0    0    0    1    0    0
## P103    0    0    0    0    0    0    1    0    0    0    0    0    0    0    1
## P104    0    1    0    0    0    0    0    0    0    0    0    0    1    1    0
## P105    1    0    0    0    0    0    0    0    0    1    1    0    0    0    0
## P106    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
##      M350 M351 M352 M353 M354 M355 M356 M357 M358 M359 M360 M361 M362 M363 M364
## P101    1    1    0    0    0    1    1    0    0    0    0    1    1    0    0
## P102    1    1    0    0    0    0    0    0    1    0    1    0    1    0    0
## P103    1    1    0    0    1    0    0    0    1    0    0    0    1    0    0
## P104    0    0    1    0    1    0    0    0    0    0    0    0    1    0    1
## P105    0    1    0    0    0    1    1    1    0    1    0    1    0    0    0
## P106    1    1    0    0    0    0    0    0    1    0    0    0    1    0    0
##      M365 M366 M367 M368 M369 M370 M371 M372 M373 M374 M375 M376 M377 M378 M379
## P101    1    0    0    0    0    0    1    0    0    0    0    0    1    0    0
## P102    0    0    0    1    1    0    0    0    0    0    0    0    0    1    1
## P103    0    0    0    1    1    0    0    0    0    0    0    0    1    0    1
## P104    1    1    0    0    0    0    0    0    0    1    0    0    0    0    0
## P105    0    0    0    1    0    0    0    0    1    0    0    1    0    0    1
## P106    1    0    0    0    1    0    0    0    0    0    0    0    0    0    0
##      M380 M381 M382 M383 M384 M385 M386 M387 M388 M389 M390 M391 M392 M393 M394
## P101    0    0    0    1    0    0    0    0    0    1    1    0    0    0    0
## P102    0    0    0    0    0    1    1    0    0    1    0    1    0    1    0
## P103    1    1    0    1    0    0    1    1    0    0    1    0    0    0    0
## P104    0    0    0    1    1    0    0    1    0    0    0    0    0    1    1
## P105    0    0    0    0    0    0    0    0    0    0    0    0    0    1    1
## P106    0    0    0    0    1    0    1    0    1    1    0    0    0    1    0
##      M395 M396 M397 M398 M399 M400 M401 M402 M403 M404 M405 M406 M407 M408 M409
## P101    0    0    0    1    0    0    0    0    1    0    1    0    0    1    1
## P102    0    0    0    0    0    0    1    0    1    0    0    0    0    0    0
## P103    1    0    0    1    0    0    0    0    1    0    0    0    0    0    0
## P104    0    0    0    0    0    1    1    0    0    0    1    0    0    0    0
## P105    1    0    0    0    0    0    0    0    0    0    1    0    0    0    0
## P106    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
##      M410 M411 M412 M413 M414 M415 M416 M417 M418 M419 M420 M421 M422 M423 M424
## P101    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## P102    0    1    0    0    0    0    0    0    0    0    1    0    0    0    0
## P103    0    1    0    0    0    0    0    0    1    0    1    1    0    0    0
```

```
## P104    0    0    1    0    0    1    0    1    0    0    0    1    1    1    1
## P105    1    1    0    0    0    1    0    1    0    0    1    0    0    0    0
## P106    0    0    0    0    0    1    1    1    0    0    1    0    0    1    1
##      M425 M426 M427 M428 M429 M430 M431 M432 M433 M434 M435 M436 M437 M438 M439
## P101    0    1    1    0    0    1    0    1    0    0    0    0    1    1    0
## P102    0    1    0    0    0    0    1    0    1    0    0    0    1    0    0
## P103    1    0    0    0    0    0    1    0    0    0    1    0    0    0    0
## P104    0    0    1    0    1    0    0    0    1    0    1    0    1    0    1
## P105    1    1    0    1    0    0    0    0    1    0    0    0    1    0    1
## P106    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##      M440 M441 M442 M443 M444 M445 M446 M447 M448 M449 M450 M451 M452 M453 M454
## P101    0    0    0    0    1    0    0    0    1    0    1    1    0    0    0
## P102    1    0    1    0    1    1    0    0    1    0    0    0    1    0    0
## P103    0    0    1    1    0    1    0    1    1    0    1    1    0    0    1
## P104    0    0    1    0    0    1    0    1    1    0    0    0    0    0    0
## P105    0    1    0    0    1    0    0    0    0    1    0    0    0    1    0
## P106    0    1    1    1    0    1    1    0    0    0    0    0    0    0    0
##      M455 M456 M457 M458 M459 M460 M461 M462 M463 M464 M465 M466 M467 M468 M469
## P101    1    0    1    0    0    0    0    0    0    0    0    1    0    1    0
## P102    1    0    0    0    0    1    0    0    0    1    0    0    1    1    0
## P103    0    0    0    1    0    1    0    0    1    0    0    0    1    1    0
## P104    1    0    0    1    1    0    1    0    0    0    0    0    0    0    0
## P105    0    0    1    1    1    0    0    1    0    0    0    0    1    1    0
## P106    0    1    1    1    0    0    0    1    0    0    0    0    1    0    0
##      M470 M471 M472 M473 M474 M475 M476 M477 M478 M479 M480 M481 M482 M483 M484
## P101    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
## P102    0    0    0    0    0    0    1    1    1    0    0    0    1    1    0
## P103    1    0    0    1    0    1    1    0    1    0    0    0    1    0    0
## P104    1    1    1    1    0    1    0    0    1    0    0    0    1    0    0
## P105    1    0    0    0    0    0    0    0    1    1    0    0    1    1    1
## P106    0    1    0    0    0    1    1    0    0    1    0    1    0    0    0
##      M485 M486 M487 M488 M489 M490 M491 M492 M493 M494 M495 M496 M497 M498 M499
## P101    0    0    1    0    0    0    0    1    0    1    0    0    0    0    0
## P102    0    0    0    0    0    0    0    0    0    0    0    1    0    1    0
## P103    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
## P104    1    0    0    0    0    0    0    0    0    1    0    1    1    0    0
## P105    0    0    0    1    0    0    0    1    1    0    0    0    0    1    0
## P106    1    1    1    0    0    0    1    0    0    0    1    1    0    1    1
##      M500 M501 M502 M503 M504 M505 M506 M507 M508 M509 M510 M511 M512 M513 M514
## P101    0    0    1    0    0    1    1    1    0    0    0    1    0    0    1
## P102    1    0    1    0    0    0    0    1    1    0    0    0    0    0    1
## P103    0    0    0    0    0    1    0    0    0    0    1    0    1    1    0
## P104    0    1    1    0    0    1    0    0    0    0    1    0    0    1    0
## P105    0    0    0    1    0    1    0    0    0    0    0    1    0    0    1
## P106    1    0    1    0    0    0    0    0    0    0    1    1    1    1    1
##      M515 M516 M517 M518 M519 M520
## P101    0    1    0    1    0    1
## P102    0    0    0    0    0    0
## P103    0    0    1    0    0    0
## P104    0    1    0    1    0    1
## P105    1    0    0    0    1    1
## P106    0    0    0    0    0    0
```

```
dim(noallergy)
```

```
## [1] 100 520
```

```
n1 = dim(allergy)[1]
n2 = dim(noallergy)[1]
p = dim(noallergy)[2]

x.mean1 = sapply(allergy, mean)
x.mean2 = sapply(noallergy, mean)

p.hat = (x.mean1*n1+x.mean2*n2)/(n1+n2)
x.var = (p.hat*(1-p.hat))

# Test: H0.i: mu.i1 == mu.i2  vs H1.i: mu.i1 != mu.i2
```

5.4.1 Qa

```r
# a) Report the significant mutations, imposing a probability of at most 1%
#    that the single mutation is judged as influential if it isn't.
z.i = (x.mean1-x.mean2)/sqrt(x.var*(1/n1+1/n2))
p.i = ifelse(z.i<0, 2*pnorm(z.i),2*(1-pnorm(z.i)))

which(p.i<.01)
```

```
## M152 M160 M174 M188 M227 M258 M291 M336 M345 M355 M372 M446
##  152  160  174  188  227  258  291  336  345  355  372  446
```

5.4.2 Qb

```r
# b) Report the significant mutations, imposing a probability of at most 1%
#    that at least one of the non-influential mutations is judged as influential.
# Bonferoni test
k = 520
which(p.i*k<.01)
```

```
## M188 M372 M446
##  188  372  446
```

```r
# or
p.Bf = p.adjust(p.i, method='bonferroni')
which(p.Bf<.01)
```

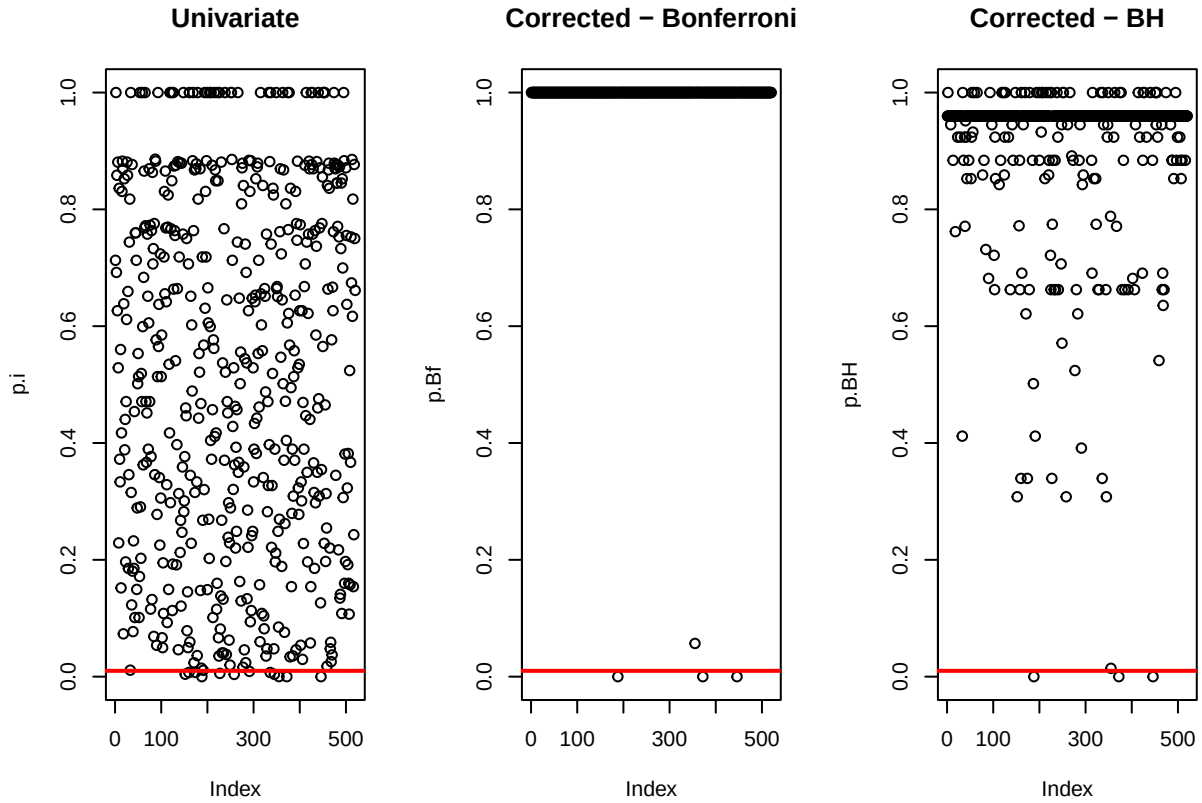
```
## M188 M372 M446
##  188  372  446
```

5.4.3 Qc

```r
# c) Report the significant mutations, imposing that the expected proportion
#    of mutations erroneously judged to be influential among all those judged to be
#    influential is lower than or equal to 1%
# Benjamini-Hockberg (control the false discovery rate)
p.BH = p.adjust(p.i, method='BH')
which(p.BH<.01)
```

```
## M188 M372 M446
##  188  372  446
```

```r
par(mfrow=c(1,3))
plot(p.i, main='Univariate')
abline(h=.01, lwd=2, col='red')

plot(p.Bf, main='Corrected - Bonferroni')
abline(h=.01, lwd=2, col='red')

plot(p.BH, main='Corrected - BH')
abline(h=.01, lwd=2, col='red')
```



5.5 Exam exercise

```
### Additional execise on mean differences between independent Gaussian:
### Pb 1 of 19/01/2022
### L'ho risolto io quindi non so se è giusto

# The Galician Food Association has launched an award for the Best Pulpo a la Gallega (meaning
# Galician-style octopus). As part of the challenge, two tasters are sent to evaluate the
# 30 finalist octopus dishes in ACoruna and the 30 finalist octopus dishes in Pontevedra.
# Files "acoruna.txt" and "pontevedra.txt" collect the evaluations on each of the finalist
# dishes given by the tasters in ACoruna and Pontevedra, respectively. Assume the evaluations
# on different dishes to be independent, and the evaluations of the two tasters on the same
# octopus dish to come from a bivariate Gaussian distribution.
# a) Perform a statistical test of level 99% to verify if the mean evaluations in the two
#    cities differ. State and verify the model assumptions.
# b) Interpret the results of the test at point (a) through two Bonferroni intervals of
#    global level 99% for appropriate differences in the mean. Comment the result.
# c) Is there statistical evidence to state that, at level 99%, the average evaluations of
#    A Coruna's octopus dishes are in mean higher than those of Pontevedra's octopus dishes?
#
#    [by average evaluation of a octopus dishes is meant the one obtained by averaging the
#    evaluations of the two tasters on that dishes]

acrn = read.table("Lab05/acoruna.txt")
pntv = read.table("Lab05/pontevedra.txt")

dim(acrn)
```

```
## [1] 30  2
```

```
dim(pntv)
```

```
## [1] 30  2
```

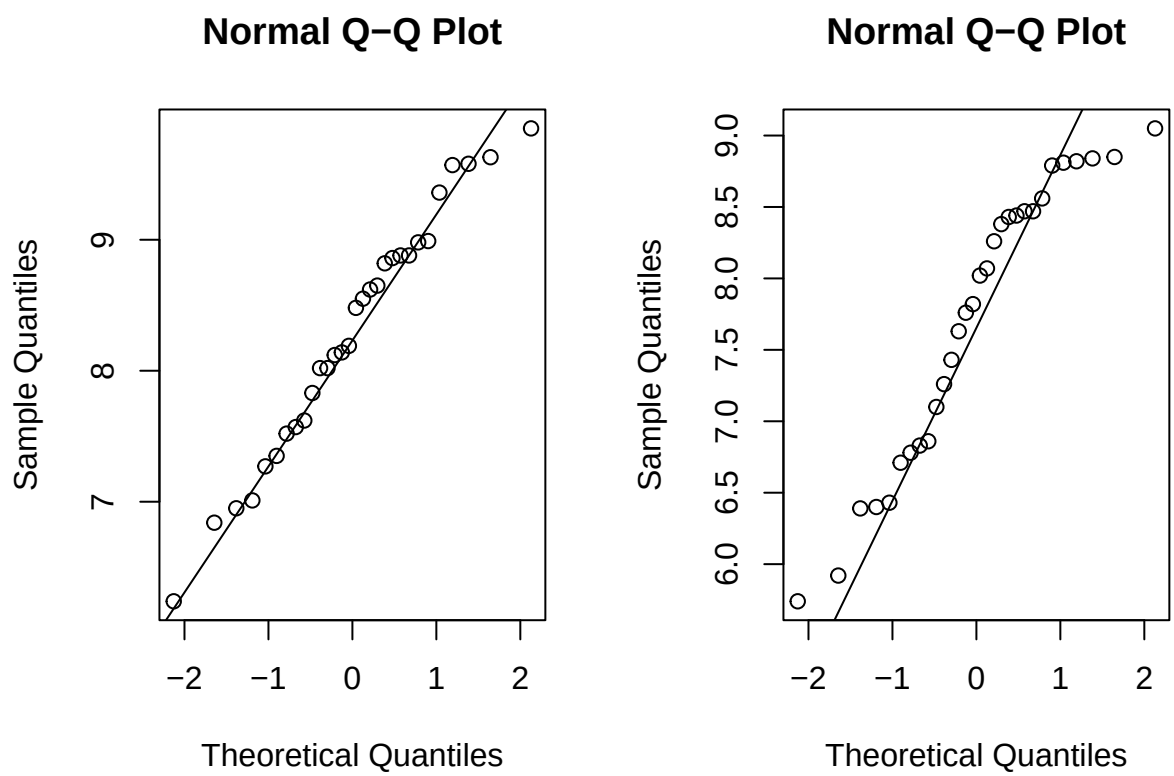
```
# T1 and T2 are the two tasters sent to the two locations
head(acrn)
```

```
##              T1   T2
## Octopus.1 8.62 8.07
## Octopus.2 8.88 8.02
## Octopus.3 7.83 8.47
## Octopus.4 9.63 8.82
## Octopus.5 8.02 8.43
## Octopus.6 8.86 8.38
```

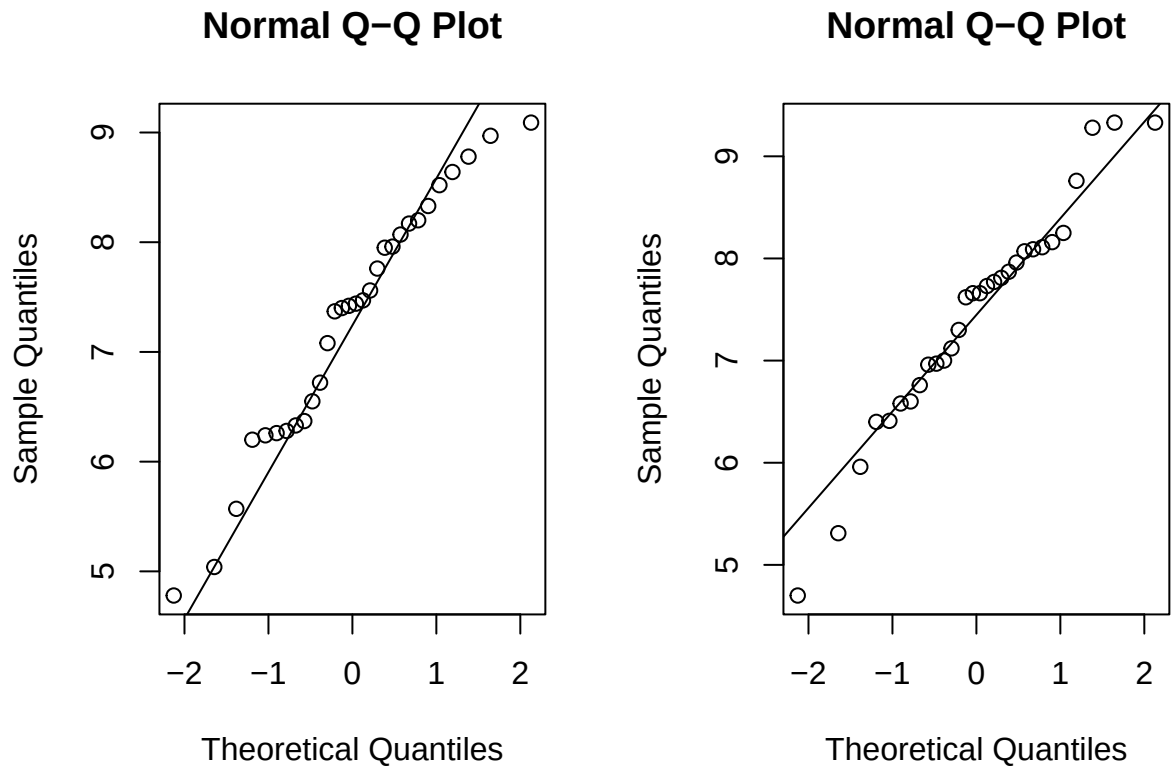
```
head(pntv)
```

```
##              T1   T2
## Octopus.1 5.04 4.70
## Octopus.2 6.37 6.60
## Octopus.3 6.72 8.25
## Octopus.4 7.56 7.30
## Octopus.5 6.24 5.31
## Octopus.6 8.17 8.07
```

```
par(mfrow=c(1,2))
qqnorm(acrn$T1)
qqline(acrn$T1)
qqnorm(acrn$T2)
qqline(acrn$T2)
```




```
par(mfrow=c(1,2))
qqnorm(pntv$T1)
qqline(pntv$T1)
qqnorm(pntv$T2)
qqline(pntv$T2)
```




```
mvn(data=acrn)$multivariateNormality
```

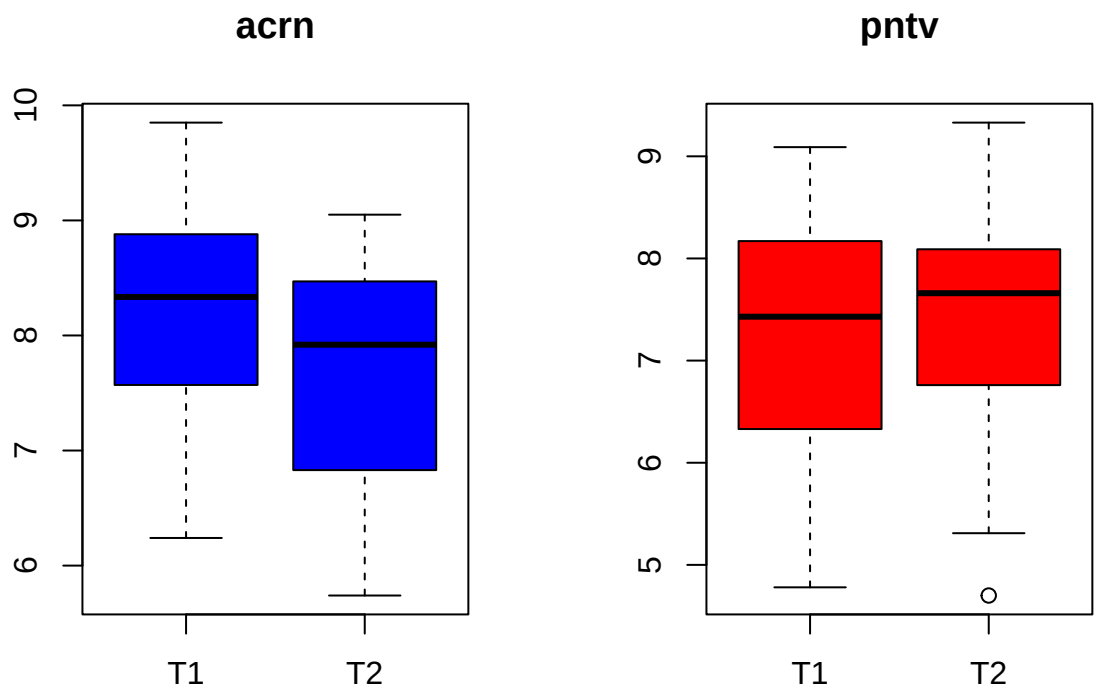
```
##            Test        HZ  p value MVN
## 1 Henze-Zirkler 0.6321491 0.151378 YES
```

```
mvn(data=pntv)$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.5403069 0.2651369 YES
```

5.5.1 Qa

```
par(mfrow=c(1,2))
boxplot(acrn, col="blue", main="acrn")
boxplot(pntv, col="red" , main="pntv")
```




```
# sembrano diverse?

# a) Perform a statistical test of level 99% to verify if the mean evaluations in the two
#    cities differ. State and verify the model assumptions.

acrn.mean = colMeans(acrn)
pntv.mean = colMeans(pntv)
acrn.cov = cov(acrn)
pntv.cov = cov(pntv)
# we build a random vector of the difference, since the two gaussian vectors (acrn and pntv)
# are independent. So
# D = A-P ~ N( muA-muP, covA+covB )
D = acrn
D$T1 = acrn$T1-pntv$T1
D$T2 = acrn$T2-pntv$T2

# Test H0: muD == 0 vs H1: muD != 0

# 1) gaussianity assumption
delta = 0
mshapiro.test(t(D))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.98092, p-value = 0.8494
```

```
mvn(data = D)$multivariateNormality
```

```
##            Test       HZ  p value MVN
## 1 Henze-Zirkler 0.449383 0.438557 YES
```

```
# 2) test statistic
n = dim(D)[1]
n
```

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

```r
meanD = colMeans(D)
meanD
```

```
##        T1        T2
## 0.9956667 0.2596667
```

```r
mu0 = c(0,0)
scovD = cov(D)
x.T2 = n * (meanD-mu0) %*% scovD %*% (meanD-mu0)
x.T2
```

```
##          [,1]
## [1,] 98.58743
```

```r
# Radius of the ellipsoid
cfr.fisher = ((n-1)*p/(n-p))*qf(1-alpha,p,n-p)
cfr.fisher
```

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

```r
# Test:
x.T2 < cfr.fisher # so we reject H0
```

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

```r
# Conclusion: there seems to be evidence to say that muD!=0, ie the mean evaluations
# in the two cities differ
```

5.5.2 Qb

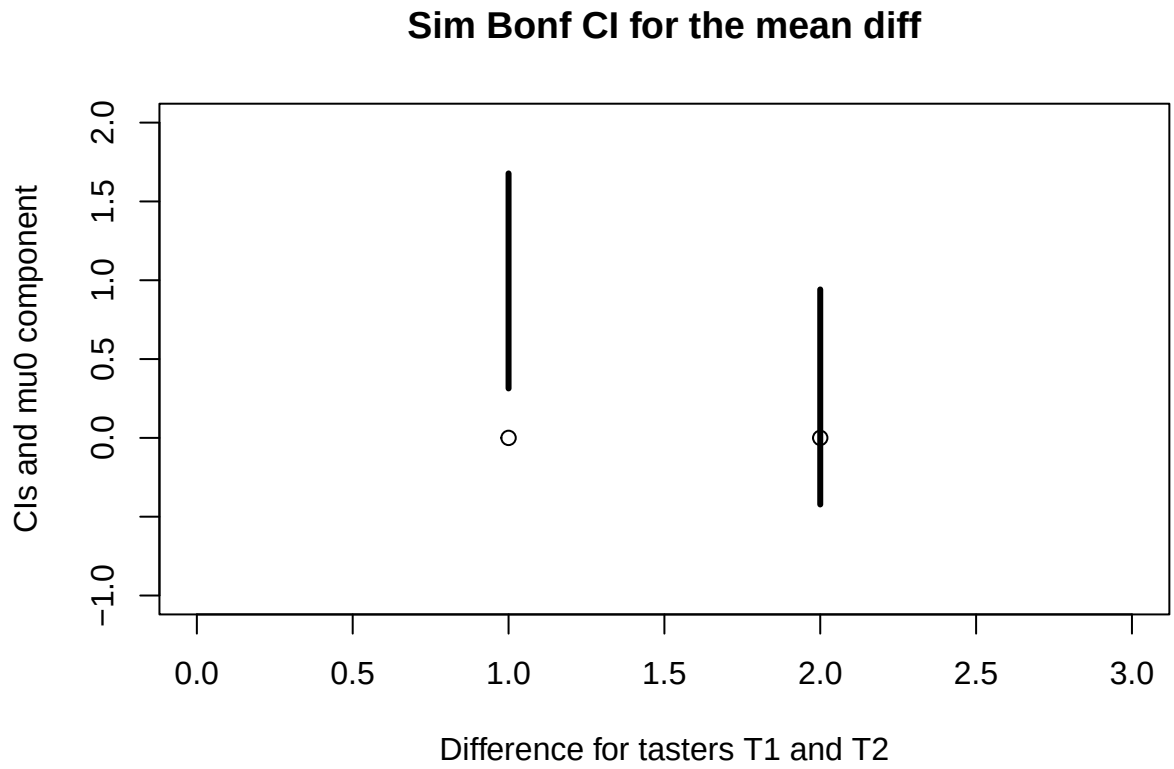
```r
# b) Interpret the results of the test at point (a) through two Bonferroni intervals of
#    global level 99% for appropriate differences in the mean. Comment the result.

# We are doing two CIs so for Bonferroni we set
k=2

alpha=0.01
# CI for the T1 difference
T1_eval_diff = cbind(
    inf = meanD[1] - qt(1-alpha/(2*k),n-1) * sqrt((scovD[1,1]-2*scovD[1,2]+scovD[2,2])/n),
    sup = meanD[1] + qt(1-alpha/(2*k),n-1) * sqrt((scovD[1,1]-2*scovD[1,2]+scovD[2,2])/n)
)
T2_eval_diff = cbind(
    inf = meanD[2] - qt(1-alpha/(2*k),n-1) * sqrt((scovD[1,1]-2*scovD[1,2]+scovD[2,2])/n),
    sup = meanD[2] + qt(1-alpha/(2*k),n-1) * sqrt((scovD[1,1]-2*scovD[1,2]+scovD[2,2])/n)
)

T1_eval_diff
```

```
##          inf      sup
## T1 0.3135622 1.677771
```

```r
T2_eval_diff
```

```
##           inf       sup
## T2 -0.4224378 0.9417712
```

```r
matplot(0:3,0:3, pch='',ylim=c(-1,2), main='Sim Bonf CI for the mean diff',
        xlab="Difference for tasters T1 and T2", ylab ="CIs and mu0 component")
segments(1, T1_eval_diff[1], 1, T1_eval_diff[2], lwd=3)
segments(2, T2_eval_diff[1], 2, T2_eval_diff[2], lwd=3)
points(1:2,mu0)
```



5.5.3 Qc

```
# c) Is there statistical evidence to state that, at level 99%, the average evaluations of
#    A Coruna's octopus dishes are in mean higher than those of Pontevedra's octopus dishes?
#    [by average evaluation of a octopus dishes is meant the one obtained by averaging the
#    evaluations of the two tasters on that dishes]

avg_A = rowMeans(acrn)
avg_P = rowMeans(pntv)
shapiro.test(avg_A)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  avg_A
## W = 0.94139, p-value = 0.09915
```

```
shapiro.test(avg_P)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  avg_P
## W = 0.9742, p-value = 0.659
```

```
# per entrambe accettiamo la gaussianità al 99% (?)
D_avg = avg_A - avg_P

# Test: H0 muD>0, H1 muD<=0
mu0 = 0
t0 = mean(D_avg)-mu0
t0
```

```
## [1] 0.6276667
```

```r
qcfr = sqrt(var(D_avg)/n)*qnorm(1-alpha,0,1)
qcfr
```

```
## [1] 0.6054607
```

```r
t0<qcfr # quindi accettiamo H0 (?)
```

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

6 Lab 06 - Anova and Manova

```r
###--------------------###
### LAB 6 (31/03/2023) ###
###--------------------###

### TOPICS:
### One-way ANOVA and MANOVA
### Two-ways ANOVA and MANOVA
#-------------------------------------------------------------------------------

library(MVN)
library(car)
library(heplots)
```

6.1 One-way ANOVA

```r
##### (p=1, g=6)
#####---------------
# Dataset Chicken Weights:
# Data from an experiment to measure and compare the effectiveness
# of various feed supplements on the growth rate of chickens.
# (71 observations)

# help(chickwts)
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

```r
dim(chickwts)
```

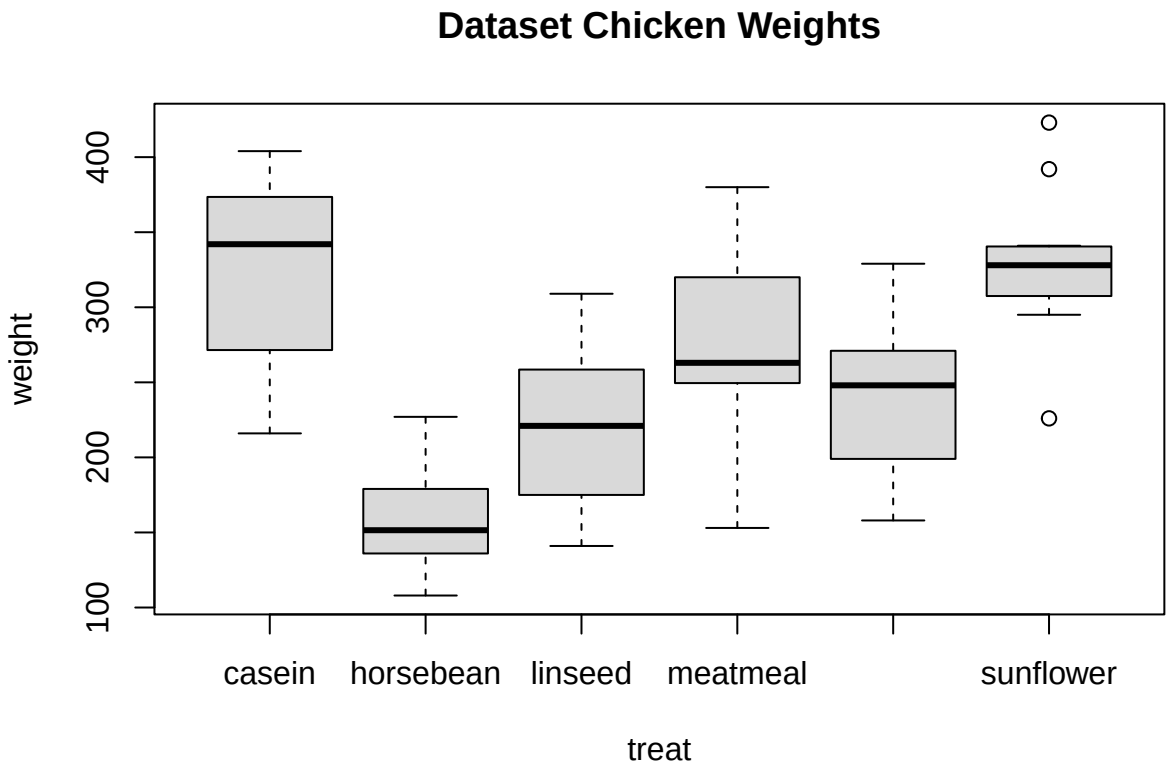
```
## [1] 71  2
```

```r
summary(chickwts)
```

```
##      weight             feed
##  Min.   :108.0   casein   :12
##  1st Qu.:204.5   horsebean:10
##  Median :258.0   linseed  :12
##  Mean   :261.3   meatmeal :11
##  3rd Qu.:323.5   soybean  :14
##  Max.   :423.0   sunflower:12
```

```r
attach(chickwts)

# Group-wise boxplot
```

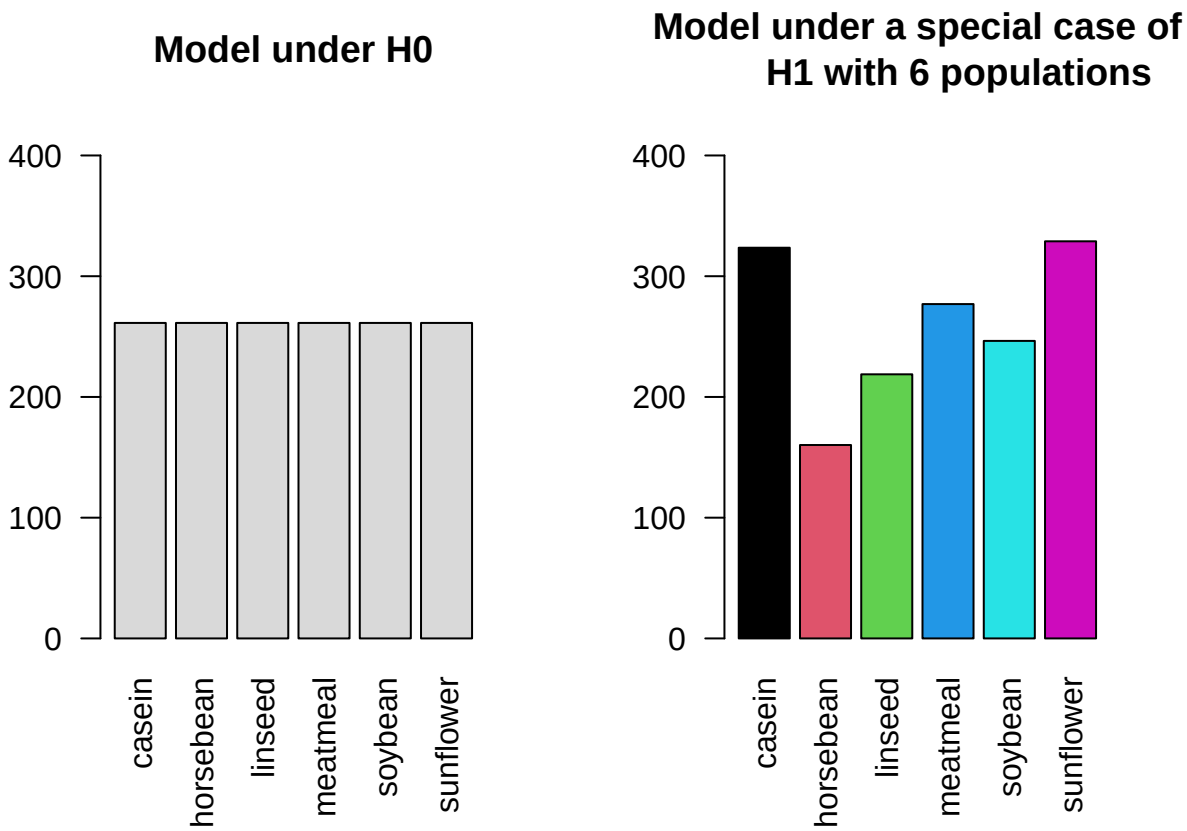
```r
plot(feed, weight, xlab='treat', ylab='weight', col='grey85',
     main='Dataset Chicken Weights')
```




```
# From the first two boxplots we can see that maybe there will be
# actually a difference in the groups
# But that is the between variability, but maybe we have a small number of data
# Like maybe the first group has lots of data, while the second less
# So always think in terms of variability, ie uncertainty
```

6.1.1 Model

```
### Model: weigth.ij = mu + tau.i + eps.ij; eps.ij ~ N(0,sigma^2)
### Test:
### H0: tau.1 = tau.2 = tau.3 = tau.4 = tau.5 = tau.6 = 0
### H1: (H0)^c
### ie,
### H0: The feed supplements don't have effect
###     (= "chickens belong to a single population")
### H1: At least one feed supplement has effect
###     (= "chickens belong to 2, 3, 4, 5 or 6 populations")

# Plot showing the estimator for the mean in H0 and a particular case of H1
# where we assume that there are as much populations as groups


par(mfrow=c(1,2))
barplot(rep(mean(weight),6), names.arg=levels(feed), ylim=c(0, max(weight)),
        las=2, col='grey85', main='Model under H0')
barplot(tapply(weight, feed, mean), names.arg=levels(feed), ylim=c(0, max(weight)),
        las=2, col=palette("default"), main='Model under a special case of
        H1 with 6 populations')
```




```
# This is a case of one-way ANOVA: one variable (weight) observed
# over g=6 levels (feed)
n       = length(feed)      # total number of obs. = n1 + ... + n6
ng      = table(feed)       # number of obs. in each group
treat   = levels(feed)      # levels of the treatment
g       = length(treat)     # number of levels (ie, of groups)
```

6.1.2 Verify the assumptions

```
# 1) normality (univariate! not pooled or multivariate) in each group (6 tests)
Ps = c(shapiro.test(weight[feed==treat[1]])$p,
       shapiro.test(weight[feed==treat[2]])$p,
       shapiro.test(weight[feed==treat[3]])$p,
       shapiro.test(weight[feed==treat[4]])$p,
       shapiro.test(weight[feed==treat[5]])$p,
       shapiro.test(weight[feed==treat[6]])$p)
Ps
```

```
## [1] 0.2591841 0.5264499 0.9034734 0.9611795 0.5063768 0.3602904
```

```
# 2) same covariance structure (= same sigma^2) the second hp for
# ANOVA, omoschedasticity
Var = c(var(weight[feed==treat[1]]),
        var(weight[feed==treat[2]]),
        var(weight[feed==treat[3]]),
        var(weight[feed==treat[4]]),
        var(weight[feed==treat[5]]),
        var(weight[feed==treat[6]]))
Var
```

```
## [1] 4151.720 1491.956 2728.568 4212.091 2929.956 2384.992
```

```
# test of homogeneity of variances for normal samples (Bartlett's test)
# H0: sigma.1 = sigma.2 = sigma.3 = sigma.4 = sigma.5 = sigma.6
# H1: there exist i,j s.t. sigma.i!=sigma.j
# WARNING: Test extremely sensitive to departures from normality (low robustness)
```

```r
# and unequality of variances
# Normality assumption mandatory
bartlett.test(weight, feed)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  weight and feed
## Bartlett's K-squared = 3.2597, df = 5, p-value = 0.66
```

```r
# So maybe do comunque l'ANOVA, questo test si "preoccupa troppo", l'ANOVA è
# molto robusta invece (al massimo gli outliers le danno fastidio)
```

6.1.3 Actual fit

```r
### One-way ANOVA
###-----------------
# help(aov)

# variable target ~ feature
fit = aov(weight ~ feed)
# Same syntax (and result) as it's like doing a linear model

summary(fit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## feed         5 231129   46226   15.37 5.94e-10 ***
## Residuals   65 195556    3009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
### How to read the summary:
#              Df   Sum Sq      Mean Sq      F value     Pr(>F)
#  treat      (g-1) SStreat  SStreat/(g-1)  Fstatistic  p-value [H0: tau.i=0 for
#  Residuals  (n-g) SSres     SSres/(n-g)                          every i]
#---
#  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
###

### We reject the test (as low pvalue), ie we have evidence to state that the
### treatment (feed supplement) has an effect on the growth rate of chicken.

### Which supplement is responsible for this? To see this, we need to
### do g*(g-1)/2 comparisons (all the pairs in g groups, g choose 2).
### We use Bonferroni as we look simultaneously at the intervals
k = g*(g-1)/2
alpha= 0.05

Mediag = tapply(weight, feed, mean) # group-wise means
SSres = sum(residuals(fit)^2)
S = SSres/(n-g)

# Example: CI for the difference "casein - horsebean"
paste(treat[1],"-",treat[2])
```

```
## [1] "casein - horsebean"
```

```r
as.numeric(c(Mediag[1]-Mediag[2] - qt(1-alpha/(2*k), n-g) * sqrt(S * ( 1/ng[1] +
                                                                      1/ng[2])),
             Mediag[1]-Mediag[2] + qt(1-alpha/(2*k), n-g) * sqrt(S * ( 1/ng[1] +
                                                                     1/ng[2]))))
```

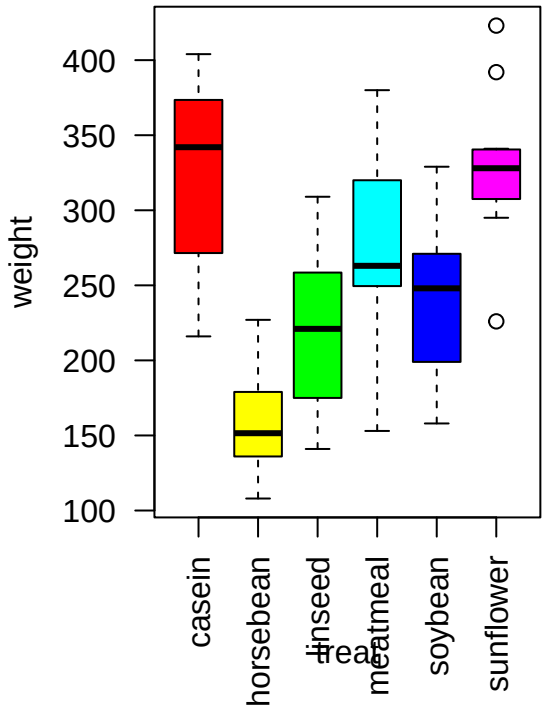
```
## [1]  91.81006 234.95661
```

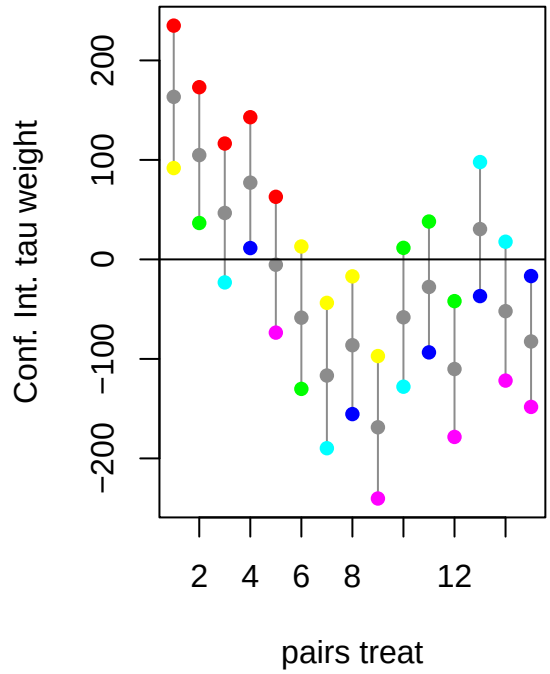
6.1.4 CI for all the differences

```r
ICrange=NULL
for(i in 1:(g-1)) {
    for(j in (i+1):g) {
        print(paste(treat[i],"-",treat[j]))
        print(as.numeric(c(Mediag[i]-Mediag[j] - qt(1-alpha/(2*k), n-g) *
                           sqrt(S * ( 1/ng[i] + 1/ng[j])),
                          Mediag[i]-Mediag[j] + qt(1-alpha/(2*k), n-g) *
                           sqrt(S * ( 1/ng[i] + 1/ng[j]))))
        ICrange=rbind(ICrange,as.numeric(c(Mediag[i]-Mediag[j] -
                                           qt(1-alpha/(2*k), n-g) * sqrt(S *
                                                       (1/ng[i] + 1/ng[j])),
                                          Mediag[i]-Mediag[j] +
                                           qt(1-alpha/(2*k), n-g) * sqrt(S *
                                                       (1/ng[i] + 1/ng[j]))))
    }}
```

```
## [1] "casein - horsebean"
## [1]  91.81006 234.95661
## [1] "casein - linseed"
## [1]  36.59089 173.07578
## [1] "casein - meatmeal"
## [1] -23.10193 116.45041
## [1] "casein - soybean"
## [1]  11.3947 142.9148
## [1] "casein - sunflower"
## [1] -73.57578  62.90911
## [1] "horsebean - linseed"
## [1] -130.12328   13.02328
## [1] "horsebean - meatmeal"
## [1] -189.7462  -43.6720
## [1] "horsebean - soybean"
## [1] -155.4390  -17.0181
## [1] "horsebean - sunflower"
## [1] -240.28994  -97.14339
## [1] "linseed - meatmeal"
## [1] -127.93526   11.61708
## [1] "linseed - soybean"
## [1] -93.43863  38.08149
## [1] "linseed - sunflower"
## [1] -178.40911  -41.92422
## [1] "meatmeal - soybean"
## [1] -36.86983  97.83087
## [1] "meatmeal - sunflower"
## [1] -121.7837   17.7686
## [1] "soybean - sunflower"
## [1] -148.24816  -16.72803
```

```r
par(mfrow=c(1,2))
plot(feed, weight, xlab='treat', ylab='weight', col = rainbow(6), las=2)

h = 1
plot(c(1,g*(g-1)/2),range(ICrange), pch='',xlab='pairs treat',
     ylab='Conf. Int. tau weight')
for(i in 1:(g-1)) {
    for(j in (i+1):g) {
        ind = (i-1)*g-i*(i-1)/2+(j-i)
        lines (c(h,h), c(ICrange[ind,1],ICrange[ind,2]), col='grey55');
        points(h, Mediag[i]-Mediag[j], pch=16, col='grey55');
        points(h, ICrange[ind,1], col=rainbow(6)[j], pch=16);
        points(h, ICrange[ind,2], col=rainbow(6)[i], pch=16);
        h = h+1
    }}
abline(h=0)
```






```
# All the ones that dont cross zero are the ones for which we would reject
# the fact that "there are no differences"

### Be careful to apply transitivity in the statistical context!!
### {A not significantly (stat.) different from B} [A = B]   (red vs cyan)
###                                and
### {B not significantly (stat.) different from C} [B = C]   (cyan vs green)
###                        does NOT imply
### {A not significantly (stat.) different from C} [A = C]   (red vs green)
### Note. If we accept H0, we are not proving that A=B but
###       we are saying that we can't prove that A!=B
```

6.1.5 Multiple testing

```
# Let's change the criterion to control the univariate rejection
# (multiple testing)

# We build k one-at-a-time confidence intervals, each of level alpha
# (not including the Bonferroni correction)
Auni = matrix(0,6,6)
for(i in 1:6) {
    for(j in i:6) {
        Auni[i,j] = Mediag[i]-Mediag[j] + qt(1-alpha/2, n-g) * sqrt( S *
                                                  ( 1/ng[i] + 1/ng[j] ) )}
    for(j in 1:i) {
        Auni[i,j] = Mediag[j]-Mediag[i] - qt(1-alpha/2, n-g) * sqrt( S *
                                                  ( 1/ng[i] + 1/ng[j] ) )}
    Auni[i,i]     = 0
}

par(mfrow=c(1,2))
h = 1
plot(c(1,g*(g-1)/2),range(Auni), pch='', xlab='pairs treat',
     ylab='CI delta weight', main='Univariate Conf. Int.', col='grey55')
for(i in 1:5) {
    for(j in (i+1):6) {lines (c(h,h), c(Auni[i,j],Auni[j,i]));
        points(h, Mediag[i]-Mediag[j], pch=16, col='grey55');
        points(h, Auni[i,j], col=rainbow(6)[i], pch=16);
```

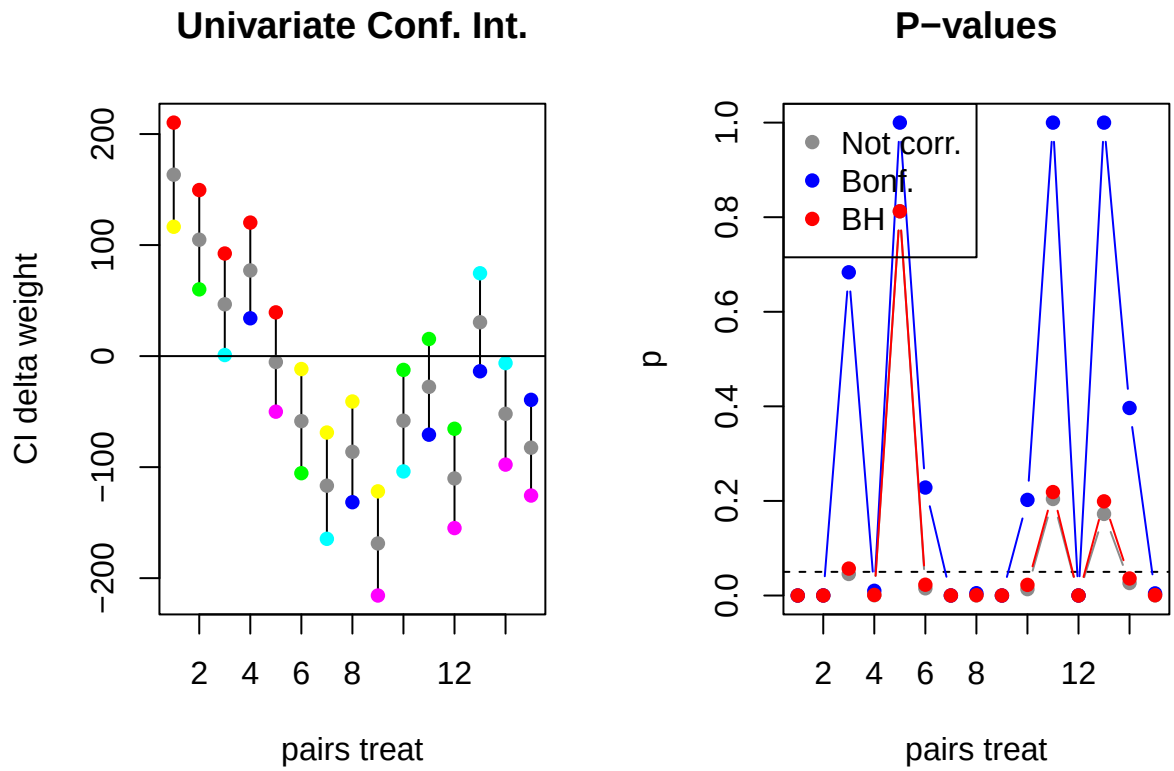
```r
        points(h, Auni[j,i], col=rainbow(6)[j], pch=16);
        h = h+1
    }}
abline(h=0)
# Now here without Bonferroni we get a stricter CI

# We compute the p-values of the univariate tests

# Matrix of tests for the difference between all the pairs
P = matrix(0,6,6)
for(i in 1:6) {
    for(j in i:6) {
        P[i,j] = (1-pt(abs((Mediag[i]-Mediag[j]) / sqrt( S * ( 1/ng[i] +
                                                 1/ng[j] ) ) ), n-g))*2}
    for(j in 1:i) {
        P[i,j] = (1-pt(abs((Mediag[i]-Mediag[j]) / sqrt( S * ( 1/ng[i] +
                                                1/ng[j] ) ) ), n-g))*2}
    P[i,i]     = 0
}
P
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 0.000000e+00 2.067997e-09 1.493344e-05 4.556672e-02 0.0006654079
## [2,] 2.067997e-09 0.000000e+00 1.522197e-02 7.478012e-06 0.0003246269
## [3,] 1.493344e-05 1.522197e-02 0.000000e+00 1.347894e-02 0.2041446467
## [4,] 4.556672e-02 7.478012e-06 1.347894e-02 0.000000e+00 0.1725539145
## [5,] 6.654079e-04 3.246269e-04 2.041446e-01 1.725539e-01 0.0000000000
## [6,] 8.124949e-01 8.203778e-10 6.211836e-06 2.643548e-02 0.0002980438
##              [,6]
## [1,] 8.124949e-01
## [2,] 8.203778e-10
## [3,] 6.211836e-06
## [4,] 2.643548e-02
## [5,] 2.980438e-04
## [6,] 0.000000e+00
```

```r
# Vector of p-values
p = c(P[1, 2:6], P[2, 3:6], P[3, 4:6], P[4, 5:6], P[5, 6])
p
```

```
##  [1] 2.067997e-09 1.493344e-05 4.556672e-02 6.654079e-04 8.124949e-01
##  [6] 1.522197e-02 7.478012e-06 3.246269e-04 8.203778e-10 1.347894e-02
## [11] 2.041446e-01 6.211836e-06 1.725539e-01 2.643548e-02 2.980438e-04
```

```r
plot(1:15, p, ylim=c(0,1), type='b', pch=16, col='grey55', xlab='pairs treat',
     main='P-values')
abline(h=alpha, lty=2)

# Bonferroni correction
p.bonf = p.adjust(p, 'bonf')
lines(1:15, p.bonf, col='blue', pch=16, type='b')

# Correction according to the false discovery rate (Benjamini-Hockberg)
p.fdr = p.adjust(p, 'fdr')
lines(1:15, p.fdr, col='red', pch=16, type='b')

legend('topleft', c('Not corr.', 'Bonf.', 'BH'), col=c('grey55', 'blue', 'red'),
       pch=16)
```




```
which(p.bonf<alpha)
```

```
## [1]  1  2  4  7  8  9 12 15
```

```
which(p.fdr<alpha)
```

```
##  [1]  1  2  4  6  7  8  9 10 12 14 15
```

```
# BH will give a FDR<5%
# Bonf will give a prob of error FWE<5% (?)

detach(chickwts)
```

6.2 One-way MANOVA

```
##### (p=4, g=3)
# So we have a vector of features, not just 1 as before (wheigth)
#####---------------

# help(iris)
head(iris)
```

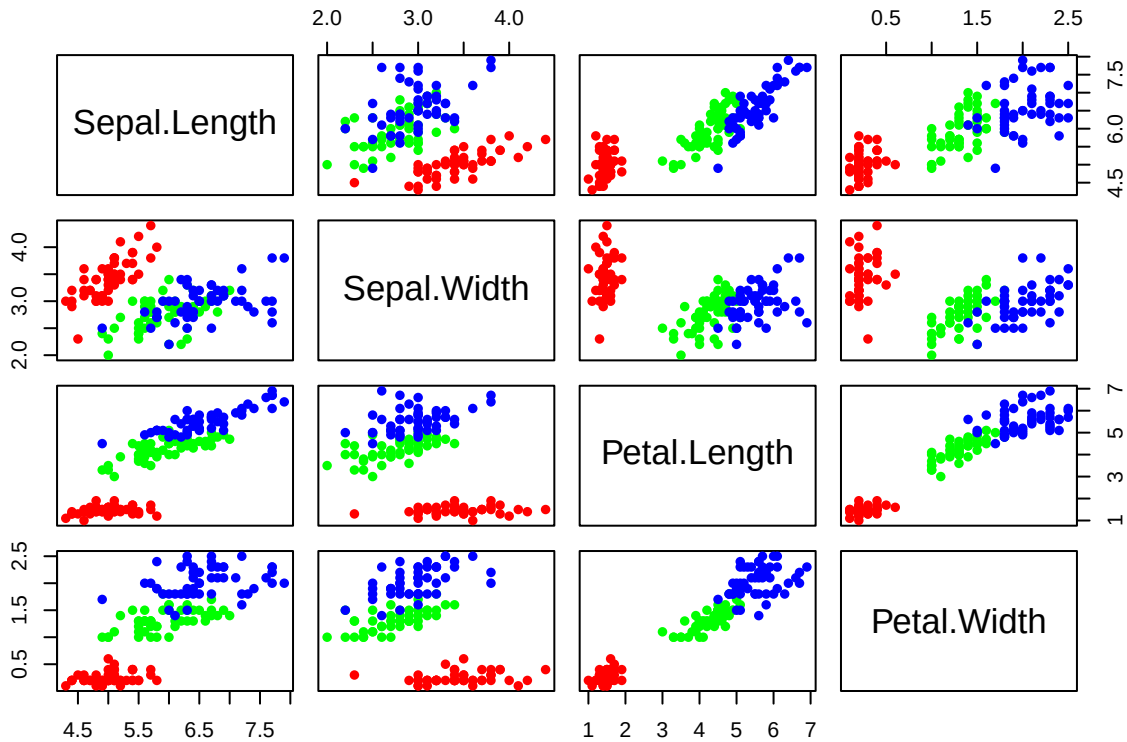
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
dim(iris)
```

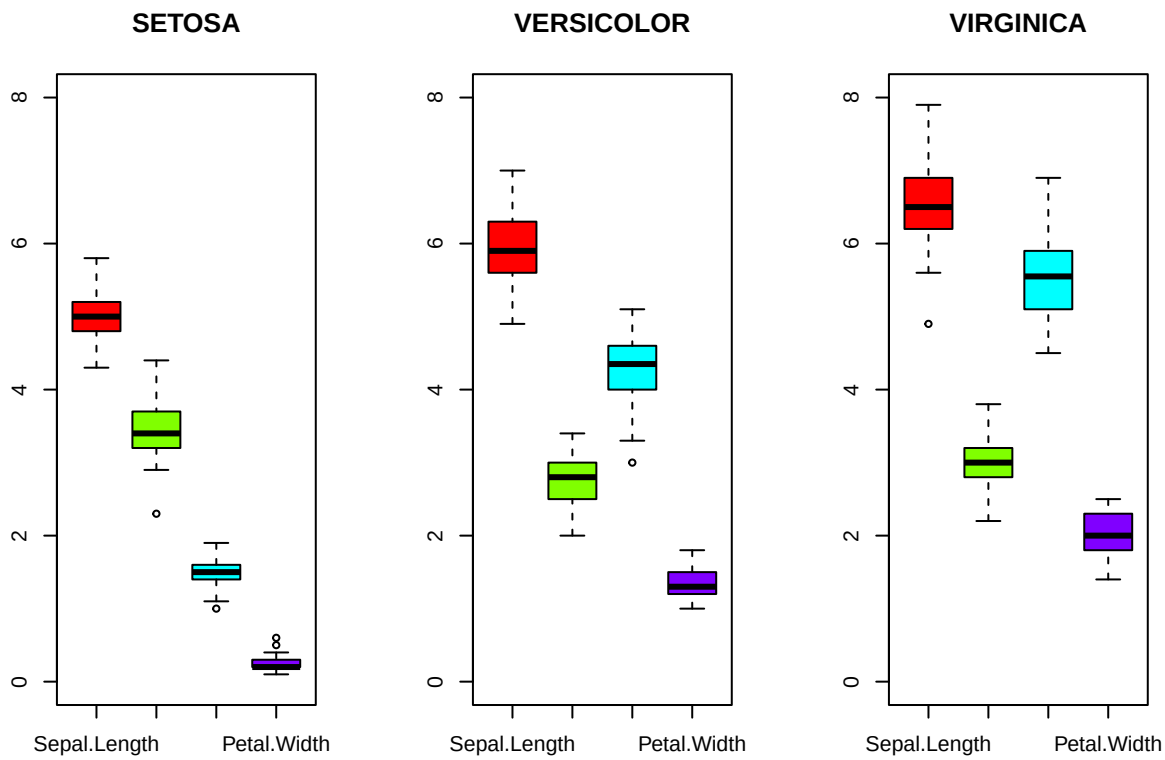
```
## [1] 150   5
```

```r
### Variables: Sepal and Petal Length and Sepal and Petal Width of iris
### (p = 4)
### Groups: species (setosa, versicolor, virginica; g = 3)
### n1 = n2 = n3 = 50 (balanced design)

attach(iris)

# Here we separate the data
species.name = factor(Species, labels=c('setosa','versicolor','virginica'))
iris4        = iris[,1:4] # variables

### Data exploration
colore = rep(rainbow(3), each = 50)
```

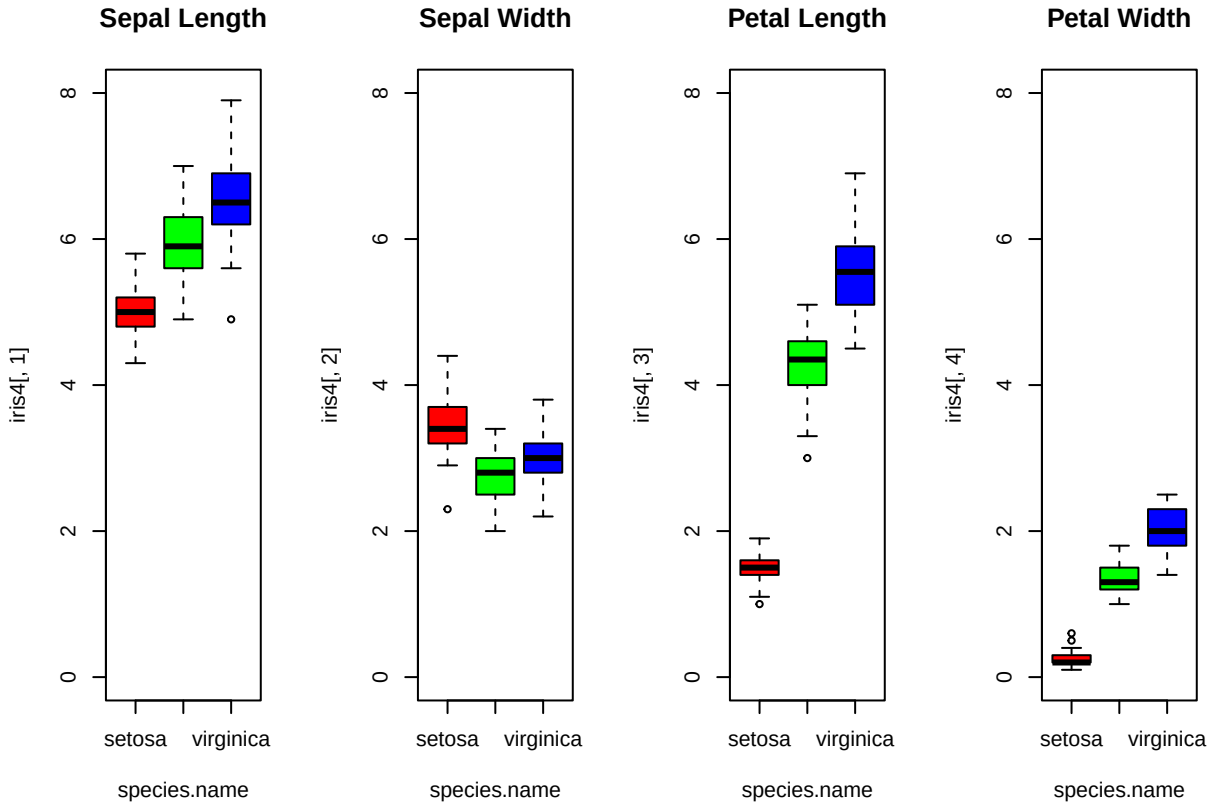
```r
pairs(iris4, col = colore, pch=16)
```




```r
# We can see that the red one is always almost different, and also quite the others
# So we can think that the anova will give us that there is a difference

# And the var/cov looks a bit different, but ANOVA is robust so nevermind
# And remember that if the problem is the scale of our data we can always
# transform them

i1 = which(species.name=='setosa')
i2 = which(species.name=='versicolor')
i3 = which(species.name=='virginica')

par(mfrow=c(1,3))
boxplot(iris4[i1,], main='SETOSA',     ylim=c(0,8), col = rainbow(4))
boxplot(iris4[i2,], main='VERSICOLOR', ylim=c(0,8), col = rainbow(4))
boxplot(iris4[i3,], main='VIRGINICA',  ylim=c(0,8), col = rainbow(4))
```




```
par(mfrow=c(1,4))
boxplot(iris4[,1]~species.name, main='Sepal Length', ylim=c(0,8), col = rainbow(3))
boxplot(iris4[,2]~species.name, main='Sepal Width', ylim=c(0,8), col = rainbow(3))
boxplot(iris4[,3]~species.name, main='Petal Length', ylim=c(0,8), col = rainbow(3))
boxplot(iris4[,4]~species.name, main='Petal Width', ylim=c(0,8), col = rainbow(3))
```



6.2.1 Model

```
### Model: X.ij = mu + tau.i + eps.ij
### eps.ij ~ N_p(0, Sigma), X.ij, mu, tau.i and eps.ij in R^4
# Same of the anova, but now mu tau ecc are vectors. in R^4
### Test:
### H0: tau.1 = tau.2 = tau.3  = (0,0,0,0)'
### H1: (H0)^c ie there exists at least two components with very different features
### (even just one pair of features)
### that is
### H0: The membership to an iris species hasn't any significant effect on the mean
###     of X.ij (in any direction of R^4)
### H1: There exists at least one direction in R^4 along which at least two species
###     have some feature significantly different

# This is a case of one-way MANOVA: four variables (Sepal.Length, Sepal.Width,
# Petal.Length, Petal.Width) observed over g=3 levels (setosa, versicolor, virginica)

n1 = length(i1)
n2 = length(i2)
n3 = length(i3)
n  = n1 + n2 + n3

g  = length(levels(species.name))
p  = 4
```

6.2.2 Verify the assumptions

```
# 1)  normality (multivariate) in each group (3 tests, on gaussian in R^4)
Ps = NULL
for(i in 1:g) {
    mvn.test = mvn(data = iris[get(paste('i',i, sep='')), 1:4])
    Ps = c(Ps, mvn.test$multivariateNormality$`p value`)
}
Ps
```

```
## [1] 0.04995356 0.22619915 0.49702369
```

```
# even here dont worry and go ahead with ANOVA

# 2) same covariance structure (= same covariance matrix Sigma)
# Scale the data just if the diagonal elements are really really different
S  =  cov(iris4)
S1 =  cov(iris4[i1,])
S2 =  cov(iris4[i2,])
S3 =  cov(iris4[i3,])

# Qualitatively:
round(S1, digits=1)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length          0.1         0.1            0           0
## Sepal.Width           0.1         0.1            0           0
## Petal.Length          0.0         0.0            0           0
## Petal.Width           0.0         0.0            0           0
```

```
round(S2, digits=1)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length          0.3         0.1          0.2         0.1
## Sepal.Width           0.1         0.1          0.1         0.0
## Petal.Length          0.2         0.1          0.2         0.1
## Petal.Width           0.1         0.0          0.1         0.0
```

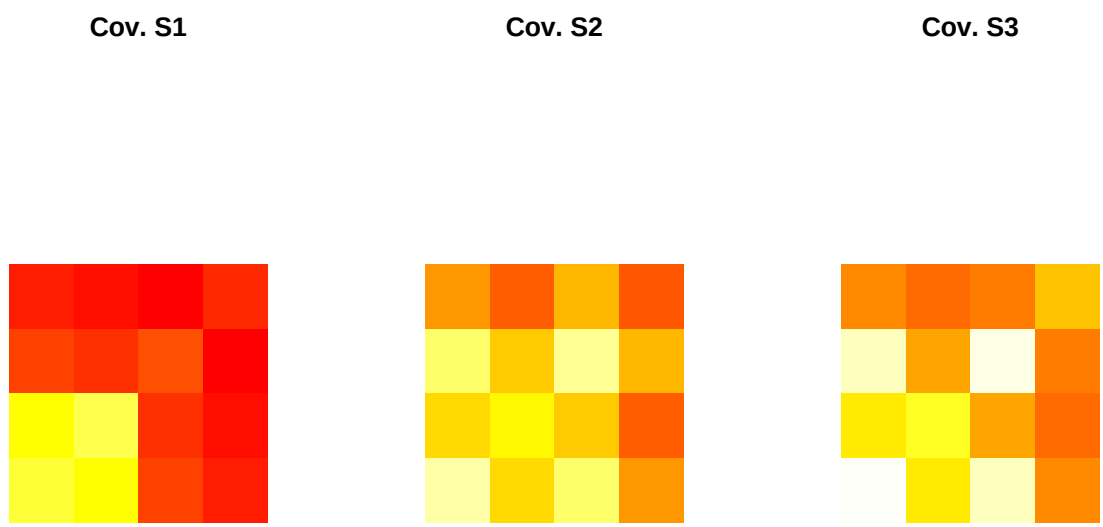
```
round(S3, digits=1)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length          0.4         0.1          0.3         0.0
## Sepal.Width           0.1         0.1          0.1         0.0
## Petal.Length          0.3         0.1          0.3         0.0
## Petal.Width           0.0         0.0          0.0         0.1
```

```r
# Box's M test for homogeneity of covariance matrices
# Should be done only if n_i > ~20 for all i, p < 5 and g < 5
# WARNING: Very sensitive to departure from normality and too restrictive
# for ANOVA especially if we have a high number of samples
summary(boxM(iris4, species.name))
```

```
## Summary for Box's M-test of Equality of Covariance Matrices
##
## Chi-Sq:   140.943
## df:   20
## p-value: < 2.2e-16
##
## log of Covariance determinants:
##     setosa versicolor  virginica     pooled
## -13.067360 -10.874325  -8.927058  -9.958539
##
## Eigenvalues:
##        setosa  versicolor  virginica     pooled
## 1 0.236455690 0.487873944 0.69525484 0.44356592
## 2 0.036918732 0.072384096 0.10655123 0.08618331
## 3 0.026796399 0.054776085 0.05229543 0.05535235
## 4 0.009033261 0.009790365 0.03426585 0.02236372
##
## Statistics based on eigenvalues:
##                 setosa   versicolor    virginica       pooled
## product   2.113088e-06 1.893828e-05 0.0001327479 4.732183e-05
## sum       3.092041e-01 6.248245e-01 0.8883673469 6.074653e-01
## precision 5.576122e-03 7.338788e-03 0.0169121236 1.304819e-02
## max       2.364557e-01 4.878739e-01 0.6952548382 4.435659e-01
```

```r
# Other way to compare: heatmap on the covariance matrices
par(mfrow=c(1,3))
image(S1, col=heat.colors(100),main='Cov. S1', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3), (0:100)/100, na.rm=TRUE))
image(S2, col=heat.colors(100),main='Cov. S2', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3), (0:100)/100, na.rm=TRUE))
image(S3, col=heat.colors(100),main='Cov. S3', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3), (0:100)/100, na.rm=TRUE))
```



```r
# In any case, we decide to carry on the MANOVA
# Note: We can verify the assumptions a posteriori on the residuals of
#       the estimated model
```

6.2.3 Actual fit

```r
### One-way MANOVA
###----------------

# help(manova)
# help(summary.manova)

# "target" ~ grouping factor
fit = manova(as.matrix(iris4) ~ species.name)
summary.manova(fit, test="Wilks")
```

```
##               Df    Wilks approx F num Df den Df    Pr(>F)
## species.name   2 0.023439   199.15      8    288 < 2.2e-16 ***
## Residuals    147
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Very low pvalue -> we reject H0 -> there is a difference

# These test are the ones based on the determinant/trace ecc
# Exact tests for p<=2 or g<=3 already implemented in R

# Note: since g=3 the test is exact
#       (cfr. JW pag.300)

### Reject the test, ie, we have statistical evidence to state that
### the factor "Species" has an effect on the mean features of the flowers.
### Who's the responsible for this?

# To discriminate where is the difference we can permform multiple ANOVA
# tests on each of the p (4 here) features

### Via ANOVA: for each of the p=4 variables we perform an ANOVA test
###            to verify if the membership to a group has influence
###            on the mean of the variable (we explore separately the
###            4 axes directions in R^4)
summary.aov(fit)
```

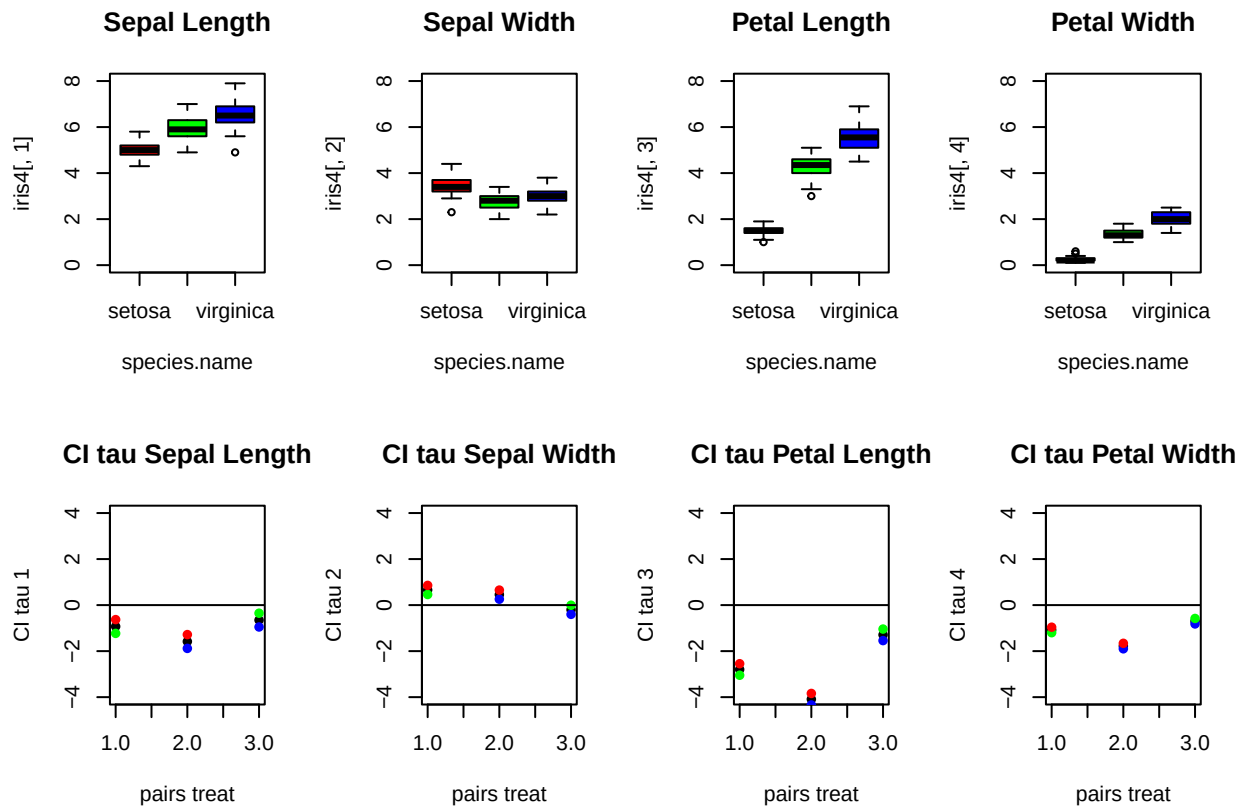
```
##  Response Sepal.Length :
##               Df Sum Sq Mean Sq F value    Pr(>F)
## species.name   2 63.212  31.606  119.26 < 2.2e-16 ***
## Residuals    147 38.956   0.265
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Response Sepal.Width :
##               Df Sum Sq Mean Sq F value    Pr(>F)
## species.name   2 11.345  5.6725   49.16 < 2.2e-16 ***
## Residuals    147 16.962  0.1154
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Response Petal.Length :
##               Df Sum Sq Mean Sq F value    Pr(>F)
## species.name   2 437.10 218.551  1180.2 < 2.2e-16 ***
## Residuals    147  27.22   0.185
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Response Petal.Width :
##               Df Sum Sq Mean Sq F value    Pr(>F)
## species.name   2 80.413  40.207  960.01 < 2.2e-16 ***
## Residuals    147  6.157   0.042
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Each of the 4 variables is significantly influenced by the
# factor species.
# Note: this analysis does NOT say:
#       a) which group differ
#       b) with respect to which variables the groups in (a) differ
# => As for the ANOVA, we build confidence intervals (many more!)

### Via Bonferroni
alpha = 0.05
k = p*g*(g-1)/2
qT = qt(1-alpha/(2*k), n-g)

W = summary.manova(fit)$SS$Residuals
m  = sapply(iris4, mean)         # estimates mu
m1 = sapply(iris4[i1,], mean)    # estimates mu.1 = mu + tau.1
m2 = sapply(iris4[i2,], mean)    # estimates mu.2 = mu + tau.2
m3 = sapply(iris4[i3,], mean)    # estimates mu.3 = mu + tau.3

inf12 = m1 - m2 - qT * sqrt(diag(W)/(n-g) * (1/n1+1/n2))
sup12 = m1 - m2 + qT * sqrt(diag(W)/(n-g) * (1/n1+1/n2))
inf13 = m1 - m3 - qT * sqrt(diag(W)/(n-g) * (1/n1+1/n3))
sup13 = m1 - m3 + qT * sqrt(diag(W)/(n-g) * (1/n1+1/n3))
inf23 = m2 - m3 - qT * sqrt(diag(W)/(n-g) * (1/n2+1/n3))
sup23 = m2 - m3 + qT * sqrt(diag(W)/(n-g) * (1/n2+1/n3))

CI = list(setosa_versicolor    = cbind(inf12, sup12),
          setosa_virginica     = cbind(inf13, sup13),
          versicolor_virginica = cbind(inf23, sup23))
CI
```

```
## $setosa_versicolor
##                   inf12      sup12
## Sepal.Length -1.2296895 -0.6303105
## Sepal.Width   0.4602476  0.8557524
## Petal.Length -3.0485232 -2.5474768
## Petal.Width  -1.1991389 -0.9608611
##
## $setosa_virginica
##                   inf13      sup13
## Sepal.Length -1.8816895 -1.2823105
## Sepal.Width   0.2562476  0.6517524
## Petal.Length -4.3405232 -3.8394768
## Petal.Width  -1.8991389 -1.6608611
##
## $versicolor_virginica
##                   inf23        sup23
## Sepal.Length -0.9516895 -0.352310544
## Sepal.Width  -0.4017524 -0.006247629
## Petal.Length -1.5425232 -1.041476789
## Petal.Width  -0.8191389 -0.580861086
```

```r
# Now we have a complete frame (intervals for all the components of
# tau_1, tau_2 e tau_3): it is fault of all the groups and all the
# variables!

par(mfrow=c(2,4))
boxplot(iris4[,1]~species.name, main='Sepal Length', ylim=c(0,8),
                                                     col = rainbow(3))
boxplot(iris4[,2]~species.name, main='Sepal Width', ylim=c(0,8),
                                                     col = rainbow(3))
boxplot(iris4[,3]~species.name, main='Petal Length', ylim=c(0,8),
                                                     col = rainbow(3))
boxplot(iris4[,4]~species.name, main='Petal Width', ylim=c(0,8),
                                                     col = rainbow(3))

mg = rbind(m1,m2,m3)
sp.name = c('Sepal Length','Sepal Width', 'Petal Length', 'Petal Width')
for(k in 1:4){
    plot(c(1,g*(g-1)/2),ylim=c(-4,4), xlim=c(1,3), pch='',
         xlab='pairs treat', ylab=paste('CI tau',k),
         main=paste('CI tau',sp.name[k]))
    lines (c(1,1), c(CI[[1]][k,1],CI[[1]][k,2]));
    points(1, mg[1,k]-mg[2,k], pch=16);
    points(1, CI[[1]][k,1], col=rainbow(g)[2], pch=16);
```

```
    points(1, CI[[1]][k,2], col=rainbow(g)[1], pch=16);
    lines (c(2,2), c(CI[[2]][k,1],CI[[2]][k,2]));
    points(2, mg[1,k]-mg[3,k], pch=16);
    points(2, CI[[2]][k,1], col=rainbow(g)[3], pch=16);
    points(2, CI[[2]][k,2], col=rainbow(g)[1], pch=16);
    lines (c(3,3), c(CI[[3]][k,1],CI[[3]][k,2]));
    points(3, mg[2,k]-mg[3,k], pch=16);
    points(3, CI[[3]][k,1], col=rainbow(g)[3], pch=16);
    points(3, CI[[3]][k,2], col=rainbow(g)[2], pch=16);
    abline(h=0)
}
```

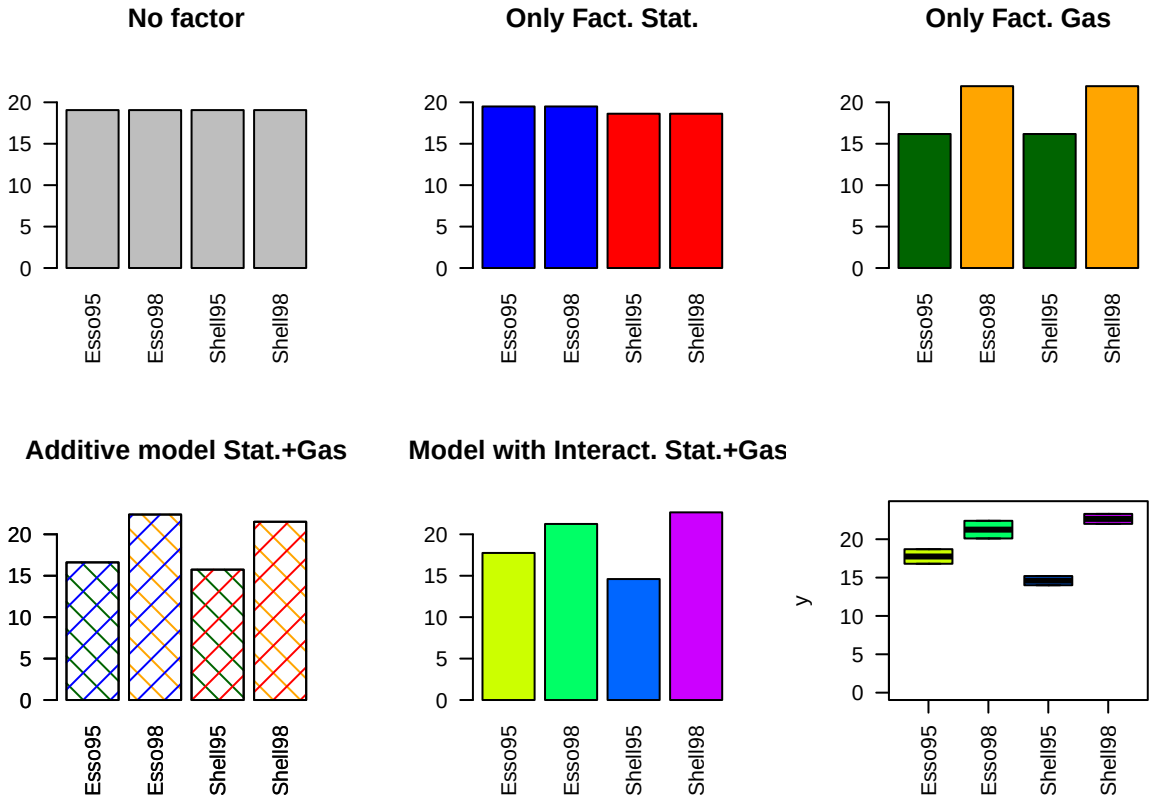



```
# At least one difference is enough for rejecting the MANOVA

# From here we see that everything is different (sepal length and widht, and the others)
# This analysis is like a preambol for the building a classifier, as we checked that
# the distribution of the features (the f_i) are different between the different species
```

6.3 Two-ways ANOVA

```
##### (p=1, g=2, b=2)
#####----------------------

### Problem 4 of 14/09/06
###------------------------
# In a small village in Switzerland there are two gas stations:
# one of Esso and one of Shell. Both sell either gasoline 95 octanes
# and 98 octanes.
# A young statistician wants to find out which is the best gas station
# and the best gasoline to refuel his car, in order to maximize the
# number of kilometers covered with a single refueling.
# After 8 refuelings, the measured performances are:
# km/l  : (18.7, 16.8, 20.1, 22.4, 14.0, 15.2, 22.0, 23.3)
# distr.: ('Esso','Esso','Esso','Esso','Shell','Shell','Shell','Shell')
# benz. : ('95','95','98','98','95','95','98','98')
# (a) Via a two-ways ANOVA identify which is the best station and the
```

```r
#     best gasoline for the young statistician to refuel his car.
# (b) Is there an interaction between the gas station and the gasoline?

### Variables: (target) distance covered [km/l]
### factor1:   Gas station   (0=Esso, 1=Shell)
### factor2:   Gasoline      (0=95,   1=98)
### Balanced design

kmpl        = c(18.7, 16.8, 20.1, 22.4, 14.0, 15.2, 22.0, 23.3)
distr       = factor(c('Esso','Esso','Esso','Esso','Shell','Shell','Shell','Shell'))
benz        = factor(c('95','95','98','98','95','95','98','98'))
distr_benz  = factor(c('Esso95','Esso95','Esso98','Esso98','Shell95','Shell95','Shell98','Shell98'))

g = length(levels(distr))
b = length(levels(benz))
#! So we have 8 obs, 4 groups, 2 obs for each group
n = length(kmpl)/(g*b)
#! n will now be the number of obs we have for each group

M           = mean(kmpl)
Mdistr      = tapply(kmpl, distr, mean)
Mbenz       = tapply(kmpl, benz, mean)
Mdistr_benz = tapply(kmpl, distr_benz, mean)


par(mfrow=c(2,3), las=2)
barplot(rep(M,4), names.arg=levels(distr_benz), ylim=c(0,24), main='No factor')
barplot(rep(Mdistr,each=2), names.arg=levels(distr_benz), ylim=c(0,24),
        col=rep(c('blue','red'),each=2), main='Only Fact. Stat.')
barplot(rep(Mbenz,times=2), names.arg=levels(distr_benz), ylim=c(0,24),
        col=rep(c('darkgreen','orange'),times=2), main='Only Fact. Gas')
barplot(c(Mdistr[1]+Mbenz[1]-M, Mdistr[1]+Mbenz[2]-M, Mdistr[2]+Mbenz[1]-M,
          Mdistr[2]+Mbenz[2]-M), names.arg=levels(distr_benz), ylim=c(0,24),
        col=rep(c('darkgreen','orange'),times=2), density=rep(10,4), angle=135,
        main='Additive model Stat.+Gas')
barplot(c(Mdistr[1]+Mbenz[1]-M, Mdistr[1]+Mbenz[2]-M, Mdistr[2]+Mbenz[1]-M,
          Mdistr[2]+Mbenz[2]-M), names.arg=levels(distr_benz), ylim=c(0,24),
        col=rep(c('blue','red'),each=2), density=rep(10,4), add=T)
barplot(Mdistr_benz, names.arg=levels(distr_benz), ylim=c(0,24),
        col=rainbow(5)[2:5], main='Model with Interact. Stat.+Gas.')
plot(distr_benz, kmpl, col=rainbow(5)[2:5], ylim=c(0,24),xlab='')
```




```
#! In the interaction we mean that we have an effect from the gas station,
#! the gasoline, and the combination of the two

#! These plots already show some differences, but now with the tests we
#! actually see if those differences are significative
```

6.3.1 Model

```
###----------------

### Model with interaction (complete model):
### X.ijk = mu + tau.i + beta.j + gamma.ij + eps.ijk; eps.ijk~N(0,sigma^2),
###     i=1,2 (effect station), j=1,2 (effect gasoline)

#! We dont even verify the assumptions as we have very few data
#! We can just see we have similar-scale data magari

#! target ~ var1 + var2 + interact(var1,var2)
fit.aov2.int = aov(kmpl ~ distr + benz + distr:benz)
summary.aov(fit.aov2.int)
```

6.3.1.1 With interaction

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## distr        1   1.53    1.53   1.018 0.37001
## benz         1  66.70   66.70  44.357 0.00264 **
## distr:benz   1  10.35   10.35   6.884 0.05857 .
## Residuals    4   6.01    1.50
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
### Test:
#! Firstly we test if we have interaction (for the error dof story)
### 1) H0: gamma.11 = gamma.12 = gamma.21 = gamma.22 = 0    vs H1: (H0)^c
###    ie,
###    H0: There is no significant interaction between the factors station
```

```
###        and gasoline in terms of performances
#! No interaction here means that just gasoline is important, not where we do it
#! or just where we go is important, and not what we choose
###    H1: There exists a significant interaction between the factors station
###        and gasoline in terms of performances
###
### 2) H0: tau.1 = tau.2 = 0    vs   H1: (H0)^c
###    ie,
###    H0: The effect "gas station" doesn't significantly influence performances
###    H1: The effect "gas station" significantly influences performances
###
### 3) H0: beta.1 = beta.2 = 0    vs   H1: (H0)^c
###    ie,
###    H0: The effect "gasoline" doesn't significantly influence performances
###    H1: The effect "gasoline" significantly influences performances


# Test 1): Let's focus on the row of the summary distr:benz :
#             Df Sum Sq Mean Sq F value  Pr(>F)
# distr:benz   1  10.35   10.35   6.884 0.05857 .
#
# The P-value of test 1) is 0.05857. Reject at 10%, don't reject at 1%,5%
# -> so maybe it is significant, maybe
#! F values are Mean sq of variabile / Mean sq of residuals

# Test 2): Let's focus on the row of the summary distr:
#             Df Sum Sq Mean Sq F value  Pr(>F)
# distr        1   1.53    1.53   1.018 0.37001
#
# The P-value of test 2) is 0.37001. Don't reject at 10%, 5%, 1%-> not significant

# Test 3): Let's focus on the row of the summary benz:
#             Df Sum Sq Mean Sq F value  Pr(>F)
# benz         1  66.70   66.70  44.357 0.00264 **
#
# The P-value of test 3) is 0.00264. Reject at 10%, 5%, 1% -> significant


# From test 1): We don't have strong evidence that the interaction has effect
# => remove the interaction term and estimate the model without interaction

#! As we want to flavour the simplest model, if there is no much difference
#! And so we get more power on the (measure of the) variability, to get a
#! less uncertain estimate for the variance
```

```
### Additive model:
### X.ijk = mu + tau.i + beta.j + eps.ijk; eps.ijk~N(0,sigma^2),
###     i=1,2 (effect station), j=1,2 (effect gasoline)
fit.aov2.ad = aov(kmpl ~ distr + benz)
summary.aov(fit.aov2.ad)
```

6.3.1.2 Without interaction

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## distr        1   1.53    1.53   0.468 0.52440
## benz         1  66.70   66.70  20.378 0.00632 **
## Residuals    5  16.37    3.27
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Remark: by removing the interaction, the residual degrees of freedom increase!

# Test: 2bis) H0: tau.1 = tau.2 = 0    vs   H1: (H0)^c
# From the summary:
#             Df Sum Sq Mean Sq F value  Pr(>F)
# distr        1   1.53    1.53   0.468 0.52440
# The P-value of test 2bis) is 0.52440. Don't reject at 10%, 5%, 1% -> not significant

# Test: 3bis) H0: beta.1 = beta.2 = 0    vs   H1: (H0)^c
# From the summary:
#             Df Sum Sq Mean Sq F value  Pr(>F)
```

```r
# benz         1  66.70   66.70  20.378 0.00632 **
# The P-value of test 2bis) is 0.00632. Reject at 10%, 5%, 1% -> significant

### Note: These aren't the only tests we can do!
### Example: global test for the significance of the two treatments
###          (model without interaction)
SSdistr = sum(n*b*(Mdistr - M)^2)              # or from the summary: 1.53
SSbenz  = sum(n*g*(Mbenz  - M)^2)              # or from the summary: 66.70
SSres   = sum((kmpl - M)^2) - (SSdistr+SSbenz)   # or from the summary: 16.37

#! H0: mu_ij = mu ie tau_i = beta_j = 0 forall i,j
#! H1: mu_ij = mu + tau_i + beta_j
Ftot      = ( (SSdistr + SSbenz) / ((g-1)+(b-1)))/(SSres / (n*g*b-g-b+1))
Ptot      = 1 - pf(Ftot, (g-1)+(b-1), n*g*b-g-b+1) # attention to the dof!
#! This is the pvalue, the P(being > qf) = 1 - pf(ecc)
Ptot
```

```
## [1] 0.01646126
```

```r
#! So there is an effect of at least one of the two variables

# Test 2bis): there is no evidence that the factor "gas station" has
#             effect on the performances (don't reject at any reasonable
#             level [high p-value!])
# => we remove the variable "station" and reduce to a one-way ANOVA

### Reduced additive model (ANOVA one-way, b=2):
### X.jk = mu + beta.j + eps.jk; eps.jk~N(0,sigma^2),
###     j=1,2 (effect gasoline)
fit.aov1 = aov(kmpl ~ benz)
#! We are keeping simplifying and making more accurate the model
summary.aov(fit.aov1)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## benz         1   66.7   66.70   22.36 0.00323 **
## Residuals    6   17.9    2.98
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
SSres = sum(residuals(fit.aov1)^2)
```

6.3.2 CIs for the differences

```r
### Interval at 90% for the differences (reduced additive model)
#! Difference between first level of the group wrt the second
### [b=2, thus one interval only]
IC = c(diff(Mbenz) - qt(0.95, (n*g-1)*b) * sqrt(SSres/((n*g-1)*b) *(1/(n*g) + 1/(n*g))),
        diff(Mbenz) + qt(0.95, (n*g-1)*b) * sqrt(SSres/((n*g-1)*b) *(1/(n*g) + 1/(n*g))))
names(IC) = c('Inf', 'Sup')
IC # IC for mu(98)-mu(95)
```

```
##      Inf      Sup
## 3.401886 8.148114
```

```r
#! So the 98 seems to let us make more kms
#! ICs are useful if we want to stress more on the lower bound, and also to
#! provide easy transformations of our data (f(data) -> f(IC))

diff(Mbenz) # point estimation
```

```
##    98
## 5.775
```

```r
#! But "pure differences" need to take care also of variability!

### Note: we should have verified the hypotheses of normality and variance
###       homogeneity for the complete model, but with only 2 data for each
###       group we can't perform the tests.
### => we verify the assumptions on the reduced model (one-way ANOVA)
```

```r
# 1) normality (univariate) in each group (2 tests)
Ps = c(shapiro.test(kmpl[ benz==levels(benz)[1] ])$p,
        shapiro.test(kmpl[ benz==levels(benz)[2] ])$p)
Ps
```

```
## [1] 0.9138954 0.6664017
```

```r
# 2) homogeneity of variances
bartlett.test(kmpl, benz)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  kmpl and benz
## Bartlett's K-squared = 0.42658, df = 1, p-value = 0.5137
```

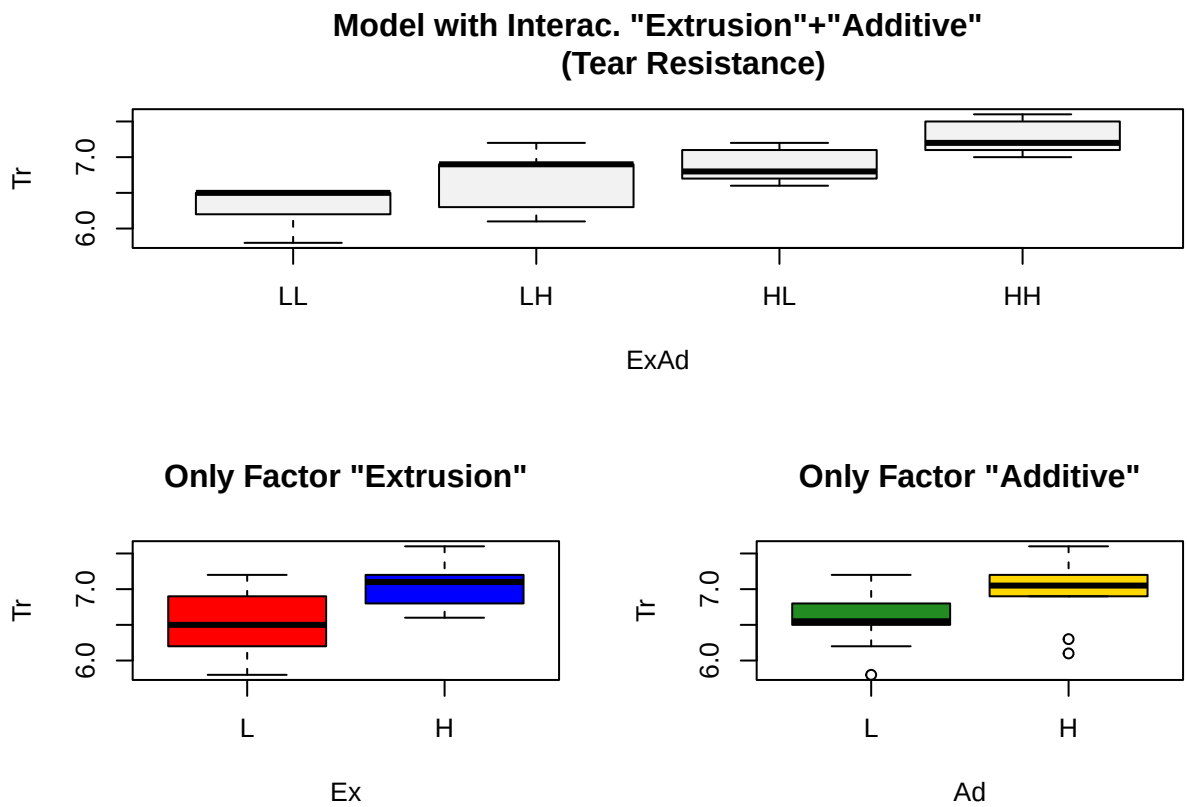
6.4 Two-ways MANOVA

```r
##### (p=3, g=2, b=2)
#! Three features, two groups/factors with two sub groups
#####----------------------

### Example 6.13 JW (p.319)
###------------------------
# The optimum conditions for extruding plastic films have been
# examined using a technique called Evolutionary Operation. In
# the course of the study, three responses were measured:
# X.1 = Tear Resistance
# X.2 = Gloss
# X.3 = Opacity
# at two levels of the factors:
# factor.1 = Rate of Extrusion     (0=Low level, 1=High level)
# factor.2 = Amount of an Additive (0=Low level, 1=High level)
# The measurements were repeated n=5 times at each combination
# of the factor levels (2*2 combinations, [0|1] && [0|1])

plastic = read.table('Lab06/T6-4.dat',col.names=c('Ex','Ad','Tr','Gl','Op'))
plastic
```

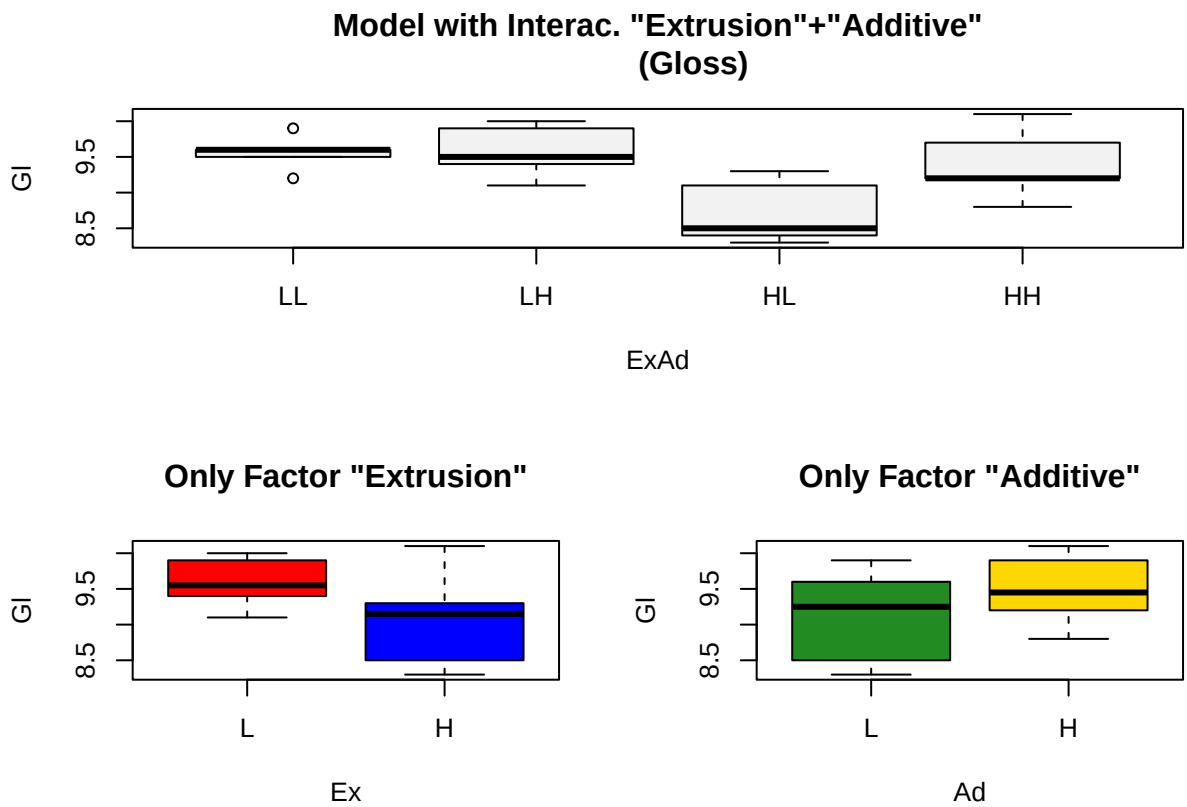
```
##    Ex Ad  Tr   Gl  Op
## 1   0  0 6.5  9.5 4.4
## 2   0  0 6.2  9.9 6.4
## 3   0  0 5.8  9.6 3.0
## 4   0  0 6.5  9.6 4.1
## 5   0  0 6.5  9.2 0.8
## 6   0  1 6.9  9.1 5.7
## 7   0  1 7.2 10.0 2.0
## 8   0  1 6.9  9.9 3.9
## 9   0  1 6.1  9.5 1.9
## 10  0  1 6.3  9.4 5.7
## 11  1  0 6.7  9.1 2.8
## 12  1  0 6.6  9.3 4.1
## 13  1  0 7.2  8.3 3.8
## 14  1  0 7.1  8.4 1.6
## 15  1  0 6.8  8.5 3.4
## 16  1  1 7.1  9.2 8.4
## 17  1  1 7.0  8.8 5.2
## 18  1  1 7.2  9.7 6.9
## 19  1  1 7.5 10.1 2.7
## 20  1  1 7.6  9.2 1.9
```

```r
Ex   = factor(plastic$Ex, labels=c('L','H')) # Treat.1
Ad   = factor(plastic$Ad, labels=c('L','H')) # Treat.2

ExAd = Ex
levels(ExAd) = c('LL','LH','HL','HH')
ExAd[Ex=='L' & Ad=='L'] = 'LL'
ExAd[Ex=='L' & Ad=='H'] = 'LH'
ExAd[Ex=='H' & Ad=='L'] = 'HL'
ExAd[Ex=='H' & Ad=='H'] = 'HH'
```

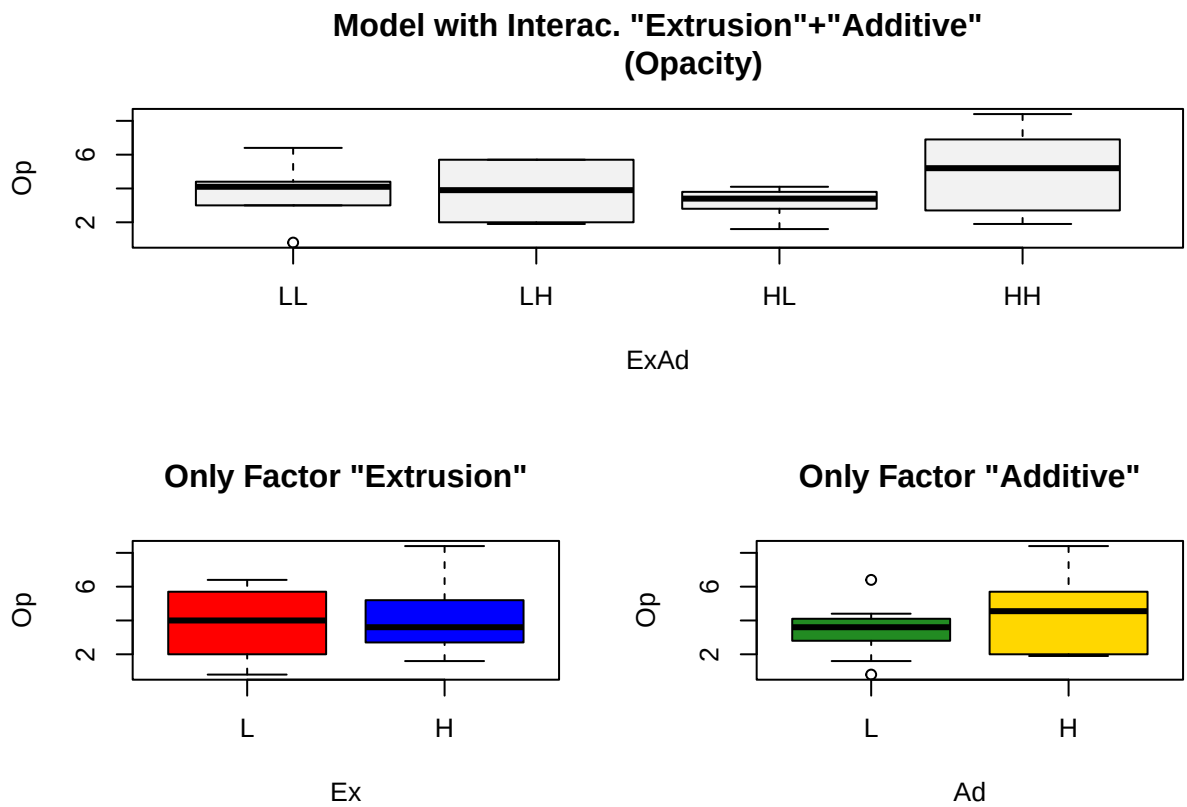
```r
#! The actual three features measured
plastic3  = plastic[,3:5]

### Graphical exploration of the data
# effect of the treatments + their interaction on the first variable

layout(matrix(c(1,1,2,3), 2, byrow=T))
boxplot(plastic3[,1]~ExAd, main='Model with Interac. "Extrusion"+"Additive"
        (Tear Resistance)', ylab='Tr', col='grey95')
boxplot(plastic3[,1]~Ex,   main='Only Factor "Extrusion"'  , ylab='Tr',
        col=c('red','blue'))
boxplot(plastic3[,1]~Ad,   main='Only Factor "Additive"'   , ylab='Tr',
        col=c('forestgreen','gold'))
```




```r
# effect of the treatments + their interaction on the second variable


layout(matrix(c(1,1,2,3), 2, byrow=T))
boxplot(plastic3[,2]~ExAd, main='Model with Interac. "Extrusion"+"Additive"
        (Gloss)', ylab='Gl', col='grey95')
boxplot(plastic3[,2]~Ex,   main='Only Factor "Extrusion"'  , ylab='Gl',
        col=c('red','blue'))
boxplot(plastic3[,2]~Ad,   main='Only Factor "Additive"'   , ylab='Gl',
        col=c('forestgreen','gold'))
```




```
# effect of the treatments + their interaction on the third variable


layout(matrix(c(1,1,2,3), 2, byrow=T))
boxplot(plastic3[,3]~ExAd, main='Model with Interac. "Extrusion"+"Additive"
        (Opacity)', ylab='Op', col='grey95')
boxplot(plastic3[,3]~Ex,   main='Only Factor "Extrusion"'  , ylab='Op',
        col=c('red','blue'))
boxplot(plastic3[,3]~Ad,   main='Only Factor "Additive"'   , ylab='Op',
        col=c('forestgreen','gold'))
```



6.4.1 Model

```
### Model with interaction (complete model):
### X.ijk = mu + tau.i + beta.j + gamma.ij + eps.ijk; eps.ijk~N_p(0,Sigma), [p=3]
###     i=1,2 (effect Extrusion), j=1,2 (effect Additive),
###     X.ijs, mu, tau.i, beta.j, gamma.ij in R^3

library(MVN)
### Verify the assumptions (although we only have 5 data in each group!)
# 1) normality (multivariate) in each group (4 test)

Ps = c(mvn(plastic3[ExAd==levels(ExAd)[1],])$multivariateNormality$`p value`,
        mvn(plastic3[ExAd==levels(ExAd)[2],])$multivariateNormality$`p value`,
        mvn(plastic3[ExAd==levels(ExAd)[3],])$multivariateNormality$`p value`,
        mvn(plastic3[ExAd==levels(ExAd)[4],])$multivariateNormality$`p value`)
```

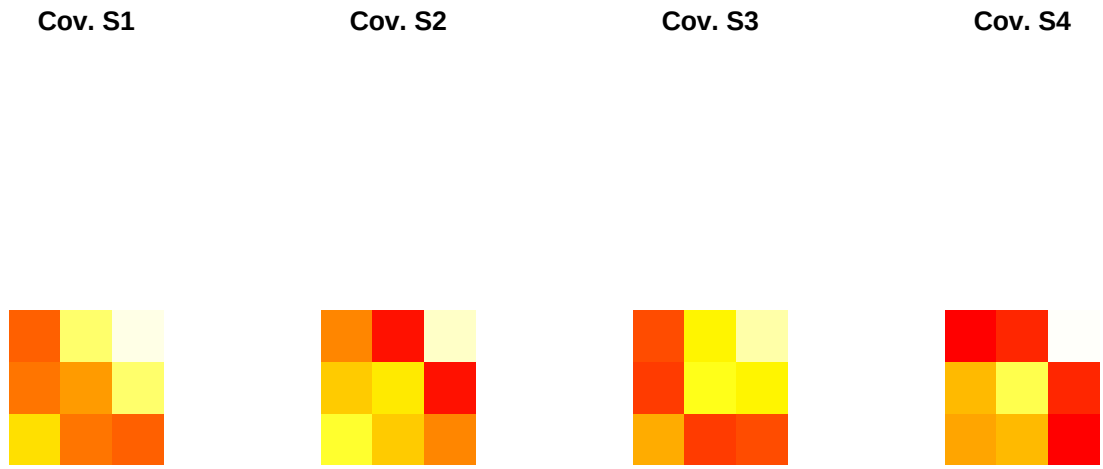
```
## Error in FUN(newX[, i], ...): sample size must be greater than 7
```

```
Ps
```

```
## [1] 0.9138954 0.6664017
```

```
#! This gives an error: we have too few data

# 2) homogeneity of the covariance (qualitatively)
S1 =  cov(plastic3[ ExAd==levels(ExAd)[1],])
S2 =  cov(plastic3[ ExAd==levels(ExAd)[2],])
S3 =  cov(plastic3[ ExAd==levels(ExAd)[3],])
S4 =  cov(plastic3[ ExAd==levels(ExAd)[4],])

par(mfrow=c(1,4))
image(S1, col=heat.colors(100),main='Cov. S1', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3,S4), (0:100)/100, na.rm=TRUE))
image(S2, col=heat.colors(100),main='Cov. S2', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3,S4), (0:100)/100, na.rm=TRUE))
image(S3, col=heat.colors(100),main='Cov. S3', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3,S4), (0:100)/100, na.rm=TRUE))
```

```
image(S4, col=heat.colors(100),main='Cov. S4', asp=1, axes = FALSE,
      breaks = quantile(rbind(S1,S2,S3,S4), (0:100)/100, na.rm=TRUE))
```




```
###-----------------
### Model with interaction (complete model):
### X.ijk = mu + tau.i + beta.j + gamma.ij + eps.ijk; eps.ijk~N_p(0,Sigma), [p=3]
###     i=1,2 (effect Extrusion), j=1,2 (effect Additive),
###     X.ijs, mu, tau.i, beta.j, gamma.ij in R^3

#! So with interaction for now (well we just started)

#! And Manova as we take the whole 3 (numeric) cols of the plastic data
fit = manova( as.matrix(plastic3) ~ Ex + Ad + Ex:Ad)
summary.manova(fit, test="Wilks")
```

6.4.1.0.1 With interaction

```
##           Df   Wilks approx F num Df den Df   Pr(>F)
## Ex         1 0.38186   7.5543      3     14 0.003034 **
## Ad         1 0.52303   4.2556      3     14 0.024745 *
## Ex:Ad      1 0.77711   1.3385      3     14 0.301782
## Residuals 16
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#! The interaction does not seem significant
### Model without interaction (additive model):
### X.ijk = mu + tau.i + beta.j + eps.ijk; eps.ijk~N_p(0,Sigma), [p=3]
###     i=1,2 (effect Extrusion), j=1,2 (effect additive),
###     X.ijs, mu, tau.i, beta.j, in R^3
fit2= manova( as.matrix(plastic3) ~ Ex + Ad)
summary.manova(fit2, test="Wilks")
```

6.4.1.1 Without interaction

```
##           Df   Wilks approx F num Df den Df  Pr(>F)
## Ex         1 0.38684   7.9253      3     15 0.00212 **
## Ad         1 0.55384   4.0279      3     15 0.02753 *
## Residuals 17
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Both the treatments have a significant effect on the mean (but not
# their interaction, that we could remove)

# Let's verify if this is true for all the variables through appropriate
# conf. int. and tests on the components:

# ANOVA on the components (we look at the 3 axes-directions in R^3
#                          separately)
#! As Manova just says "there is an effect on the set of those variables"
#! But to answer to which one we come back to anova
summary.aov(fit2)
```

```
##  Response Tr :
##             Df Sum Sq Mean Sq F value    Pr(>F)
## Ex           1 1.7405 1.74050  16.769 0.0007549 ***
## Ad           1 0.7605 0.76050   7.327 0.0149597 *
## Residuals   17 1.7645 0.10379
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Response Gl :
##             Df Sum Sq Mean Sq F value  Pr(>F)
## Ex           1 1.3005 1.30050  6.9688 0.01720 *
## Ad           1 0.6125 0.61250  3.2821 0.08774 .
## Residuals   17 3.1725 0.18662
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Response Op :
##             Df Sum Sq Mean Sq F value Pr(>F)
## Ex           1  0.421  0.4205  0.1038 0.7513
## Ad           1  4.901  4.9005  1.2094 0.2868
## Residuals   17 68.884  4.0520
```

6.4.2 CIs for the differences

```
# Bonferroni
#! To build Simulateneous ICs for each possible differences
alpha = 0.05
g = 2 #! group/factor 1 has two values
b = 2 #! group/factor 2 has two values
p = 3 #! we study a set of 3 featrues
n = 5 #! with 5 obs for each combination
N = n*g*b # 20

#! Variance within
W = summary.manova(fit2)$SS$Residuals

# how many comparisons?
k = g*(g-1)/2*p + b*(b-1)/2*p
# because we have: g levels on the first treatment on p components
#                  b levels on the second treatment on p components
k
```

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

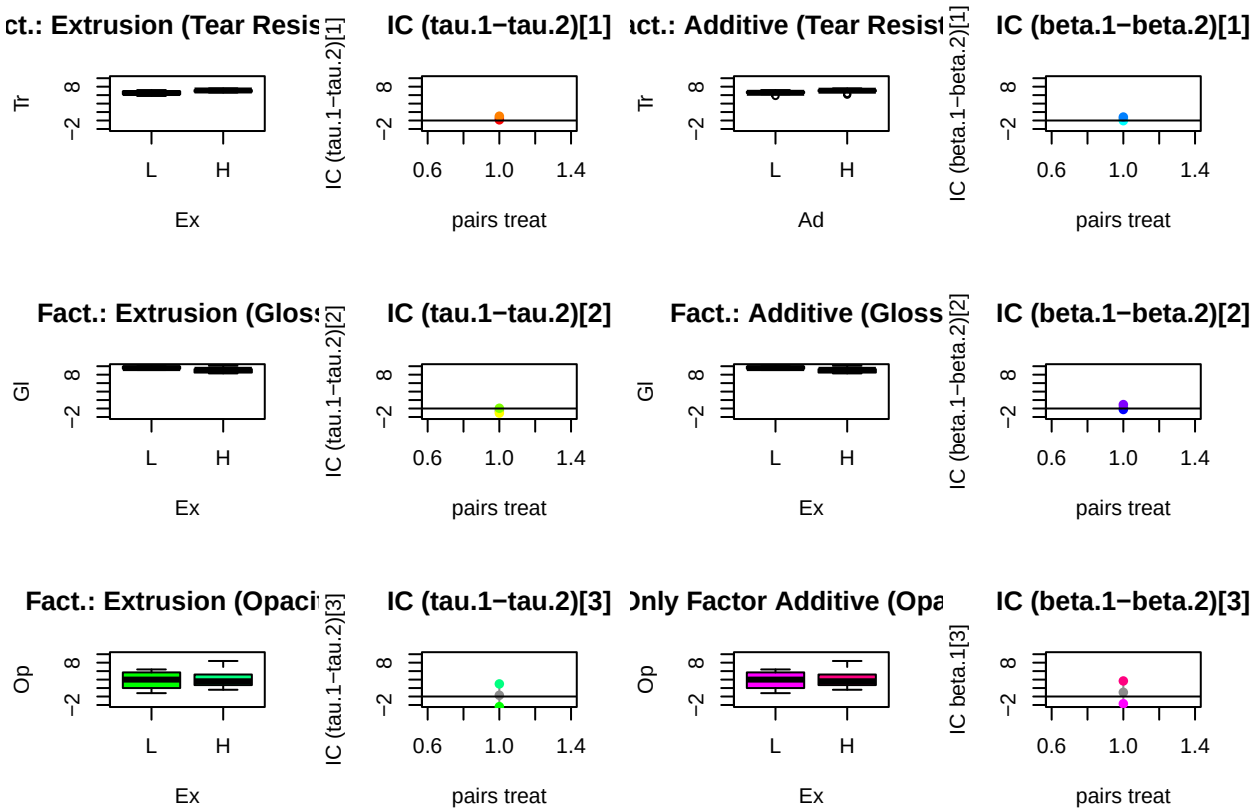
```
#! But from anova we set k=3 studying just the three cases that actually
#! look significant (well do it with care, is "data snooping" this!)

#! CIs due to the effect of the different combinations of the two treatments
qT = qt(1 - alpha / (2 * k), g*b*n-g-b+1)
# the degrees of freedom of the residuals on the additive model are
# g*b*n-g-b+1

mExL  = sapply(plastic3[Ex=='L',],mean)
```

```r
mExH  = sapply(plastic3[Ex=='H',],mean)
#! The CIs are univariate, we select the different components with diag(W)
#! to select all the variances (scalar formula, but one-liner form)
infEx = mExH-mExL - qT * sqrt( diag(W)/(g*b*n-g-b+1) * (1/10+1/10) )
supEx = mExH-mExL + qT * sqrt( diag(W)/(g*b*n-g-b+1) * (1/10+1/10) )

mAdL  = sapply(plastic3[Ad=='L',],mean)
mAdH  = sapply(plastic3[Ad=='H',],mean)
infAd = mAdH-mAdL - qT * sqrt( diag(W)/(g*b*n-g-b+1) * (1/10+1/10) )
supAd = mAdH-mAdL + qT * sqrt( diag(W)/(g*b*n-g-b+1) * (1/10+1/10) )

IC2   = list(ExH_ExL=cbind(infEx, supEx), AdH_AdL=cbind(infAd, supAd))
IC2
```

```
## $ExH_ExL
##         infEx      supEx
## Tr  0.1600616 1.01993841
## Gl -1.0864959 0.06649592
## Op -2.3963103 2.97631028
##
## $AdH_AdL
##          infAd     supAd
## Tr -0.03993841 0.8199384
## Gl -0.22649592 0.9264959
## Op -1.69631028 3.6763103
```

```r
par(mfrow=c(3,4))
boxplot(plastic3[,1]~Ex,   main='Fact.: Extrusion (Tear Resistance)'  ,
        ylab='Tr', col=rainbow(2*6)[c(1,2)], ylim=c(-2,10))
plot(c(1,g*(g-1)/2),range(IC2[[1]][1,]), pch='',main='IC (tau.1-tau.2)[1]',
     xlab='pairs treat', ylab='IC (tau.1-tau.2)[1]', ylim=c(-2,10))
lines(c(1,1), c(IC2[[1]][1,1],IC2[[1]][1,2]), col='grey55');
points(1, (mExH-mExL)[1], pch=16, col='grey55');
points(1, IC2[[1]][1,1], col=rainbow(2*6)[1], pch=16);
points(1, IC2[[1]][1,2], col=rainbow(2*6)[2], pch=16);
abline(h=0)

boxplot(plastic3[,1]~Ad,   main='Fact.: Additive (Tear Resistance)'  ,
        ylab='Tr', col=rainbow(2*6)[c(7,8)], ylim=c(-2,10))
plot(c(1,g*(g-1)/2),range(IC2[[2]][1,]), pch='',main='IC (beta.1-beta.2)[1]',
     xlab='pairs treat', ylab='IC (beta.1-beta.2)[1]', ylim=c(-2,10))
lines(c(1,1), c(IC2[[2]][1,1],IC2[[2]][1,2]), col='grey55');
points(1, (mAdH-mAdL)[1], pch=16, col='grey55');
points(1, IC2[[2]][1,1], col=rainbow(2*6)[7], pch=16);
points(1, IC2[[2]][1,2], col=rainbow(2*6)[8], pch=16);
abline(h=0)

boxplot(plastic3[,2]~Ex,   main='Fact.: Extrusion (Gloss)'  , ylab='Gl',
        col=rainbow(2*6)[c(3,4)], ylim=c(-2,10))
plot(c(1,g*(g-1)/2),range(IC2[[1]][2,]), pch='',main='IC (tau.1-tau.2)[2]',
     xlab='pairs treat', ylab='IC (tau.1-tau.2)[2]', ylim=c(-2,10))
lines(c(1,1), c(IC2[[1]][2,1],IC2[[1]][2,2]), col='grey55');
points(1, (mExH-mExL)[2], pch=16, col='grey55');
points(1, IC2[[1]][2,1], col=rainbow(2*6)[3], pch=16);
points(1, IC2[[1]][2,2], col=rainbow(2*6)[4], pch=16);
abline(h=0)

boxplot(plastic3[,2]~Ex,   main='Fact.: Additive (Gloss)'  , ylab='Gl',
        col=rainbow(2*6)[c(9,10)], ylim=c(-2,10))
plot(c(1,g*(g-1)/2),range(IC2[[2]][2,]), pch='',main='IC (beta.1-beta.2)[2]',
     xlab='pairs treat', ylab='IC (beta.1-beta.2)[2]', ylim=c(-2,10))
lines(c(1,1), c(IC2[[2]][2,1],IC2[[2]][2,2]), col='grey55');
points(1, (mAdH-mAdL)[2], pch=16, col='grey55');
points(1, IC2[[2]][2,1], col=rainbow(2*6)[9], pch=16);
points(1, IC2[[2]][2,2], col=rainbow(2*6)[10], pch=16);
abline(h=0)

boxplot(plastic3[,3]~Ex,   main='Fact.: Extrusion (Opacity)'  , ylab='Op',
        col=rainbow(2*6)[c(5,6)], ylim=c(-2,10))
plot(c(1,g*(g-1)/2),range(IC2[[1]][3,]), pch='',main='IC (tau.1-tau.2)[3]',
     xlab='pairs treat', ylab='IC (tau.1-tau.2)[3]', ylim=c(-2,10))
lines(c(1,1), c(IC2[[1]][3,1],IC2[[1]][3,2]), col='grey55');
points(1, (mExH-mExL)[3], pch=16, col='grey55');
points(1, IC2[[1]][3,1], col=rainbow(2*6)[5], pch=16);
```

```
points(1, IC2[[1]][3,2], col=rainbow(2*6)[6], pch=16);
abline(h=0)

boxplot(plastic3[,3]~Ex,   main='Only Factor Additive (Opacity)'  ,
        ylab='Op', col=rainbow(2*6)[c(11,12)], ylim=c(-2,10))
plot(c(1,g*(g-1)/2),range(IC2[[2]][3,]), pch='',main='IC (beta.1-beta.2)[3]',
     xlab='pairs treat', ylab='IC beta.1[3]', ylim=c(-2,10))
lines(c(1,1), c(IC2[[2]][3,1],IC2[[2]][3,2]), col='grey55');
points(1, (mAdH-mAdL)[3], pch=16, col='grey55');
points(1, IC2[[2]][3,1], col=rainbow(2*6)[11], pch=16);
points(1, IC2[[2]][3,2], col=rainbow(2*6)[12], pch=16);
abline(h=0)
```



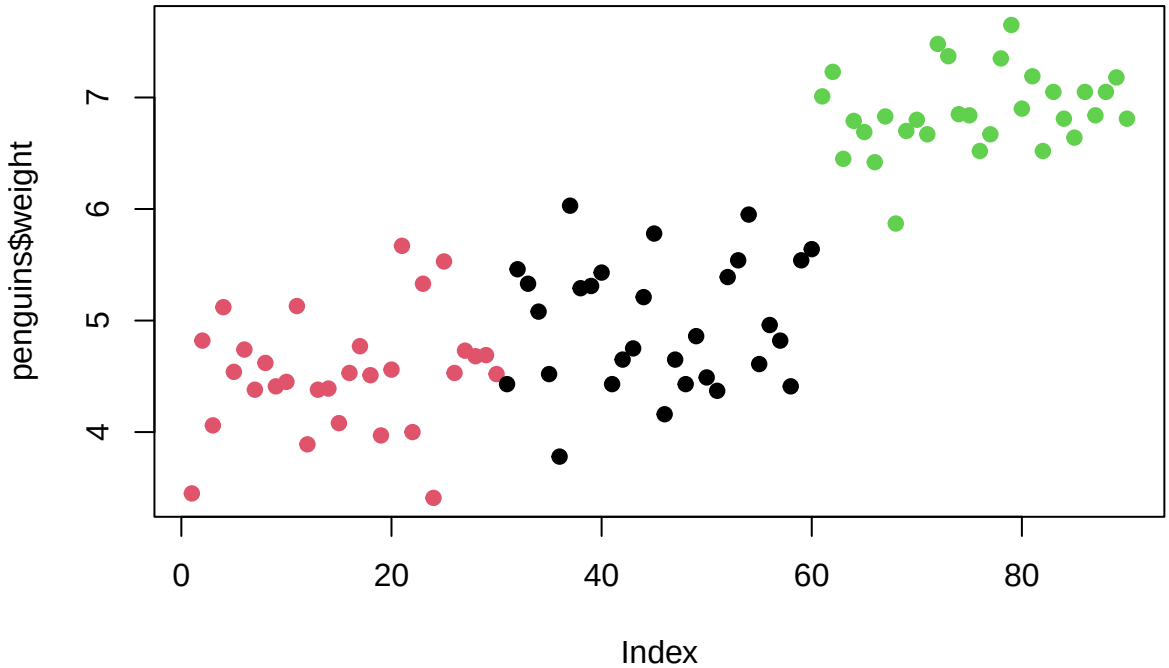
6.5 Exam exercise

```
## Problem 3 of 18/02/09
#####-----------------------
# During the austral summer three species of penguins nest in the Antarctic
# Peninsula: Chinstrap, Adelie and Gentoo.
# Some biologists of the Artowski basis measured the weight [kg] of 90 adults:
# 15 males and 15 females for each of the three species (penguins.txt file).
# a) By using an ANOVA model with two factors, claim if gender and / or species
#    belonging, significantly affect the weight.
# b) Using an appropriate model (possibly reduced), provide estimates (global
#    90% confidence) of means and variances of the groups identified at point
#    (a).

#! In this problem we cant talk of causality (in the previous one yes),
#! just association
penguins = read.table('Lab06/penguins.txt', header=T)
head(penguins)
```

```
##   weight   species gender
## 1   3.45 Chinstrap      M
## 2   4.82 Chinstrap      M
## 3   4.06 Chinstrap      M
## 4   5.12 Chinstrap      M
```

```
## 5   4.54 Chinstrap      M
## 6   4.74 Chinstrap      M
```

```
attach(penguins)

cols=rep(0,length(penguins$species))
cols=cols+as.numeric(species=="Chinstrap")*1
cols=cols+as.numeric(species=="Adelia")*2
cols=cols+as.numeric(species=="Gentoo")*3
plot(penguins$weight, col=as.factor(penguins$species),pch=19)
```



6.5.1 Qa

```
# a) By using an ANOVA model with two factors, claim if gender and / or species
#    belonging, significantly affect the weight.

# Model with interaction (complete model):
# X.ijk = mu + tau.i + beta.j + gamma.ij + eps.ijk; eps.ijk~N(0,sigma^2),
#     i=1,2 (effect gender), j=1,2,3 (effect species)

#! anova as we have just one target
fit1 = aov(weight ~ gender + species + species:gender, penguins)
summary(fit1)
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## gender          1   0.14    0.14   0.573  0.451
## species         2  92.98   46.49 191.310 <2e-16 ***
## gender:species  2   0.41    0.21   0.849  0.432
## Residuals      84  20.41    0.24
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Model without interaction (additive model):
# X.ijk = mu + tau.i + beta.j + eps.ijk; eps.ijk~N(0,sigma^2),
#     i=1,2 (effect gender), j=1,2,3 (effect species)
fit2 = aov(weight ~ gender + species, penguins)
summary(fit2)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## gender       1   0.14    0.14   0.575   0.45
## species      2  92.98   46.49 191.986 <2e-16 ***
## Residuals   86  20.83    0.24
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#! So we remove also the gender
# Reduced model (one-way):
# X.jk = mu + beta.j + eps.jk; eps.jk~N(0,sigma^2),
#     j=1,2,3 (effect species)
fit3 = aov(weight ~ species, penguins)
summary(fit3)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## species      2  92.98   46.49   192.9 <2e-16 ***
## Residuals   87  20.97    0.24
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#! Answer: just the species seems to be important, not the species

# verify assumptions (on the last model):
# 1) normality
library(MVN)
Ps = c( shapiro.test(t(weight[species==unique(species)[1]]))$p,
        shapiro.test(t(weight[species==unique(species)[2]]))$p,
        shapiro.test(t(weight[species==unique(species)[3]]))$p)
Ps
```

```
## [1] 0.3456878 0.4721578 0.5124616
```

6.5.2 Qb

```
# b) Using an appropriate model (possibly reduced), provide estimates (global
#    90% confidence) of means and variances of the groups identified at point
#    (a).bartlett.test(weight, species)

DF = fit3$df # n*g*b - 1 - (g-1) = 15*3*2-1-2 = 90-3 = 87
Spooled = sum(fit3$res^2)/DF

#! We want CIs for the effects, and for the variances, with a global
#! fixed level
means = as.vector(tapply(penguins$weight, penguins$species, mean))
names(means) = unique(species)
means
```

```
## Chinstrap    Adelia    Gentoo
##  4.976667  4.529667  6.874333
```

```
alpha = 0.10
k     = 4 # g + 1 = 4 (g Conf Int for the means and 1 for the variance)
BF    = rbind(cbind(means - sqrt(Spooled / 30) * qt(1 - alpha / (2*k), DF),
                    means + sqrt(Spooled / 30) * qt(1 - alpha / (2*k), DF)),
               c(Spooled * DF / qchisq(1 - alpha / (2*k), DF),
                 Spooled * DF / qchisq(alpha / (2*k), DF)))
rownames(BF)[4] = 'Var.'
colnames(BF) = c("inf", "sup")
BF
```

```
##                 inf       sup
## Chinstrap 4.7722444 5.1810889
## Adelia    4.3252444 4.7340889
## Gentoo    6.6699111 7.0787556
## Var.      0.1758739 0.3485624
```

```
#! 90% is the global confidence, so for all those four CIs

detach(penguins)
#! next lab: friday next next
```

7 Lab 07 - Classification (LDA, QDA, FDA, KNN, SVM)

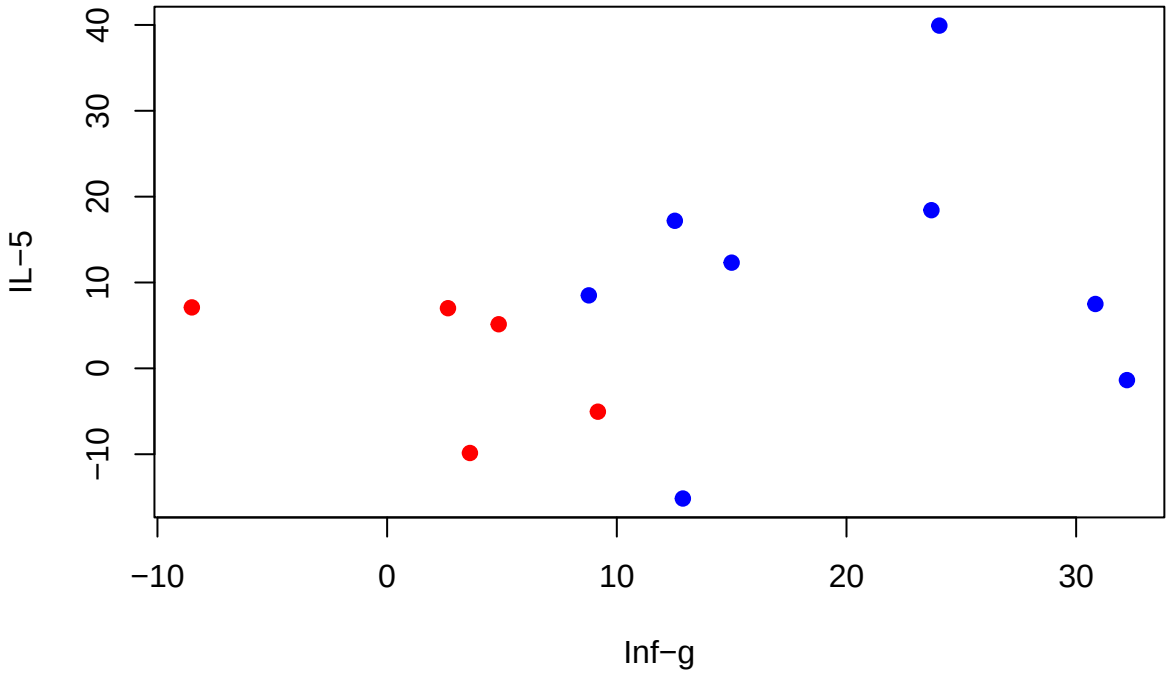
```
###--------------------###
### LAB 7 (14/04/2023) ###
###--------------------###

### TOPICS:
### Linear and Quadratic Discriminant Analysis
### Support Vector Machines
```

7.1 On 2 classes, univariate

```
#_____________________________________________________________________________________
### Example 1 (LDA with 2 classes, univariate)
###----------------------------------
cyto = read.table('Lab07/cytokines.txt', header=T)
cyto
```

```
##         Infg        IL5 group
## 1  15.000000  12.312595     A
## 2  23.698101  18.422812     A
## 3   8.781799   8.501765     A
## 4  32.211799  -1.371131     A
## 5  30.838288   7.496666     A
## 6  12.876335 -15.159156     A
## 7  12.526771  17.187205     A
## 8  24.043710  39.919920     A
## 9  -8.502941   7.099296     B
## 10  9.176056  -5.047772     B
## 11  4.857983   5.138093     B
## 12  2.649528   7.007139     B
## 13  3.605551  -9.859006     B
```

```
attach(cyto)

# Group A: favorable clinical outcome (the treatment has effect)
A = which(group == 'A')
# Group B: unfavorable clinical outcome (the treatment has no effect)
B = which(group == 'B')


plot(cyto[,1], cyto[,2], pch=19, col=c(rep('blue',8), rep('red',5)),
     xlab='Inf-g', ylab='IL-5')
```




```
#! We can see that a line at x=10 almost separate already the two classes


### Idea: we aim to find a "rule" to classify patients as Group A or Group B
### given the measurements of Inf-g (Interferon gamma) and IL-5 (Interleukin 5).
### For this example, we discard the IL-5 variable (which seems to have poor
### "discriminative power") and we consider the univariate case with Inf-g only
```

7.1.1 Verify assumptions

```
### LDA (univariate)
###------------------
# Assumptions:
# 1) if L = i, X.i ~ N(mu.i, sigma.i^2), i = A,B
# 2) sigma.A = sigma.B
# 3) c(A|B) = c(B|A) (equal misclassification costs)


# verify assumptions 1) and 2):
# 1) normality (univariate) within the groups
shapiro.test(cyto[A,1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cyto[A, 1]
## W = 0.90684, p-value = 0.3323
```

```
shapiro.test(cyto[B,1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cyto[B, 1]
## W = 0.87547, p-value = 0.2893
```

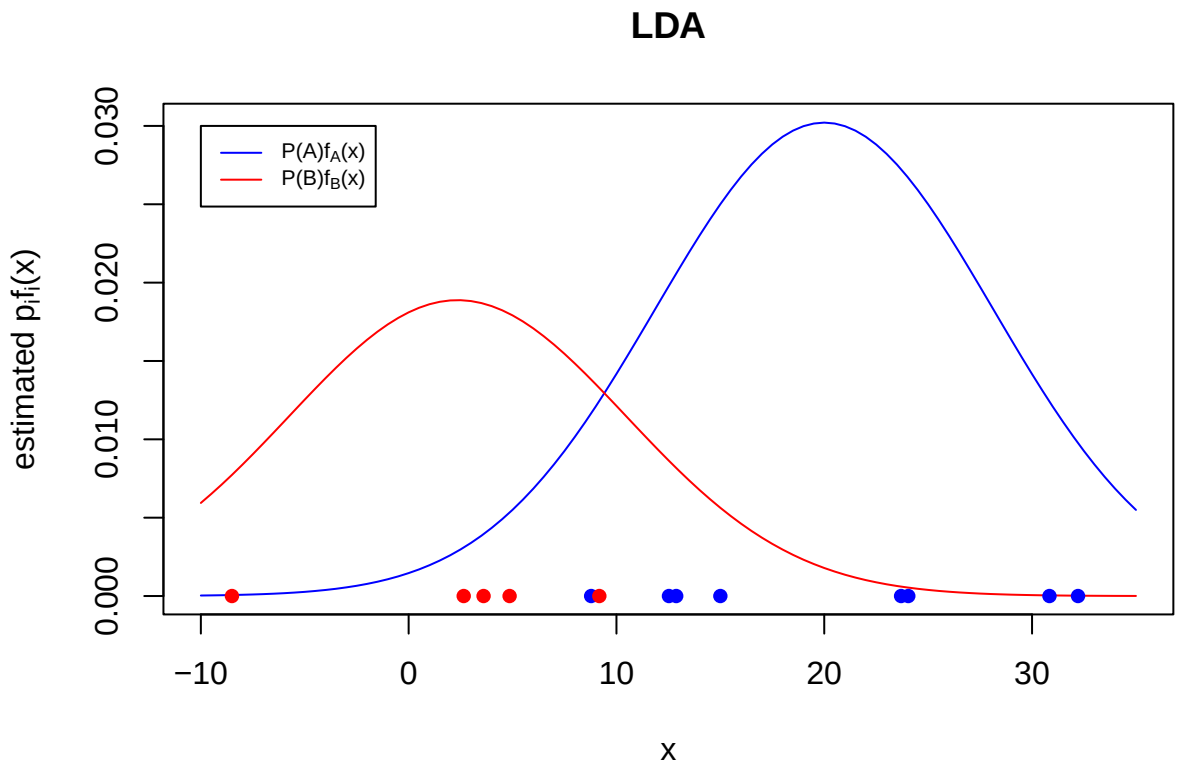
```r
# -> OK
#! To see a occhio se possono essere gaussiane meglio qqplot che hist

# 2) equal variance (univariate)
# var.test is used instead of bartlett.test when there are only 2 groups
var.test(cyto[A,1], cyto[B,1])
```

```
##
##  F test to compare two variances
##
## data:  cyto[A, 1] and cyto[B, 1]
## F = 1.8372, num df = 7, denom df = 4, p-value = 0.5812
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##   0.2024605 10.1458619
## sample estimates:
## ratio of variances
##           1.837155
```

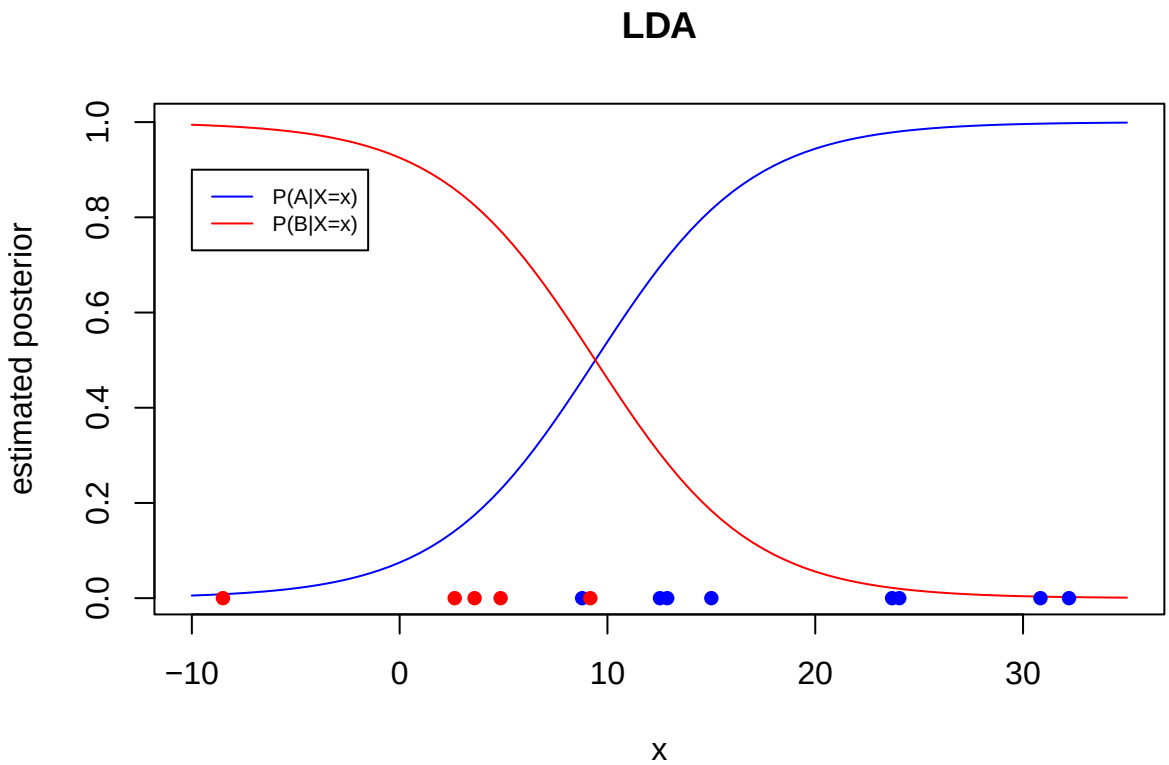
```r
# -> OK
#! So for only two-variance-comparison use this which is more precise

nA = length(A)
nB = length(B)
n  = nA + nB

# Prior probabilities (estimated from the data, no prior knowledge)
PA = nA / n
PB = nB / n
PA+PB == 1
```

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

```r
#! Changing the prior will just change the intercept, so in a more general setting
#! we can exploit this fact.
#! And we can change them according to how our dataset was collected or if we have
#! more informations (there is an unbalance in the groups? reality is different? ecc)
#! Like there is a "bias of selection" and so we can fix it

### Recall:
# the classification region is obtained by comparing pA*f.A and pB*f.B
#!                                                    prior*likelihood
MA = mean(Infg[A])
MB = mean(Infg[B])

SA = var(Infg[A])
SB = var(Infg[B])
S  = ((nA-1) * SA + (nB-1) * SB) / (nA + nB - 2)  # pooled estimate

x = seq(-10, 35, 0.5) # include the range of Infg

# par(mfrow=c(2,1))
plot(x, PA * dnorm(x, MA, sqrt(S)), type='l', col='blue',
     ylab=expression(paste('estimated ', p[i] * f[i],'(x)')), main='LDA')

points(x, PB * dnorm(x, MB, sqrt(S)), type='l', col='red')
points(Infg[A], rep(0, length(A)), pch=16, col='blue')
points(Infg[B], rep(0, length(B)), pch=16, col='red')

legend(-10, 0.03, legend=c(expression(paste('P(A)',f[A],'(x)')),
                           expression(paste('P(B)',f[B],'(x)'))),
       col=c('blue','red'), lty=1, cex = 0.7)
```




```
plot(x,
     PA * dnorm(x, MA, sqrt(S)) / (PA * dnorm(x, MA, sqrt(S)) + PB * dnorm(x, MB, sqrt(S))),
     type='l', col='blue', ylab='estimated posterior', main='LDA')

points(x,
       PB * dnorm(x, MB, sqrt(S)) / (PA * dnorm(x, MA, sqrt(S)) + PB * dnorm(x, MB, sqrt(S))),
       type='l', col='red')

points(Infg[A], rep(0, length(A)), pch=16, col='blue')
points(Infg[B], rep(0, length(B)), pch=16, col='red')

legend(-10, 0.9, legend=c('P(A|X=x)', 'P(B|X=x)'), col=c('blue','red'), lty=1,
       cex = 0.7)
```




```
#! If we normalize the first plot we will get the posterior probabilities
#! And using different priors will make the "separating line" to move
#! And also it is the same plot we would get when performing logistic regression
```

7.1.2 LDA

```
### LDA with R
### ---------------------
library(MASS)
# help(lda)

# cyto.lda = lda(group ~ Infg, prior=c(PA, PB))
cyto.lda = lda(group ~ Infg)
# Note: if we don't specify the prior probabilities, they are estimated
# from the sample
cyto.lda
```

```
## Call:
## lda(group ~ Infg)
##
## Prior probabilities of groups:
##         A         B
## 0.6153846 0.3846154
##
## Group means:
##        Infg
## A 19.997100
## B  2.357236
##
## Coefficients of linear discriminants:
##            LD1
## Infg 0.1230572
```

```
#! The "coeffs of linear discriminants" is just one as we are in the univariate case
#! and we obtain just a vertical line then
```

```r
#! We try now to classify a "new point" at x=0
# posterior probability and classification for x=0
x = data.frame(Infg = 0)

# The command predict() returns a list containing (see the # help of predict.lda):
# - the class associated with the highest posterior probability
predict(cyto.lda, x)$class
```

```
## [1] B
## Levels: A B
```

```r
#! It says B, e in effetti dal grafico di prima x=0 era nella regione rossa di B
# - the posterior probabilities for the classes
predict(cyto.lda, x)$posterior
```

```
##            A         B
## 1 0.07476761 0.9252324
```

```r
# - in lda: the coordinates of the canonical analysis of Fisher
# (Fisher's discriminant scores)
predict(cyto.lda, x)$x
```

```
##         LD1
## 1 -1.625897
```

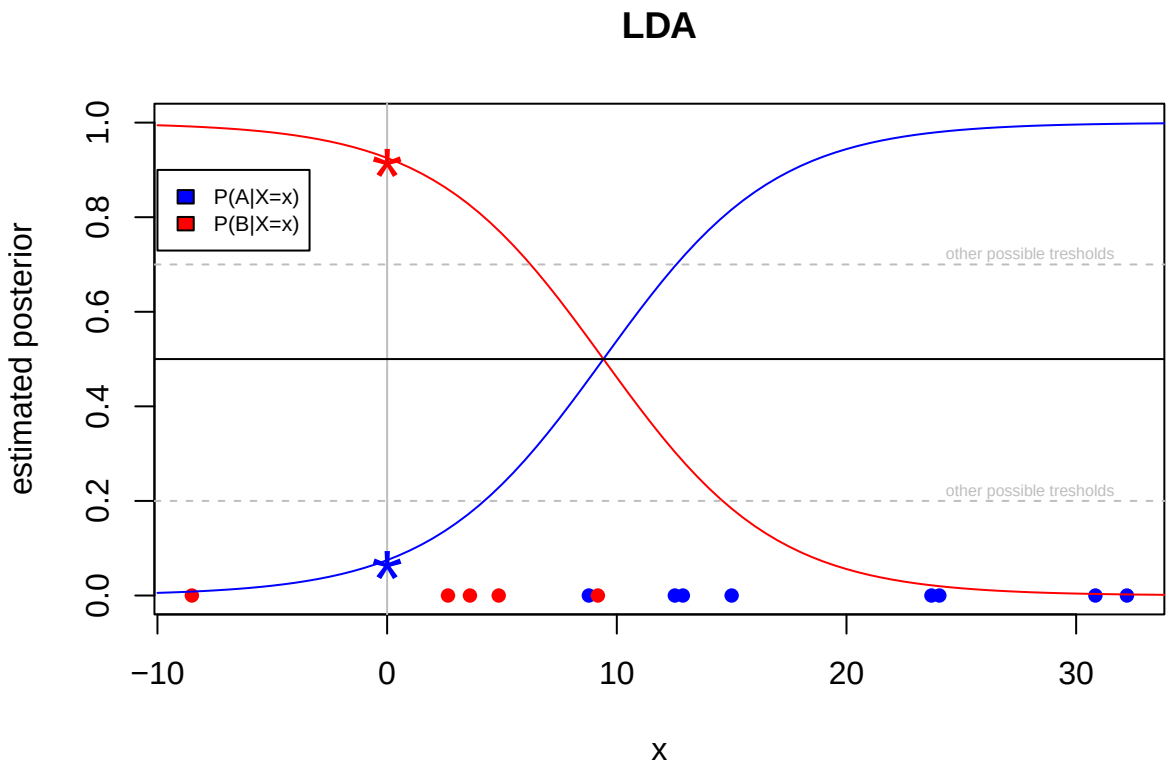
```r
#! A'x-mu, coordinates are centered as in PCA


plot(Infg[A], rep(0, length(A)), pch=16, col='blue', ylim=c(0,1),
     xlab='x', ylab='estimated posterior', main="LDA", xlim=range(Infg))
points(Infg[B], rep(0, length(B)), pch=16, col='red')
abline(v=0, col='grey')
points(c(0,0),c(predict(cyto.lda, data.frame(Infg = 0))$posterior),
       col=c('blue', 'red'), pch='*', cex=2.5)

# posterior probability for a grid of x's
x = data.frame(Infg=seq(-10, 35, 0.5))

head(predict(cyto.lda, x)$posterior)
```

```
##             A         B
## 1 0.005558367 0.9944416
## 2 0.006347566 0.9936524
## 3 0.007248000 0.9927520
## 4 0.008275102 0.9917249
## 5 0.009446369 0.9905536
## 6 0.010781615 0.9892184
```

```r
cyto.LDA.A = predict(cyto.lda, x)$posterior[,1] # posterior probability for class A
cyto.LDA.B = predict(cyto.lda, x)$posterior[,2] # posterior probability for class B

predict(cyto.lda, x)$class
```

```
##  [1] B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
## [39] B A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
## [77] A A A A A A A A A A A A A A
## Levels: A B
```

```r
lines(x[,1], cyto.LDA.A, type='l', col='blue', xlab='x', ylab='estimated posterior',
      main="LDA")
lines(x[,1], cyto.LDA.B, type='l', col='red')
abline(h = 0.5)
#! We can control the cost of misclassification by moving this line above or below
#! Like to be sure when we assing obs to certain groups
abline(h=0.7, col="gray",lty="dashed")
abline(h=0.2, col="gray",lty="dashed")
text(x=28,y=0.22,"other possible tresholds",cex=0.5, col="gray")
text(x=28,y=0.72,"other possible tresholds",cex=0.5, col="gray")

legend(-10, 0.9, legend=c('P(A|X=x)', 'P(B|X=x)'), fill=c('blue','red'), cex = 0.7)
```




```
# set prior probabilities
# Pay attention to the order in which the levels appear in factor(group)!
cyto.lda.1 = lda(group ~ Infg, prior=c(0.95,0.05))
#! The priors are specified in the order of the factor levels
factor(cyto$group)
```

```
##  [1] A A A A A A A A B B B B B
## Levels: A B
```

```
unique(cyto$group)
```

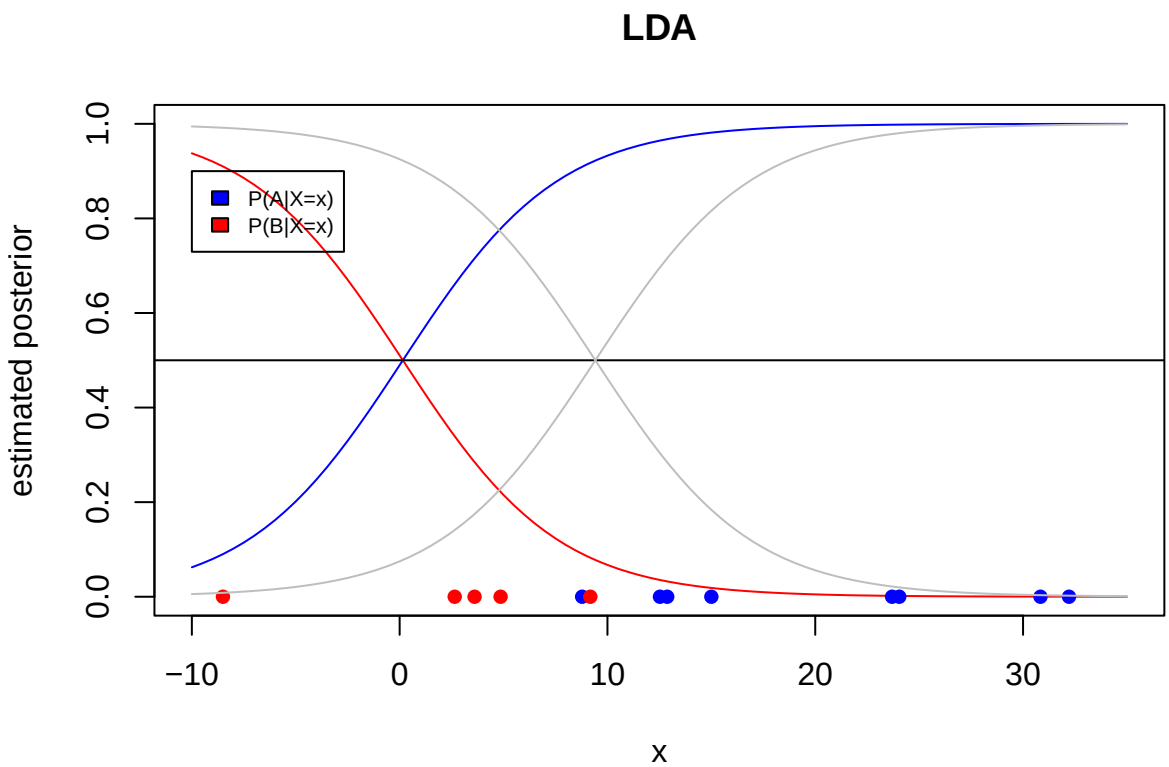
```
## [1] "A" "B"
```

```
cyto.lda.1
```

```
## Call:
## lda(group ~ Infg, prior = c(0.95, 0.05))
##
## Prior probabilities of groups:
##    A    B
## 0.95 0.05
##
## Group means:
##        Infg
## A 19.997100
## B  2.357236
##
## Coefficients of linear discriminants:
##            LD1
## Infg 0.1230572
```

```
x = data.frame(Infg=seq(-10, 35, 0.5))

cyto.LDA.A.1 = predict(cyto.lda.1, x)$posterior[,1] # posterior probability for class A
cyto.LDA.B.1 = predict(cyto.lda.1, x)$posterior[,2] # posterior probability for class B
```

```
plot  (x[,1], cyto.LDA.A.1, type='l', col='blue', xlab='x', ylab='estimated posterior',
       main="LDA", ylim=c(0,1))
points(x[,1], cyto.LDA.B.1, type='l', col='red')
abline(h = 0.5)
legend(-10, 0.9, legend=c('P(A|X=x)', 'P(B|X=x)'), fill=c('blue','red'), cex = 0.7)
points(Infg[A], rep(0, length(A)), pch=16, col='blue')
points(Infg[B], rep(0, length(B)), pch=16, col='red')

points(x[,1], cyto.LDA.A, type='l', col='grey')
points(x[,1], cyto.LDA.B, type='l', col='grey')
```




```
#! B is red group. So with those priors we want "much more proof" to say that
#! an obs is red. In fact priors were 95% defaulted to A
```

7.1.3 KNN

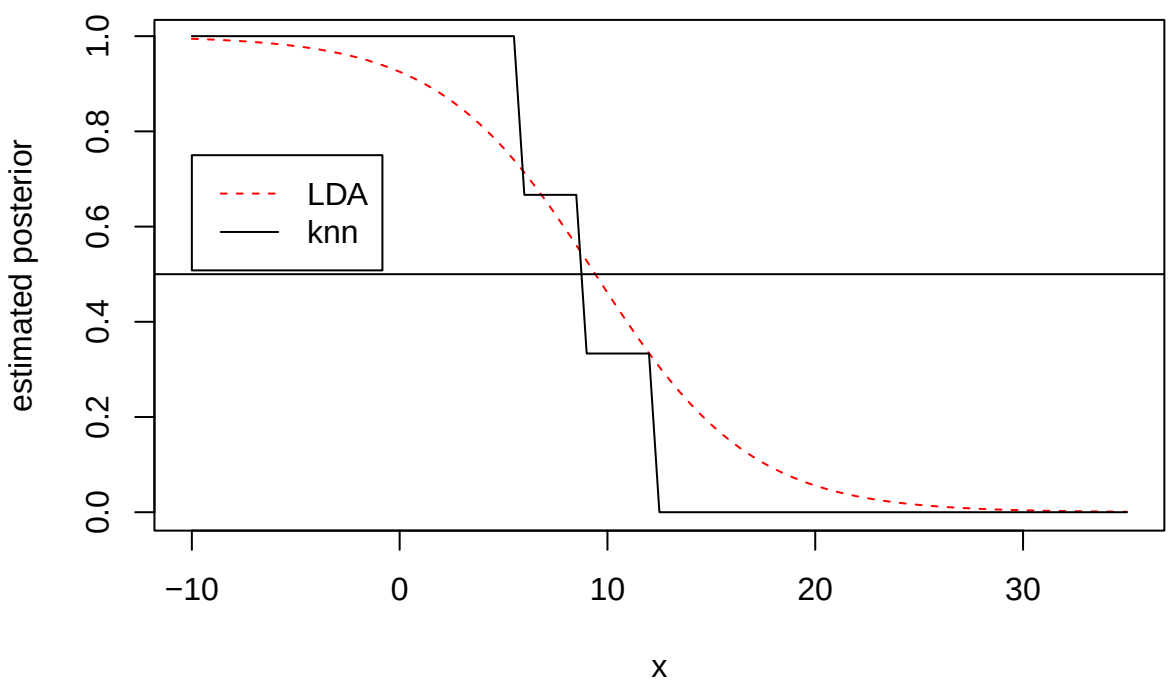
```
### k-nearest neighbor classifier
###-------------------------------

library(class)
# help(knn)
#! Very simple and intuitive to implement, and can be done on any kind of data
#! as long as we can define on them a metric, or well just a similarity function
#! But - everything depends on that choice for the dissimilarity
#!     - and there is the curse of dimensionality

#! (1) A common function: the edit one, how many steps are required to go from
#! one object to another. But is very computationally heavy.
#! (2) So maybe we can think of mapping our objects in R^p, in some way, and then
#! use the natural euclidean distance

Infg #! train
```

```
##  [1] 15.000000 23.698101  8.781799 32.211799 30.838288 12.876335 12.526771
##  [8] 24.043710 -8.502941  9.176056  4.857983  2.649528  3.605551
```

```r
x$Infg #! test
```

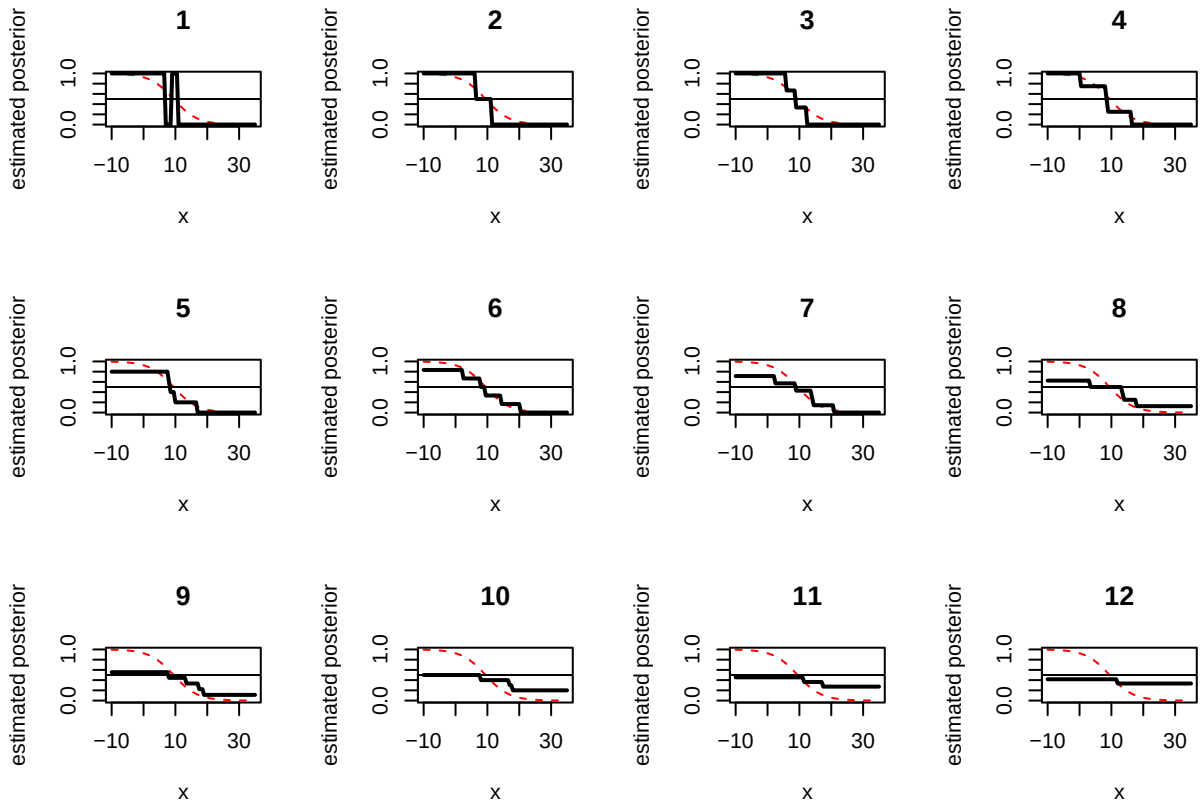
```
##  [1] -10.0  -9.5  -9.0  -8.5  -8.0  -7.5  -7.0  -6.5  -6.0  -5.5  -5.0  -4.5
## [13]  -4.0  -3.5  -3.0  -2.5  -2.0  -1.5  -1.0  -0.5   0.0   0.5   1.0   1.5
## [25]   2.0   2.5   3.0   3.5   4.0   4.5   5.0   5.5   6.0   6.5   7.0   7.5
## [37]   8.0   8.5   9.0   9.5  10.0  10.5  11.0  11.5  12.0  12.5  13.0  13.5
## [49]  14.0  14.5  15.0  15.5  16.0  16.5  17.0  17.5  18.0  18.5  19.0  19.5
## [61]  20.0  20.5  21.0  21.5  22.0  22.5  23.0  23.5  24.0  24.5  25.0  25.5
## [73]  26.0  26.5  27.0  27.5  28.0  28.5  29.0  29.5  30.0  30.5  31.0  31.5
## [85]  32.0  32.5  33.0  33.5  34.0  34.5  35.0
```

```r
#! We cant choice the distance function however, with this package
cyto.knn = knn(train = Infg, test = x, cl = group, k = 3, prob=T)
#! train data, test data,
#!      number of neighbours for the choice, and posterior probs for classifying
cyto.knn.class = cyto.knn == 'B'
cyto.knn.B = ifelse(cyto.knn.class==1,
                    attributes(cyto.knn)$prob,
                    1 - attributes(cyto.knn)$prob)


plot(x[,1], cyto.LDA.B, type='l', col='red', lty=2, xlab='x', ylab='estimated posterior')
points(x[,1], cyto.knn.B, type='l', col='black', lty=1)
abline(h = 0.5)
legend(-10, 0.75, legend=c('LDA','knn'), lty=c(2,1), col=c('red','black'))
```




```r
#! The steps are because we have 2 neighbours of one type, and 1 of the other
#! (we choose k=3 in the call of knn)

# let's change k
par(mfrow=c(3,4))
for(k in 1:12)
{
  cyto.knn = knn(train = Infg, test = x, cl = group, k = k, prob=T)
  cyto.knn.class = (cyto.knn == 'B')+0
  cyto.knn.B = ifelse(cyto.knn.class==1, attributes(cyto.knn)$prob,
                      1 - attributes(cyto.knn)$prob)
```

```
  plot(x[,1], cyto.LDA.B, type='l', col='red', lty=2, xlab='x',
       ylab='estimated posterior', main=k)
  points(x[,1], cyto.knn.B, type='l', col='black', lty=1, lwd=2)
  abline(h = 0.5)
}
```




```
#! k=1 will too much fit the noise, while k=6 seems nice, really suits the lda
#! But the actual choice can be done with also the # help of cross validation, doing
#! tests on different values
#! Anyway this plot can be done only in the univariate case

#! WARNING: ok tune parameters ecc, but the test set should be used really at the end
#! the moment before delivering our model. Otherwise we will tune to it, overfit it.
#! Cioè staremmo aggiustando il modello sul test set, but so we will be adapting to it
#! e non va bene. So tests set should be used only once.

detach(cyto)
```

7.2 On 3 classes, bivariate

```
#_____________________________________________________________________________
### Example 2 (3 classes, bivariate)
###-------------------------------------------

# help(iris)

### We consider only the first two variables, Sepal.Length and Sepal.Width
### (p=2 features, g=3 groups); we aim to build a classifier based on the
### characteristic of the sepal that identifies the iris species.

attach(iris)

species.name = factor(Species)

g = 3
```

```r
i1 = which(species.name == 'setosa')
i2 = which(species.name == 'versicolor')
i3 = which(species.name == 'virginica')

n1 = length(i1)
n2 = length(i2)
n3 = length(i3)
n = n1 + n2 + n3

detach(iris)
```

7.2.1 Jittering

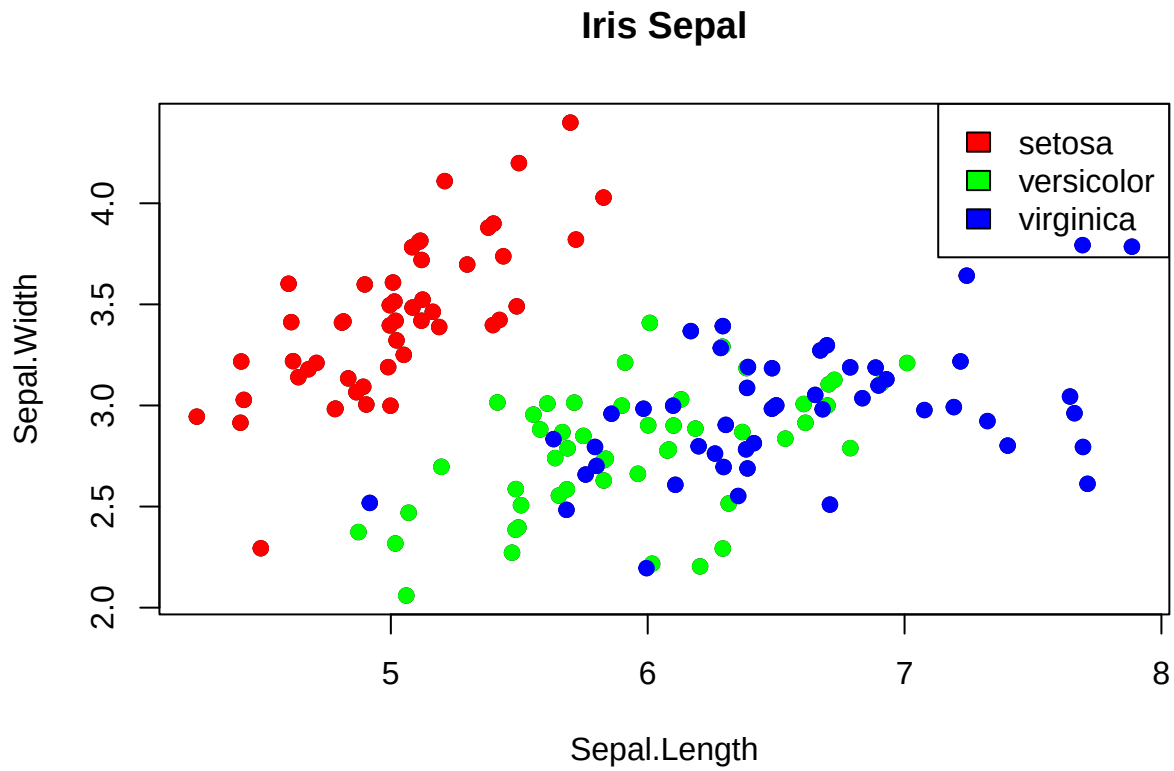
```r
iris2 = iris[,1:2]
#! unique on a dataset removes duplicate lines
dim(unique(iris2))
```

```
## [1] 117   2
```

```r
dim(iris2)
```

```
## [1] 150   2
```

```r
#! We have many duplicates, and so on plots some points will overlap
#! We can fix this by Jittering, adding a small noise to every data point
#! to recover all data to be different.
#! In general is useful to do jittering on our data, to see if our analysis
#! is robust. Ie if it works also on the new small perturbed data, or it is fragile
#! and as soon as we change data it doesnt work anymore.
#! To understand how much trust the data. As our analysis are always data driven
set.seed(1)
iris2 = iris2 + cbind(rnorm(150, sd=0.025)) # jittering

# plot the data

# cols = hcl.colors(3, palette = hcl.pals()[8])
cols = c("red","green","blue")
plot(iris2, main='Iris Sepal', xlab='Sepal.Length', ylab='Sepal.Width', pch=19)
points(iris2[i1,], col=cols[1], pch=19)
points(iris2[i2,], col=cols[2], pch=19)
points(iris2[i3,], col=cols[3], pch=19)
legend("topright", legend=levels(species.name), fill=c(cols))
```




```
#! It seems there is a difference in red and others. But the others two
#! instead seems closer

m =  colMeans(iris2)
m1 = colMeans(iris2[i1,])
m2 = colMeans(iris2[i2,])
m3 = colMeans(iris2[i3,])

S1 = cov(iris2[i1,])
S2 = cov(iris2[i2,])
S3 = cov(iris2[i3,])
Sp  = ((n1-1)*S1+(n2-1)*S2+(n3-1)*S3) / (n-g)
```

7.2.2 MANOVA to check features difference

```
#! manova to see if the distribution of the features is different in the
#! different groups. Otherwise we wont be able to build the classifier

# One-way MANOVA (See LAB 6)
#! We should check the assumptions
fit = manova(as.matrix(iris2) ~ species.name)
summary.manova(fit, test="Wilks")
```

```
##               Df   Wilks approx F num Df den Df    Pr(>F)
## species.name   2 0.16695   105.66      4    292 < 2.2e-16 ***
## Residuals    147
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#! pvalue small -> we reject H0, ie there are differences
```

7.2.3 LDA

```r
# Linear Discriminant Analysis (LDA)
#! We should check the assumptions, but by the scatterplot all seems legit
#! "data approx normal, and covariance similar"
lda.iris = lda(iris2, species.name)
lda.iris
```

```
## Call:
## lda(iris2, species.name)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width
## setosa         5.008511    3.430511
## versicolor     5.938933    2.772933
## virginica      6.584188    2.970188
##
## Coefficients of linear discriminants:
##                    LD1        LD2
## Sepal.Length -2.137114 -0.8197481
## Sepal.Width   2.789529 -2.0844258
##
## Proportion of trace:
##    LD1    LD2
## 0.9644 0.0356
```

```r
#! "Coefficients of linear discriminants" as separating regions we have lines,
#! so two coefficients
#! "Proportion of trace" refers to eigvals, as LDA is derived from the spectral
#! decomposition of the covariance (?), lambda/sum of lambda
#! Ie the percentage of separation which is captured by the linear discriminant

#! Like the first is high as it should be the direction SE->NO, and from the plot
#! yes it seems to be really capturing differences in the species
#! While the second, NE->SO doesnt seem really discriminant (and in fact second
#! value is low)


# "coefficients of linear discriminants" and "proportion of trace":
# Fisher discriminant analysis.
# In particular:
# - coefficients of linear discriminants: versors of the canonical directions
#   [to be read column-wise]
# - proportion of trace: proportion of variance (between wrt to within)
#   explained by the corresponding canonical direction

Lda.iris = predict(lda.iris, iris2)
names(Lda.iris)
```

```
## [1] "class"     "posterior" "x"
```

7.2.4 Computation of AER and APER

```r
# Estimate of AER (actual error rate):
# 1) APER (apparent error rate)
# 2) estimate of AER by cross-validation
```

```r
# 1) Compute the APER
Lda.iris$class   # assigned classes
```

7.2.4.1 APER

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
```

```
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     versicolor
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     virginica  virginica  virginica  versicolor
##  [55] virginica  versicolor versicolor versicolor virginica  versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor virginica
##  [67] versicolor versicolor virginica  versicolor versicolor versicolor
##  [73] virginica  versicolor virginica  virginica  virginica  virginica
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor virginica  virginica  versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor versicolor versicolor
## [103] virginica  virginica  virginica  virginica  versicolor virginica
## [109] virginica  virginica  virginica  virginica  virginica  versicolor
## [115] versicolor virginica  virginica  virginica  virginica  versicolor
## [121] virginica  versicolor virginica  virginica  virginica  virginica
## [127] versicolor versicolor virginica  virginica  virginica  virginica
## [133] virginica  virginica  versicolor virginica  versicolor virginica
## [139] versicolor virginica  virginica  virginica  versicolor virginica
## [145] virginica  virginica  virginica  virginica  versicolor versicolor
## Levels: setosa versicolor virginica
```

```
species.name     # true labels
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
table(class.true=species.name, class.assigned=Lda.iris$class)
```

```
##             class.assigned
## class.true   setosa versicolor virginica
##   setosa         49          1         0
##   versicolor      0         36        14
##   virginica       0         15        35
```

```
#! "confusion matrix", diagonal elements are the right ones
#! All the other outside diagonal are misclassified

errors = (Lda.iris$class != species.name)
sum(errors) #! = 15+15+1
```

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

```
length(species.name)
```

```
## [1] 150
```

```r
#! sum(el outside diag) / all elements
(1+15+15)/150
```

```
## [1] 0.2066667
```

```r
APER   = sum(errors)/length(species.name)
APER
```

```
## [1] 0.2
```

```r
#! The APER estimation is valid only if we used the computed priors from
#! the empirical ones observed in the data, ie:

# REMARK: this is correct only if we estimate the prior with the empirical
#! frequencies! Otherwise we have to attribute a proper wheight to the
#! confusion matrix, so

prior = c(1/3,1/3,1/3)
G = 3
misc = table(class.true=species.name, class.assigned=Lda.iris$class)
APER = 0
for(g in 1:G)
  APER = APER + sum(misc[g,-g])/sum(misc[g,]) * prior[g]
APER
```

```
## [1] 0.2
```

```r
# 2) Compute the estimate of the AER by leave-one-out cross-validation
# by hand LOO CV (leave one out cross validation):

errors_CV = 0
for(i in 1:150){
  #! we remove the i-th obs
  LdaCV.i = lda(iris2[-i,], species.name[-i], prior=c(50,50,50)/150)
  #! we compute the error
  errors_CV = errors_CV + as.numeric(predict(LdaCV.i,iris2[i,])$class != species.name[i])
}
errors_CV
```

7.2.4.2 AER

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

```r
AERCV   = sum(errors_CV)/length(species.name)
AERCV
```

```
## [1] 0.2
```

```r
#! It usually is higher than APER, as APER is more optimistic
```

```r
# with R:
LdaCV.iris = lda(iris2, species.name, CV=TRUE)  # specify the argument CV
LdaCV.iris$class
```

7.2.4.3 With R

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     versicolor
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     virginica  virginica  virginica  versicolor
##  [55] virginica  versicolor versicolor versicolor virginica  versicolor
```

```
##  [61] versicolor versicolor versicolor versicolor versicolor virginica
##  [67] versicolor versicolor virginica  versicolor versicolor versicolor
##  [73] virginica  versicolor virginica  virginica  virginica  virginica
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor virginica  virginica  versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor versicolor versicolor
## [103] virginica  virginica  virginica  virginica  versicolor virginica
## [109] virginica  virginica  virginica  virginica  virginica  versicolor
## [115] versicolor virginica  virginica  virginica  virginica  versicolor
## [121] virginica  versicolor virginica  virginica  virginica  virginica
## [127] versicolor versicolor virginica  virginica  virginica  virginica
## [133] virginica  virginica  versicolor virginica  versicolor virginica
## [139] versicolor virginica  virginica  virginica  versicolor virginica
## [145] virginica  virginica  virginica  virginica  versicolor versicolor
## Levels: setosa versicolor virginica
```

```
species.name
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
table(class.true=species.name, class.assignedCV=LdaCV.iris$class)
```

```
##             class.assignedCV
## class.true   setosa versicolor virginica
##   setosa         49          1         0
##   versicolor      0         36        14
##   virginica       0         15        35
```

```
errorsCV = (LdaCV.iris$class != species.name)
errorsCV
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE
##  [73]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
## [121] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [145] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

```r
sum(errorsCV)
```

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

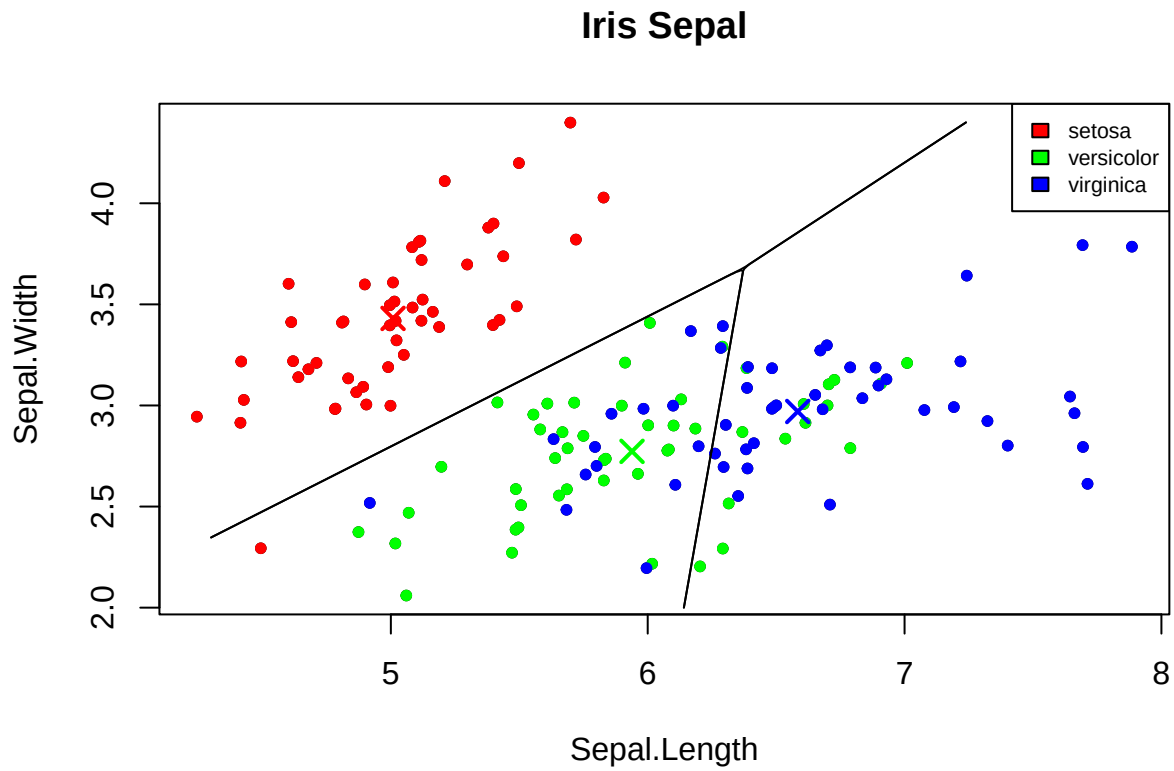
```r
AERCV   = sum(errorsCV)/length(species.name)
AERCV
```

```
## [1] 0.2
```

```r
# REMARK: correct only if we estimate the priors through the sample frequencies!
#! So the same as before, ie otherwise we have to wheight our computation
```

7.2.5 Partition generated by LDA

```r
# Plot the partition induced by LDA

plot(iris2, main='Iris Sepal', xlab='Sepal.Length', ylab='Sepal.Width', pch=20)
points(iris2[i1,], col='red', pch=20)
points(iris2[i2,], col='green', pch=20)
points(iris2[i3,], col='blue', pch=20)
legend("topright", legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

points(lda.iris$means, pch=4,col=c('red','green','blue'), lwd=2, cex=1.5)

x  = seq(min(iris[,1]), max(iris[,1]), length=200)
y  = seq(min(iris[,2]), max(iris[,2]), length=200)
xy = expand.grid(Sepal.Length=x, Sepal.Width=y)

z  = predict(lda.iris, xy)$post  # these are P_i*f_i(x,y)
#! Because we take the $post (posterior) probabilities
z1 = z[,1] - pmax(z[,2], z[,3])  # P_1*f_1(x,y)-max{P_j*f_j(x,y)}
z2 = z[,2] - pmax(z[,1], z[,3])  # P_2*f_2(x,y)-max{P_j*f_j(x,y)}
z3 = z[,3] - pmax(z[,1], z[,2])  # P_3*f_3(x,y)-max{P_j*f_j(x,y)}

# Plot the contour line of level (levels=0) of z1, z2, z3:
# P_i*f_i(x,y)-max{P_j*f_j(x,y)}=0 ie, boundary between R.i and R.j
# where j realizes the max.
contour(x, y, matrix(z1, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z2, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z3, 200), levels=0, drawlabels=F, add=T)
```




```
library(rgl)
library(mvtnorm)
open3d()
```

```
## wgl
##   1
```

```
points3d(iris2[i1,1], iris2[i1,2], 0, col='red', pch=15)
points3d(iris2[i2,1], iris2[i3,2], 0, col='green', pch=15)
points3d(iris2[i3,1], iris2[i2,2], 0, col='blue', pch=15)
surface3d(x, y, dmvnorm(xy, m1, Sp)/3, alpha=0.4, color='red')
surface3d(x, y, dmvnorm(xy, m2, Sp)/3, alpha=0.4, color='green', add=T)
surface3d(x, y, dmvnorm(xy, m3, Sp)/3, alpha=0.4, color='blue', add=T)
box3d()
```

7.2.6 QDA

```
### Quadratic Discriminant Analysis (QDA)
###----------------------------------------

# help(qda)
#! If we believe that the boundary between our groups is not linear,
#!    - dont use LDA (move like here to QDA)
#!    - or use LDA but with transformed/added variables
qda.iris = qda(iris2, species.name)
qda.iris
```

```
## Call:
## qda(iris2, species.name)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width
```

```
## setosa         5.008511    3.430511
## versicolor     5.938933    2.772933
## virginica      6.584188    2.970188
```

```
Qda.iris = predict(qda.iris, iris2)
```

7.2.7 Computation of AER and APER

```
# compute the APER
Qda.iris$class
```

7.2.7.1 APER

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     versicolor
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     virginica  virginica  virginica  versicolor
##  [55] virginica  versicolor virginica  versicolor virginica  versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor virginica
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor virginica  virginica  virginica  virginica
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor virginica  virginica  versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  versicolor
## [103] virginica  versicolor virginica  virginica  versicolor virginica
## [109] virginica  virginica  virginica  virginica  virginica  versicolor
## [115] versicolor virginica  virginica  virginica  virginica  versicolor
## [121] virginica  versicolor virginica  versicolor virginica  virginica
## [127] versicolor versicolor virginica  virginica  virginica  virginica
## [133] virginica  versicolor versicolor virginica  virginica  virginica
## [139] versicolor virginica  virginica  virginica  versicolor virginica
## [145] virginica  virginica  virginica  virginica  virginica  versicolor
## Levels: setosa versicolor virginica
```

```
species.name
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```r
table(class.true=species.name, class.assigned=Qda.iris$class)
```

```
##             class.assigned
## class.true   setosa versicolor virginica
##   setosa         49          1         0
##   versicolor      0         37        13
##   virginica       0         15        35
```

```r
errorsq = (Qda.iris$class != species.name)
errorsq
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
## [121] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [133] FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE  TRUE
```

```r
APERq   = sum(errorsq)/length(species.name)
APERq
```

```
## [1] 0.1933333
```

```r
(15+13+1)/150
```

```
## [1] 0.1933333
```

```r
APER #! of LDA
```

```
## [1] 0.2
```

```r
# REMARK: correct only if we estimate the priors through the sample frequencies!

#! QDA shows a lower APER, does it mean is it better? No, it's just that QDA has more
#! parameters, so a more complicate method, that hence will fit more precisely our
#! train data and produce a lower APER
```

```r
# Compute the estimate of the AER by leave-one-out cross-validation
QdaCV.iris = qda(iris2, species.name, CV=T)
QdaCV.iris$class
```

7.2.7.2 AER

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     versicolor
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     virginica  virginica  virginica  versicolor
##  [55] virginica  versicolor virginica  versicolor virginica  versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor virginica
##  [67] versicolor versicolor virginica  versicolor versicolor versicolor
##  [73] virginica  versicolor virginica  virginica  virginica  virginica
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor virginica  virginica  virginica  versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor versicolor versicolor
```

```
## [103] virginica  versicolor virginica  virginica  versicolor virginica
## [109] virginica  virginica  virginica  virginica  virginica  versicolor
## [115] versicolor virginica  virginica  virginica  virginica  versicolor
## [121] virginica  versicolor virginica  versicolor virginica  virginica
## [127] versicolor versicolor virginica  virginica  virginica  virginica
## [133] virginica  versicolor versicolor virginica  virginica  virginica
## [139] versicolor virginica  virginica  virginica  versicolor virginica
## [145] virginica  virginica  versicolor virginica  versicolor versicolor
## Levels: setosa versicolor virginica
```

```
species.name
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
table(class.true=species.name, class.assignedCV=QdaCV.iris$class)
```

```
##             class.assignedCV
## class.true   setosa versicolor virginica
##   setosa         49          1         0
##   versicolor      0         34        16
##   virginica       0         18        32
```

```
errorsqCV = (QdaCV.iris$class != species.name)
errorsqCV
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE
##  [73]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
## [121] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [133] FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [145] FALSE FALSE  TRUE FALSE  TRUE  TRUE
```

```
AERqCV   = sum(errorsqCV)/length(species.name)
AERqCV
```

```
## [1] 0.2333333
```

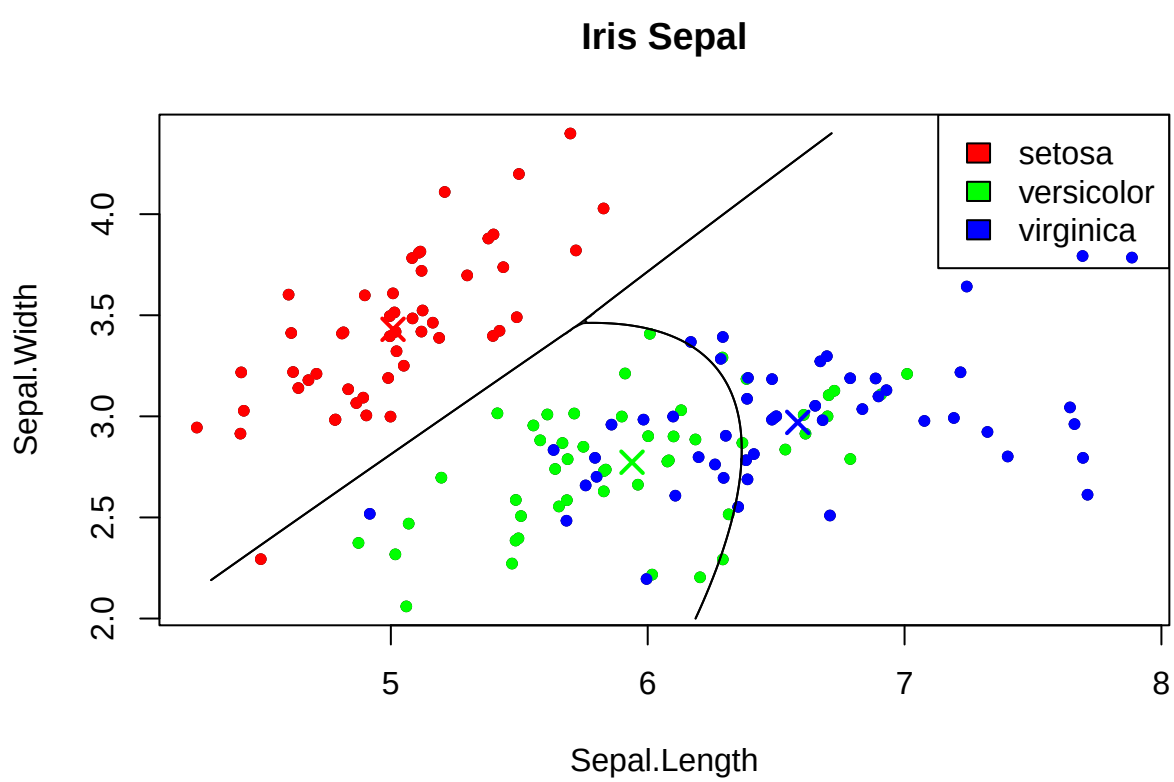
```
# REMARK: correct only if we estimate the priors through the sample frequencies!

AERCV #! of LDA
```

```
## [1] 0.2
```

```
#! In fact actually LDA seems to outperform QDA. Focus on the AER for really
#! choosing the "best" model
```

```
# Plot the partition induced by QDA

plot(iris2, main='Iris Sepal', xlab='Sepal.Length', ylab='Sepal.Width', pch=20)
points(iris2[i1,], col='red', pch=20)
points(iris2[i2,], col='green', pch=20)
points(iris2[i3,], col='blue', pch=20)
legend("topright", legend=levels(species.name), fill=c('red','green','blue'))

points(qda.iris$means, col=c('red','green','blue'), pch=4, lwd=2, cex=1.5)

x  = seq(min(iris[,1]), max(iris[,1]), length=200)
y  = seq(min(iris[,2]), max(iris[,2]), length=200)
xy = expand.grid(Sepal.Length=x, Sepal.Width=y)

z  = predict(qda.iris, xy)$post
z1 = z[,1] - pmax(z[,2], z[,3])
z2 = z[,2] - pmax(z[,1], z[,3])
z3 = z[,3] - pmax(z[,1], z[,2])

contour(x, y, matrix(z1, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z2, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z3, 200), levels=0, drawlabels=F, add=T)
```

7.2.7.3 Partition generated by QDA



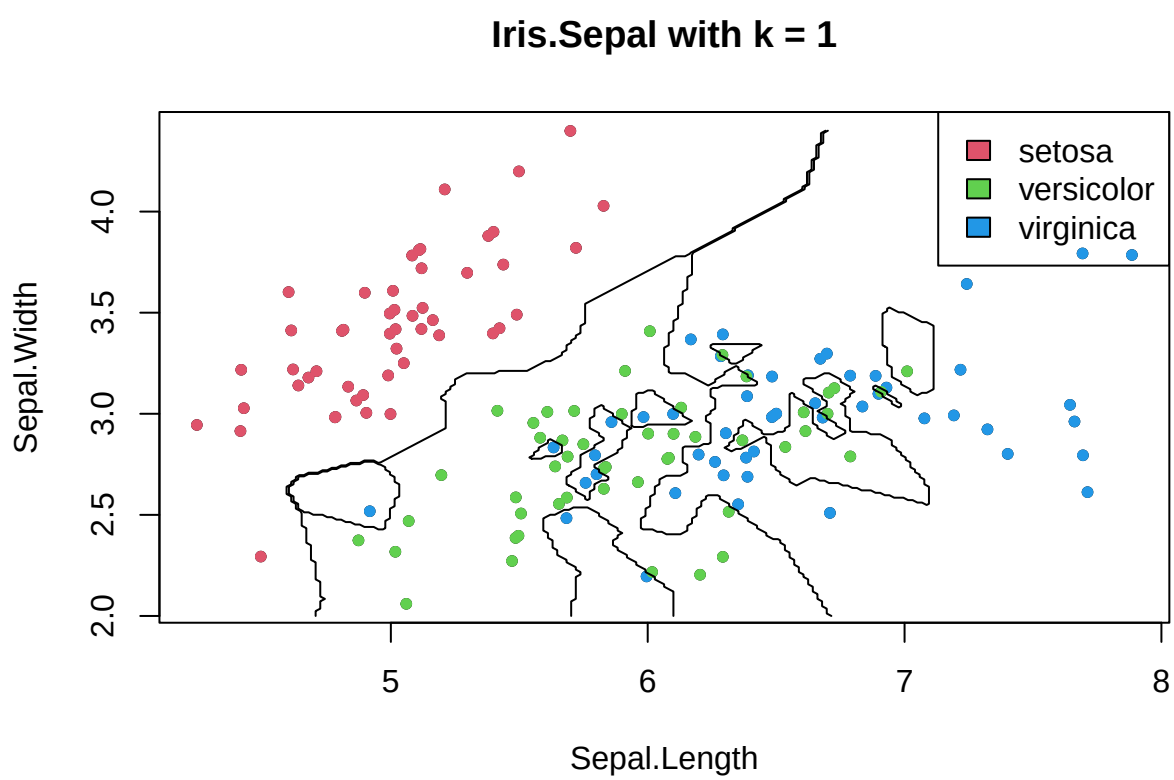
```
open3d()
```

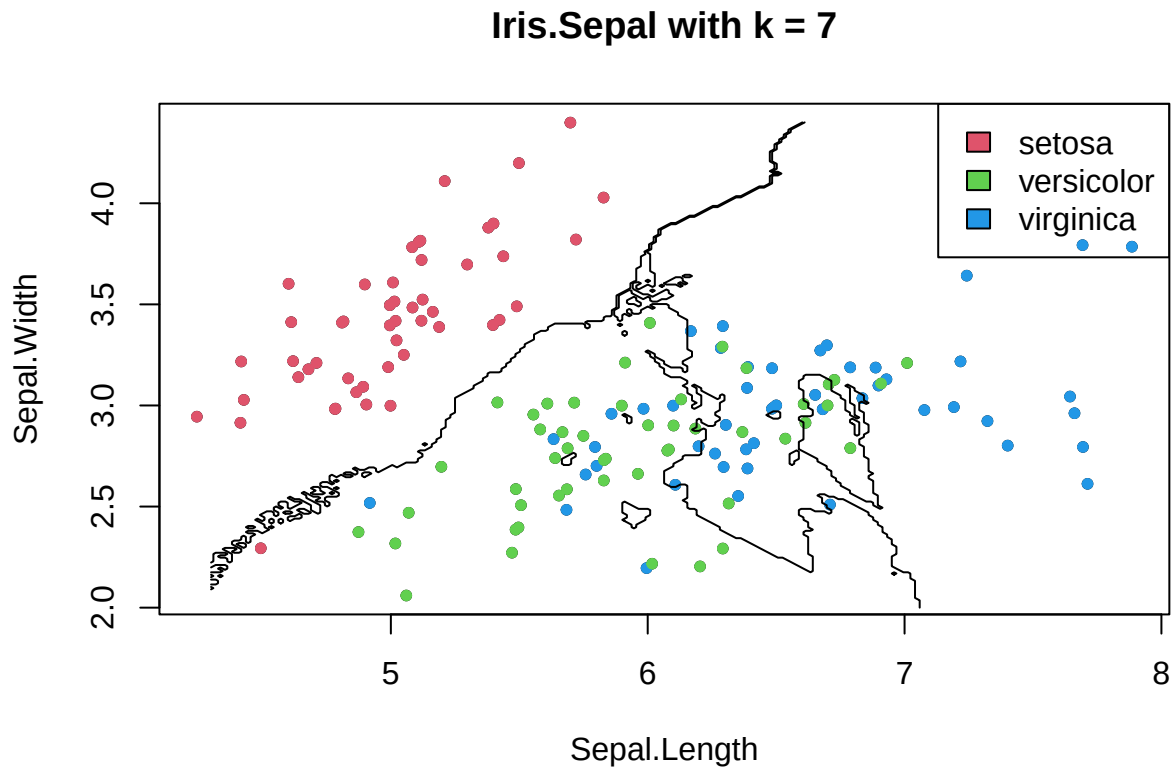
```
## wgl
##   2
```

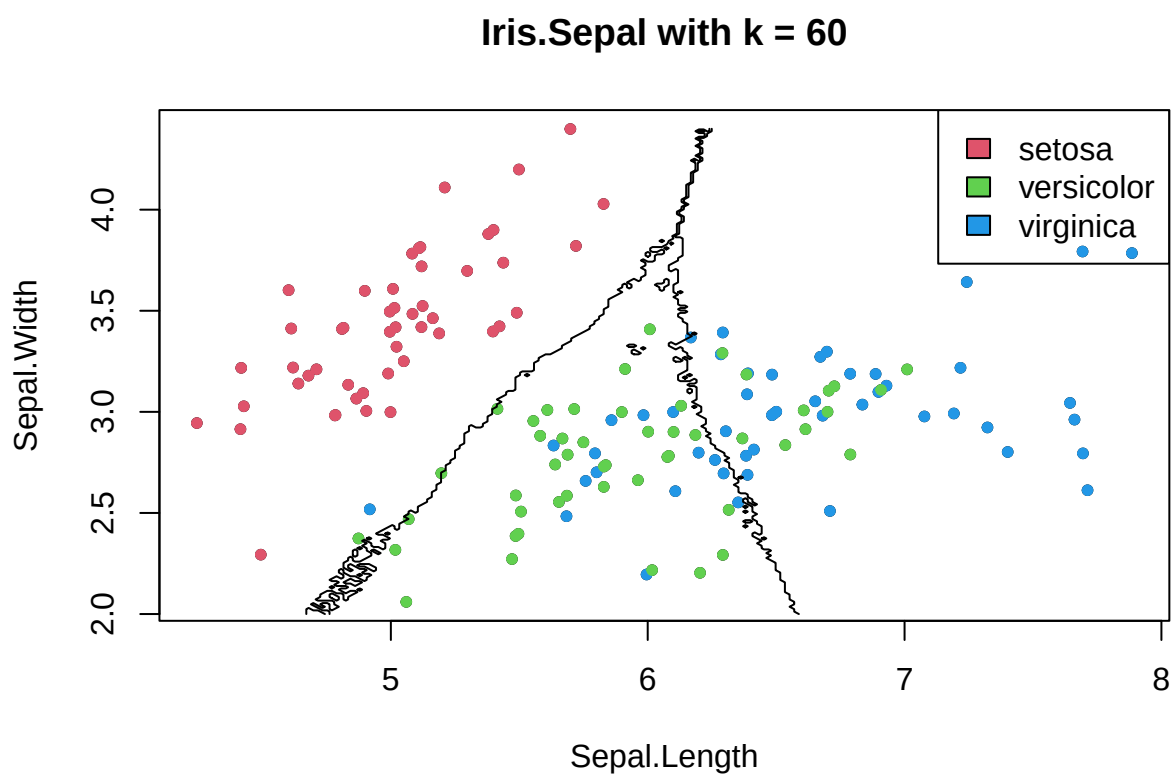
```
points3d(iris2[i1,1], iris2[i1,2], 0, col='red', pch=15)
points3d(iris2[i2,1], iris2[i3,2], 0, col='green', pch=15)
points3d(iris2[i3,1], iris2[i2,2], 0, col='blue', pch=15)
surface3d(x, y, dmvnorm(xy, m1, S1) / 3, alpha=0.4, color='red')
surface3d(x, y, dmvnorm(xy, m2, S2) / 3, alpha=0.4, color='green', add=T)
surface3d(x, y, dmvnorm(xy, m3, S3) / 3, alpha=0.4, color='blue', add=T)
box3d()
```

7.2.8 KNN

```
### knn-classifier
###----------------
# Plot the partition induced by knn

for (kk in c(1,7,60)){

  plot(iris2, xlab='Sepal.Length', ylab='Sepal.Width', pch=20,
       main=paste("Iris.Sepal with k =",kk))
  points(iris2[i1,], col=2, pch=20)
  points(iris2[i3,], col=4, pch=20)
  points(iris2[i2,], col=3, pch=20)
  legend("topright", legend=levels(species.name), fill=c(2,3,4))

  x  = seq(min(iris[,1]), max(iris[,1]), length=200)
  y  = seq(min(iris[,2]), max(iris[,2]), length=200)
  xy = expand.grid(Sepal.Length=x, Sepal.Width=y)

  iris.knn = knn(train = iris2, test = xy, cl = iris$Species, k = kk)
  z  = as.numeric(iris.knn)
  contour(x, y, matrix(z, 200), levels=c(1.5, 2.5), drawlabels=F, add=T)
}
```







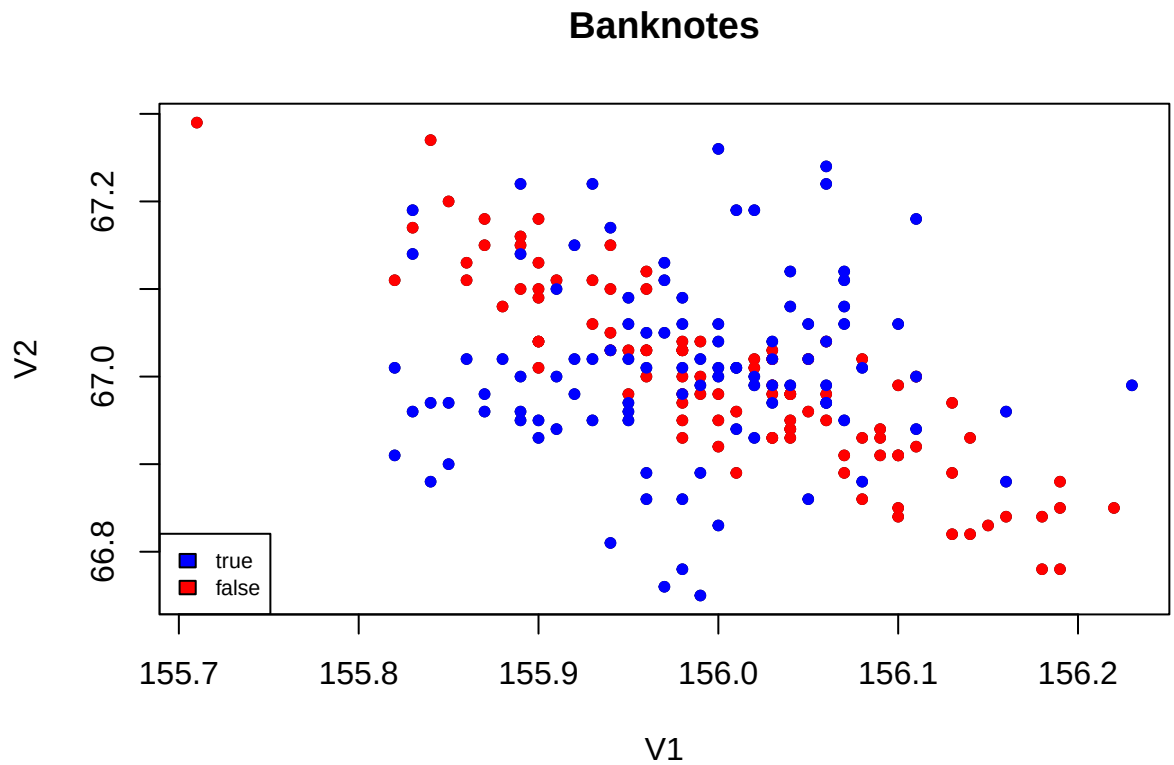

```
#! The problem of KNN is that it may overfit a lot, there are sometimes sub-regions
#! insted of clear ones as with L/QDA. But we can fix it working on k:
#!  - lower k: lots of disconnected regions as we try to capture all the close
#!   data, so there is overfit (perfect APER, bad AER)
#!  - higher k: better, we start to re-get the regions of L/QDA
```

7.3 On 2 classes, bivariate

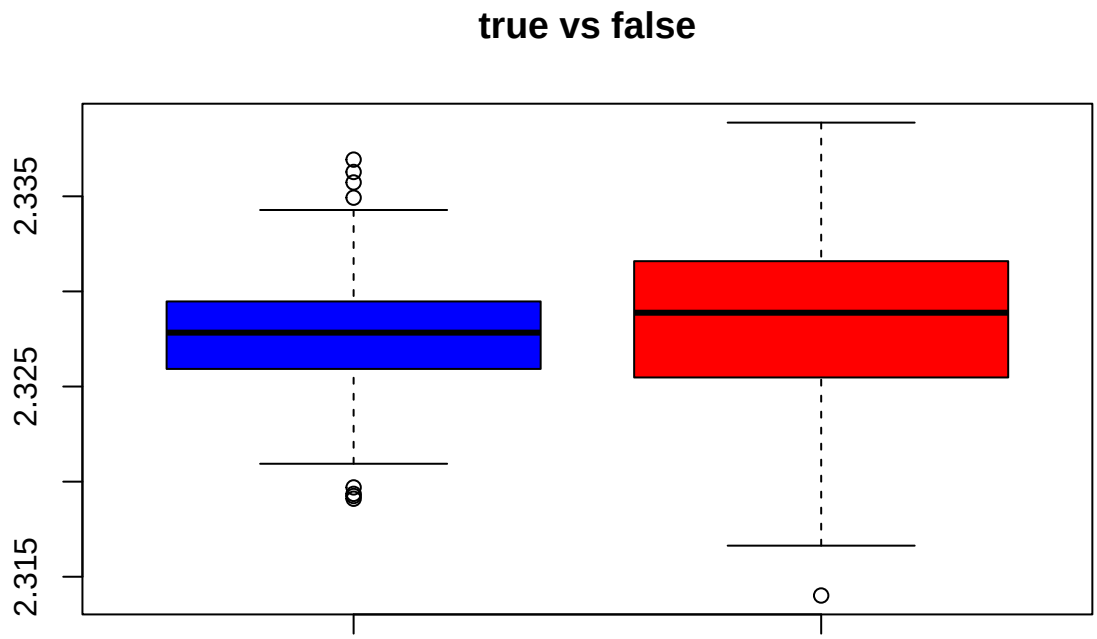
```r
#_____________________________________________________________________________
##### Problem 2 of 9/09/2009
#####--------------------------
# The vending machines of the Exxon fuel contain optical detectors able
# to measure the size of the banknotes inserted. Knowing that 0.1% of the
# 10$ banknotes in circulation are counterfeit, Exxon would like to implement a
# software to identify false 10$ banknotes, as to minimize the economic losses.
# Assuming that:
#  - both the populations of real and false banknotes follow a normal
#    distribution (with different mean and covariance matrices);
#  - accepting a false banknote leads to an economic loss of 10$;
#  - rejecting a true banknote brings a economic loss quantifiable in 5 cents;
# satisfy the following requests of the Exxon:
# a) build an appropriate classifier, estimating the unknown parameters
#    starting from the two datasets moneytrue.txt and moneyfalse.txt, containing
#    data about 100 true banknotes and 100 counterfeit banknotes (in mm).
#    Qualitatively show the two classification regions in a graph;
# b) calculate the APER of the classifier and, based on the APER, estimate the
#    expected economic damage of the classifier;
# c) what is the estimated probability that the first 10$ banknote inserted in the
#    machine is rejected?

true = read.table('Lab07/moneytrue.txt',header=TRUE)
false = read.table('Lab07/moneyfalse.txt',header=TRUE)

banknotes = rbind(true,false)
head(banknotes)
```

```
##       V1    V2
## 1 156.06 66.97
## 2 156.08 66.88
## 3 155.95 67.06
## 4 155.98 67.09
## 5 155.85 66.90
## 6 155.92 67.02
```

```r
vf = factor(rep(c('true','false'),each=100), levels=c('true','false'))


plot(banknotes[,1:2], main='Banknotes', xlab='V1', ylab='V2', pch=20)
points(false, col='red', pch=20)
points(true, col='blue', pch=20)
legend('bottomleft', legend=levels(vf), fill=c('blue','red'), cex=.7)
```




```
#! Some data points may overlap (we can see the "grid of points")
#! so consider also doing jittering

#! We can see already a possible difference: the true ones are more random,
#! sparse, while the false ones are more regular
#! So we could take the ratio of the two features and make an analysis on it (like
#! distribution, MANOVA, test for the mean, ecc) instead on working on the data

#! Like
true_ratio = true$V1/true$V2
false_ratio = false$V1/false$V2
boxplot(true_ratio,false_ratio, col=c("blue","red"), main="true vs false")
```




```
shapiro.test(true_ratio)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  true_ratio
## W = 0.97805, p-value = 0.09359
```

```
shapiro.test(false_ratio)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  false_ratio
## W = 0.98765, p-value = 0.4826
```

```
#! ecc
```

7.3.1 Parameters definition

```
# question a)
library(MVN)
library(mvtnorm)
library(mvnormtest)
mshapiro.test(t(true))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.98682, p-value = 0.4256
```

```
mshapiro.test(t(false))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.98981, p-value = 0.6495
```

```r
#! Multivariate normality was a requirement for L/QDA

# misclassification costs
c.vf = 10
c.fv = 0.05

#prior probabilities
pf = 0.001
pt = 1-0.001

#! When we have just two classes there is the trick to combine misclassification
#! costs into the prior probabilities, as some R functions cant deal with costs
#! new priors = (old priors) adjusted by (miscl costs)

# Prior modified to account for the misclassification costs
prior.c = c(pt*c.fv/(c.vf*pf+c.fv*pt), pf*c.vf/(c.vf*pf+c.fv*pt))
prior.c
```

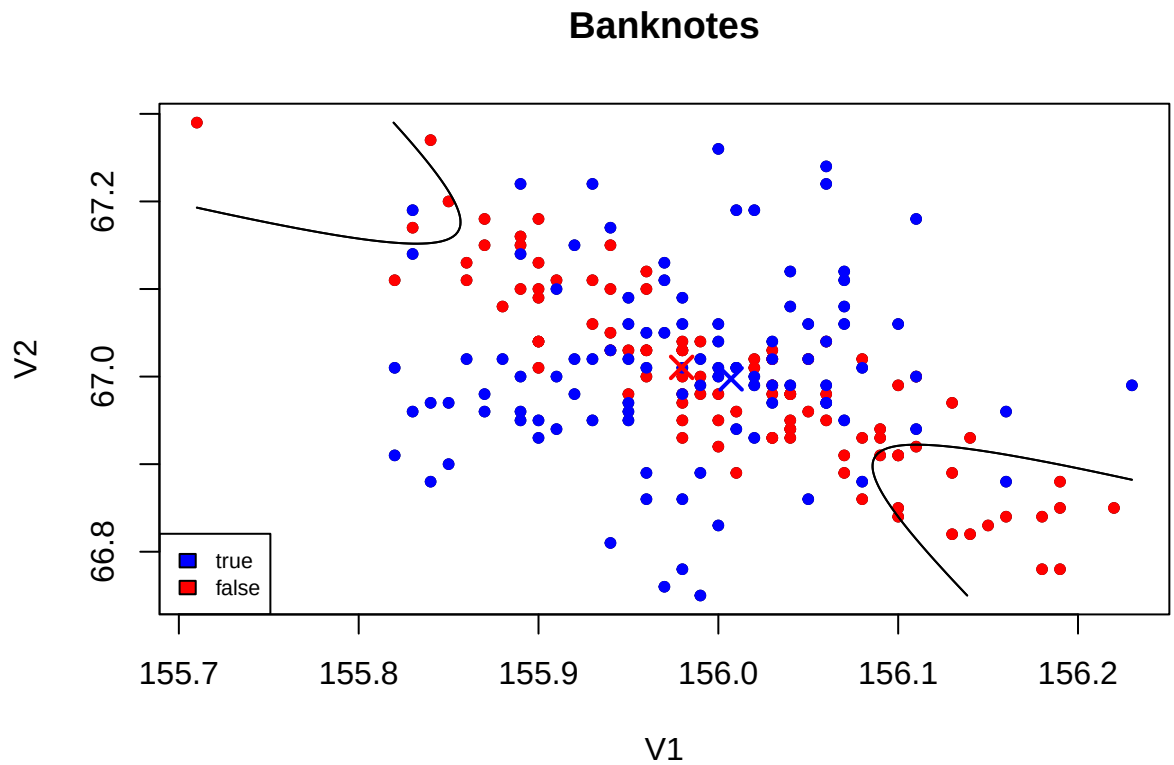
```
## [1] 0.8331943 0.1668057
```

7.3.2 QDA

```r
#! QDA or LDA? The text cleary say we dont have the assumption of equal covariance
#! matrices (requirement for LDA), so we must do QDA
qda.m = qda(banknotes, vf, prior=prior.c)
qda.m
```

```
## Call:
## qda(banknotes, vf, prior = prior.c)
##
## Prior probabilities of groups:
##      true     false
## 0.8331943 0.1668057
##
## Group means:
##             V1      V2
## true  155.9796 67.0105
## false 156.0071 66.9974
```

```r
plot(banknotes[,1:2], main='Banknotes', xlab='V1', ylab='V2', pch=20)
points(false, col='red', pch=20)
points(true, col='blue', pch=20)
legend('bottomleft', legend=levels(vf), fill=c('blue','red'), cex=.7)

points(qda.m$means, pch=4,col=c('red','blue') , lwd=2, cex=1.5)

x  = seq(min(banknotes[,1]), max(banknotes[,1]), length=200)
y  = seq(min(banknotes[,2]), max(banknotes[,2]), length=200)
xy = expand.grid(V1=x, V2=y)

z  = predict(qda.m, xy)$post
z1 = z[,1] - z[,2]
z2 = z[,2] - z[,1]

contour(x, y, matrix(z1, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z2, 200), levels=0, drawlabels=F, add=T)
```




```
m1 = colMeans(true)
m2 = colMeans(false)

S1 = cov(true)
S2 = cov(false)

open3d()
```

```
## wgl
##   3
```

```
points3d(true[,1], true[,2], 0, col='blue', pch=15)
points3d(false[,1], false[,2], 0, col='red', pch=15)
surface3d(x,y,matrix(dmvnorm(xy, m1, S1)*prior.c[1]/100 , 200), alpha=0.6, color='blue')
surface3d(x,y,matrix(dmvnorm(xy, m2, S2)*prior.c[2]/100 , 200), alpha=0.6, color='red', add=T)
box3d()
```

7.3.3 APER computation

```
# question b)
# APER
Qda.m = predict(qda.m)
table(class.true=vf, class.assigned=Qda.m$class)
```

```
##           class.assigned
## class.true true false
##      true    98     2
##      false   80    20
```

```
#! We had different priors so we adjust it
APER  = (2*pt+80*pf)/100
#! But dont consider costs of miscl for the APER computation!
#! Costs intervene later in the real "loss" sense
APER
```

```
## [1] 0.02078
```

```r
#! But again is almost always better to study the AER through CR, cross validation

#! Here
# Expected economic loss:
2/100*pt*c.fv+80/100*pf*c.vf
```

```
## [1] 0.008999
```

```r
# question c)
#! We use the total probabilities law
# P[rejected] = P[rejected | true]P[true] + P[rejected | false]P[false]
2/100*pt + 20/100*pf
```

```
## [1] 0.02018
```

7.4 FDA

```r
###-----------------------------------
### FISHER DISCRIMINANT ANALYSIS
###-----------------------------------
### Let's change viewpoint: we look for the directions that highlight
### the discrimination among groups
### -> we look for the canonical directions

#! Often it will be an iterative procedure:
#! transform data -> LDA -> plot scores -> satisfied? / fantastic
#!                                                    \ try different transformations
#! Like polinomials, log, exp, a mix of them, ecc

# REMARK. Assumptions: homogeneity of the covariance structure
# [we relax the normality assumption]

# Let's consider again the iris dataset
attach(iris)

species.name = factor(Species, labels=c('setosa','versicolor','virginica'))

g = 3

i1 = which(species.name=='setosa')
i2 = which(species.name=='versicolor')
i3 = which(species.name=='virginica')

n1 = length(i1)
n2 = length(i2)
n3 = length(i3)
n = n1+n2+n3

detach(iris)

iris2 = iris[,1:2]
head(iris2)
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

```r
set.seed(1)
```

7.4.1 Jittering

```r
iris2 = iris2 + cbind(rnorm(150, sd=0.025))
# jittering

m =  colMeans(iris2)
```

```r
m1 = colMeans(iris2[i1,])
m2 = colMeans(iris2[i2,])
m3 = colMeans(iris2[i3,])

S1 = cov(iris2[i1,])
S2 = cov(iris2[i2,])
S3 = cov(iris2[i3,])
Sp  = ((n1-1)*S1+(n2-1)*S2+(n3-1)*S3)/(n-g)

# covariance between groups (estimate)
B = 1/g*(cbind(m1 - m) %*% rbind(m1 - m) +
           cbind(m2 - m) %*% rbind(m2 - m) +
           cbind(m3 - m) %*% rbind(m3 - m))
B
```

```
##              Sepal.Length Sepal.Width
## Sepal.Length    0.4183106 -0.13442959
## Sepal.Width    -0.1344296  0.07591288
```

```r
# covariance within groups (estimate)
Sp
```

```
##              Sepal.Length Sepal.Width
## Sepal.Length   0.26682872  0.09357996
## Sepal.Width    0.09357996  0.11528495
```

```r
# how many coordinates?
g = 3
p = 2
s = min(g-1,p)
s
```

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

```r
#! The s represents the best space (here R^2) in which we could see the whole discriminants

# Matrix Sp^(-1/2)
val.Sp = eigen(Sp)$val
vec.Sp = eigen(Sp)$vec
invSp.2 = 1/sqrt(val.Sp[1])*vec.Sp[,1]%*%t(vec.Sp[,1]) +
  1/sqrt(val.Sp[2])*vec.Sp[,2]%*%t(vec.Sp[,2])
invSp.2
```

```
##            [,1]       [,2]
## [1,]  2.1570659 -0.7657082
## [2,] -0.7657082  3.3970568
```

```r
# spectral decomposition of Sp^(-1/2) B Sp^(-1/2)
spec.dec = eigen(invSp.2 %*% B %*% invSp.2)

# first canonical coordinate
a1 = invSp.2 %*% spec.dec$vec[,1]
a1
```

```
##           [,1]
## [1,] -2.137114
## [2,]  2.789529
```

```r
# second canonical coordinate
a2 = invSp.2 %*% spec.dec$vec[,2]
a2
```

```
##            [,1]
## [1,] -0.8197481
## [2,] -2.0844258
```

```r
# compare with the output of lda():
lda.iris = lda(iris2, species.name)
lda.iris
```

```
## Call:
## lda(iris2, species.name)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width
## setosa         5.008511    3.430511
## versicolor     5.938933    2.772933
## virginica      6.584188    2.970188
##
## Coefficients of linear discriminants:
##                    LD1        LD2
## Sepal.Length -2.137114 -0.8197481
## Sepal.Width   2.789529 -2.0844258
##
## Proportion of trace:
##    LD1    LD2
## 0.9644 0.0356
```

```
lda.iris$scaling
```

```
##                    LD1        LD2
## Sepal.Length -2.137114 -0.8197481
## Sepal.Width   2.789529 -2.0844258
```

```
t(lda.iris$scaling[1:2])
```

```
##           [,1]     [,2]
## [1,] -2.137114 2.789529
```

```
a1
```

```
##           [,1]
## [1,] -2.137114
## [2,]  2.789529
```

```
t(lda.iris$scaling[3:4])
```

```
##            [,1]      [,2]
## [1,] -0.8197481 -2.084426
```

```
a2
```

```
##            [,1]
## [1,] -0.8197481
## [2,] -2.0844258
```

```
spec.dec$val/sum(spec.dec$val)
```

```
## [1] 0.96439337 0.03560663
```

7.4.2 Canonical reference system

```
### How are the data classified?
# Compute the canonical coordinates of the data
cc1.iris = as.matrix(iris2)%*%a1
cc2.iris = as.matrix(iris2)%*%a2
coord.cc = cbind(cc1.iris,cc2.iris)

#! In fact the idea of Fisher's LDA is to move to a different coord system and
#! from it get our classifier, by the closeness to the (projection of the) mean
#! of each group

# Compute the coordinates of the mean within groups along the canonical directions
cc.m1 = c(m1%*%a1, m1%*%a2)
```

```r
cc.m2 = c(m2%*%a1, m2%*%a2)
cc.m3 = c(m3%*%a1, m3%*%a2)

# Assign data to groups
f.class=rep(0, n)
for(i in 1:n) # for each datum
{
  # Compute the Euclidean distance of the i-th datum from mean within the groups
  dist.m=c(d1=sqrt(sum((coord.cc[i,]-cc.m1)^2)),
           d2=sqrt(sum((coord.cc[i,]-cc.m2)^2)),
           d3=sqrt(sum((coord.cc[i,]-cc.m3)^2)))
  # Assign the datum to the group whose mean is the nearest
  f.class[i]=which.min(dist.m)
}
f.class
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 3 2 3 2 2 2 3 2 2 2 2 2 2 3 2 2 3 2 2 2 3 2
##  [75] 3 3 3 3 2 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 3 3 3 2 2 3 3 3 3 3 3 2 3 2 3 2 3 3 3 2 3 3 3 3 3
## [149] 2 2
```

```r
table(class.true=species.name, class.assigned=f.class)
```

```
##             class.assigned
## class.true    1  2  3
##   setosa     49  1  0
##   versicolor  0 36 14
##   virginica   0 15 35
```

```r
errors = 150 - sum(diag(table(class.true=species.name, class.assigned=f.class)))
errors
```

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

```r
length(species.name)
```

```
## [1] 150
```

```r
APERf   = errors/length(species.name)
APERf #! f is just for Fisher
```

```
## [1] 0.2
```

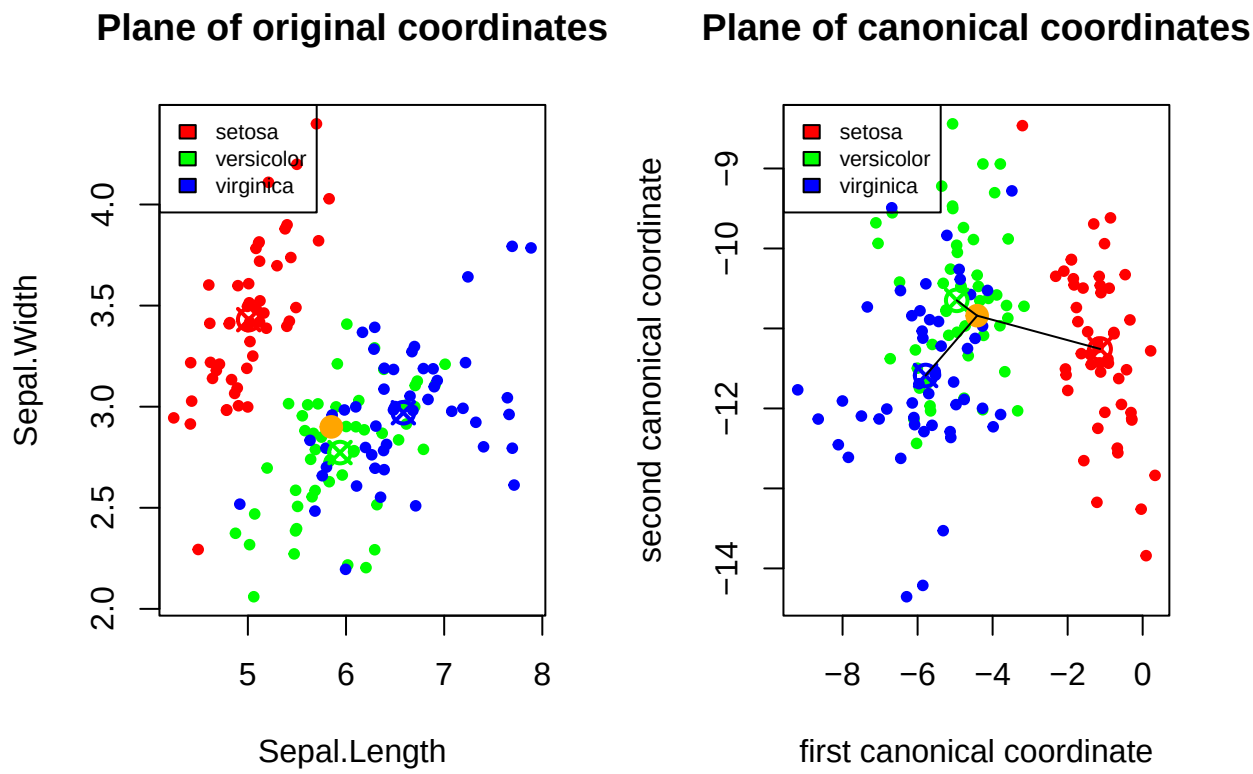
7.4.3 New obs classification

```r
### How do I classify a new observation?
x.new = c(5.85, 2.90)
# compute the canonical coordinates
cc.new = c(x.new%*%a1, x.new%*%a2)
# compute the distance from the means
dist.m = c(d1=sqrt(sum((cc.new-cc.m1)^2)),
           d2=sqrt(sum((cc.new-cc.m2)^2)),
           d3=sqrt(sum((cc.new-cc.m3)^2)))
# assign to the nearest mean
which.min(dist.m)
```

```
## d2
##  2
```

```r
color.species=rep(c('red','green','blue'), each=50)

# visually

par(mfrow=c(1,2))
plot(iris2[,1], iris2[,2], main='Plane of original coordinates',
     xlab='Sepal.Length', ylab='Sepal.Width', pch=20, col=as.character(color.species))
legend("topleft", legend=levels(species.name), fill=c('red','green','blue'), cex=.7)
```

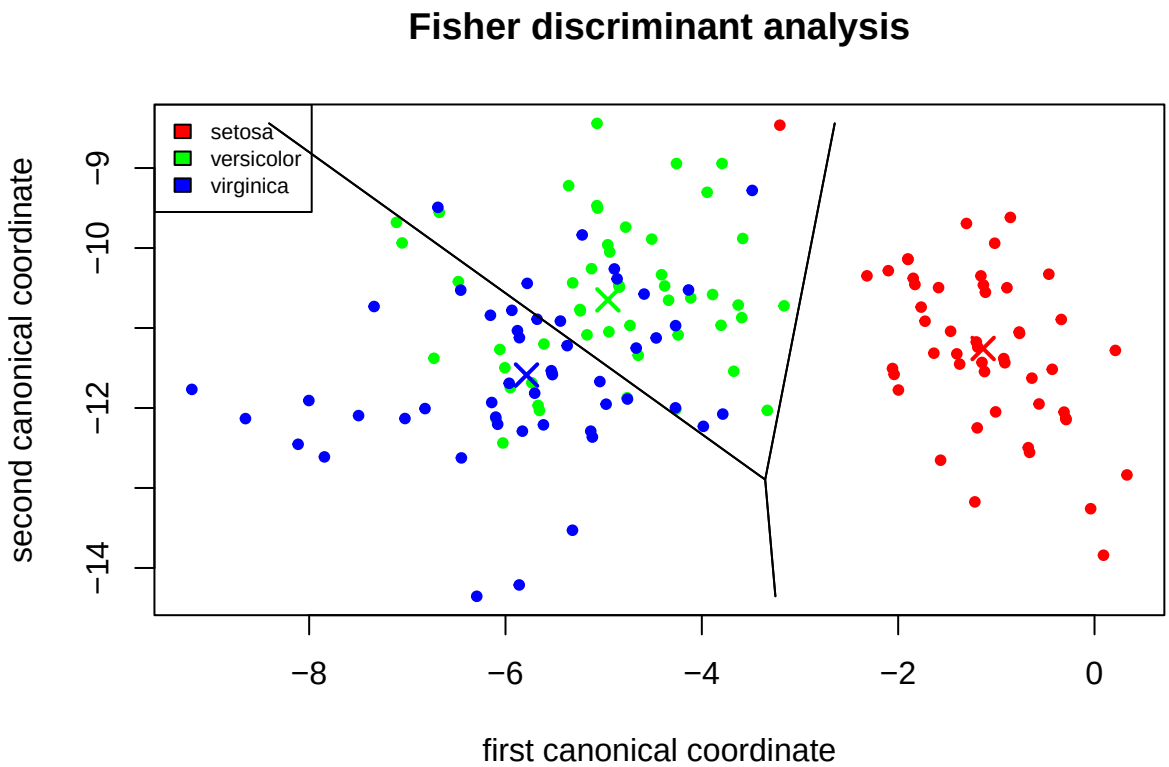
```
points(x.new[1], x.new[2], col='orange', pch=19, cex=1.5)
points(m1[1], m1[2], pch=13,col='red' , lwd=2, cex=1.5)
points(m2[1], m2[2], pch=13,col='green' , lwd=2, cex=1.5)
points(m3[1], m3[2], pch=13,col='blue' , lwd=2, cex=1.5)

plot(cc1.iris, cc2.iris, main='Plane of canonical coordinates',
     xlab='first canonical coordinate', ylab='second canonical coordinate',
     pch=20, col=as.character(color.species))
legend("topleft", legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

points(cc.m1[1], cc.m1[2], pch=13,col='red' , lwd=2, cex=1.5)
points(cc.m2[1], cc.m2[2], pch=13,col='green' , lwd=2, cex=1.5)
points(cc.m3[1], cc.m3[2], pch=13,col='blue' , lwd=2, cex=1.5)

points(cc.new[1], cc.new[2], col='orange', pch=19, cex=1.5)

segments(cc.m1[1], cc.m1[2], cc.new[1], cc.new[2])
segments(cc.m2[1], cc.m2[2], cc.new[1], cc.new[2])
segments(cc.m3[1], cc.m3[2], cc.new[1], cc.new[2])
```



7.4.4 Partition generated by FDA

```
### We plot the partition generated by the canonical coordinates
color.species = species.name
levels(color.species) = c('red','green','blue')


plot(cc1.iris, cc2.iris, main='Fisher discriminant analysis',
     xlab='first canonical coordinate', ylab='second canonical coordinate',
     pch=20, col=as.character(color.species))
legend("topleft", legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

points(cc.m1[1], cc.m1[2], pch=4,col='red' , lwd=2, cex=1.5)
points(cc.m2[1], cc.m2[2], pch=4,col='green' , lwd=2, cex=1.5)
points(cc.m3[1], cc.m3[2], pch=4,col='blue' , lwd=2, cex=1.5)

x.cc  = seq(min(cc1.iris),max(cc1.iris),len=200)
```

```
y.cc  = seq(min(cc2.iris),max(cc2.iris),len=200)
xy.cc = expand.grid(cc1=x.cc, cc2=y.cc)

z  = cbind( sqrt(rowSums(scale(xy.cc,cc.m1,scale=FALSE)^2)),
            sqrt(rowSums(scale(xy.cc,cc.m2,scale=FALSE)^2)),
            sqrt(rowSums(scale(xy.cc,cc.m3,scale=FALSE)^2)) )
z1.cc = z[,1] - pmin(z[,2], z[,3])
z2.cc = z[,2] - pmin(z[,1], z[,3])
z3.cc = z[,3] - pmin(z[,1], z[,2])

contour(x.cc, y.cc, matrix(z1.cc, 200), levels=0, drawlabels=F, add=T)
contour(x.cc, y.cc, matrix(z2.cc, 200), levels=0, drawlabels=F, add=T)
contour(x.cc, y.cc, matrix(z3.cc, 200), levels=0, drawlabels=F, add=T)
```

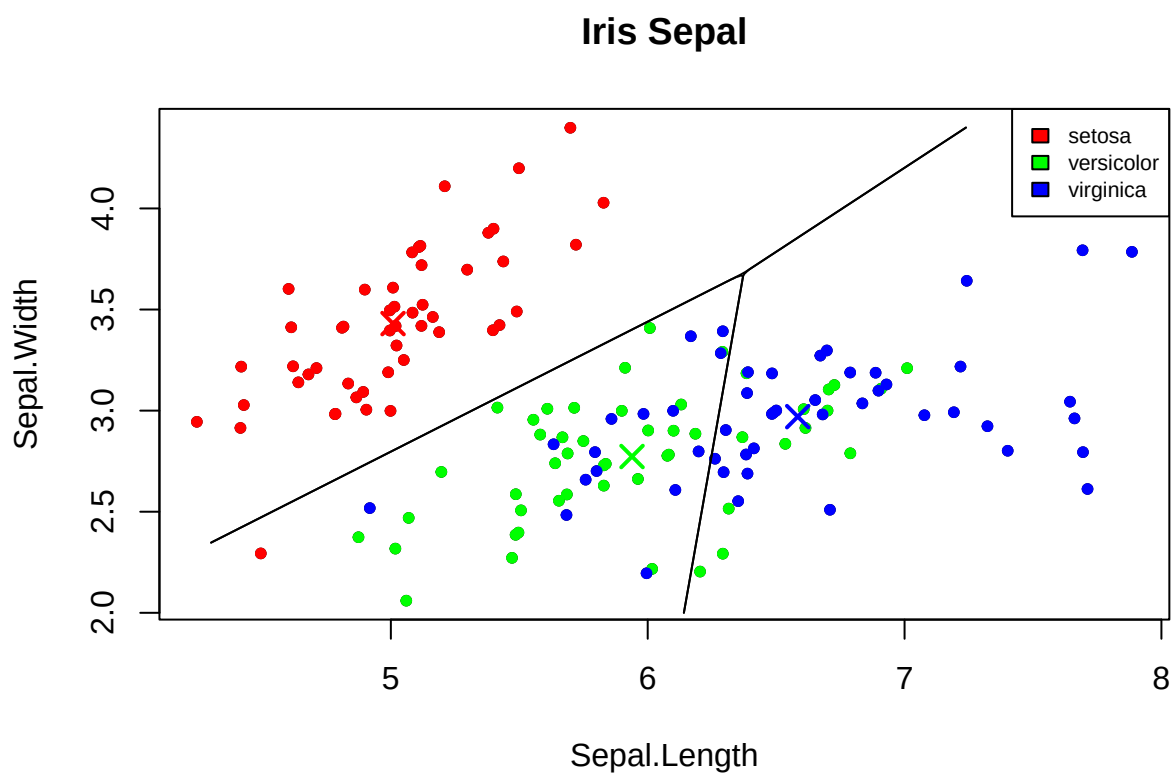



```
#! We get our Voronoi tessellation
```

7.4.5 Comparison with LDA

```
# Plot LDA
plot(iris2, main='Iris Sepal', xlab='Sepal.Length', ylab='Sepal.Width', pch=20)
points(iris2[i1,], col='red', pch=20)
points(iris2[i2,], col='green', pch=20)
points(iris2[i3,], col='blue', pch=20)
legend("topright", legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

points(rbind(m1,m2,m3), pch=4,col=c('red','green','blue') , lwd=2, cex=1.5)

x  = seq(min(iris[,1]), max(iris[,1]), length=200)
y  = seq(min(iris[,2]), max(iris[,2]), length=200)
xy = expand.grid(Sepal.Length=x, Sepal.Width=y)

z  = predict(lda.iris, xy)$post  # these are P_i*f_i(x,y)
z1 = z[,1] - pmax(z[,2], z[,3])  # P_1*f_1(x,y)-max{P_j*f_j(x,y)}
z2 = z[,2] - pmax(z[,1], z[,3])  # P_2*f_2(x,y)-max{P_j*f_j(x,y)}
z3 = z[,3] - pmax(z[,1], z[,2])  # P_3*f_3(x,y)-max{P_j*f_j(x,y)}

# Plot the contour line of level (levels=0) of z1, z2, z3:
```

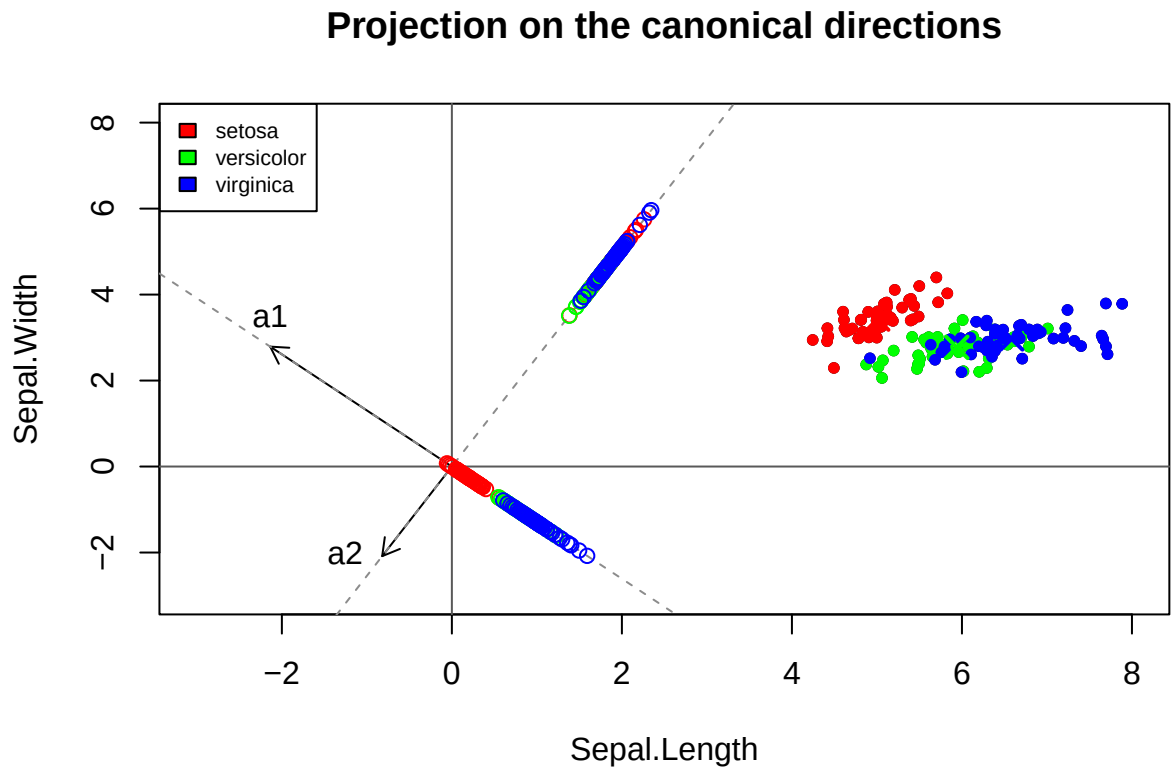
```
# P_i*f_i(x,y)-max{P_j*f_j(x,y)}=0 ie, boundary between R.i and R.j
# where j realizes the max.
contour(x, y, matrix(z1, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z2, 200), levels=0, drawlabels=F, add=T)
contour(x, y, matrix(z3, 200), levels=0, drawlabels=F, add=T)
```



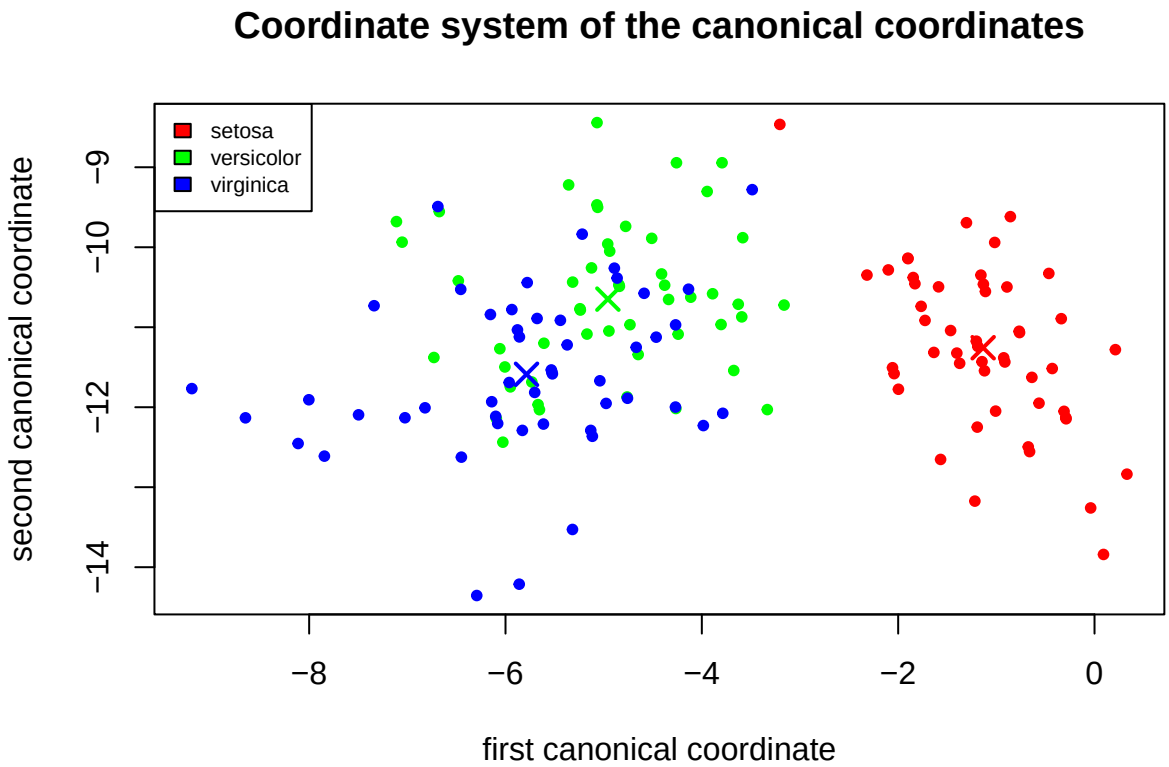

```
#! Similar plots but oriented wrt different coord systems
```

7.4.6 Projections on the canonical directions

```
# Plot of the projections on the canonical directions (not orthogonal!)

shift = c(0,0)
plot(iris2-shift, xlab='Sepal.Length', ylab='Sepal.Width', pch=20,
     xlim=c(-3,8), ylim=c(-3,8))
# title(main="Projection on the canonical directions\n(data shifted for clarity)")
title(main="Projection on the canonical directions")
points(iris2[i1,]-shift, col='red', pch=20)
points(iris2[i2,]-shift, col='green', pch=20)
points(iris2[i3,]-shift, col='blue', pch=20)
legend('topleft', legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

points(rbind(m1-shift,m2-shift,m3-shift), pch=4,col=c('red','green','blue') , lwd=2, cex=1.5)
abline(h=0,v=0, col='grey35')

arrows(x0=0, y0=0, x1=a1[1], y1=a1[2], length=.1)
arrows(x0=0, y0=0, x1=a2[1], y1=a2[2], length=.1)

text(a1[1], a1[2], 'a1',pos=3)
text(a2[1], a2[2], 'a2',pos=2)

abline(coef=c(0,(a1[2]/a1[1])), col='grey55',lty=2)
abline(coef=c(0,(a2[2]/a2[1])), col='grey55',lty=2)

points(cc1.iris*a1[1]/(sum(a1^2)),cc1.iris*a1[2]/(sum(a1^2)),col=as.character(color.species))
points(cc2.iris*a2[1]/(sum(a2^2)),cc2.iris*a2[2]/(sum(a2^2)),col=as.character(color.species))
```




```
#! We see that a1 and a2 are not orthogonal! They are uncorrelated, but NOT NEC orthogonal

plot(cc1.iris, cc2.iris, main='Coordinate system of the canonical coordinates',
     xlab='first canonical coordinate', ylab='second canonical coordinate',
     pch=20, col=as.character(color.species))
legend('topleft', legend=levels(species.name), fill=c('red','green','blue'), cex=.7)

cc.m1 = c(m1%*%a1, m1%*%a2)
cc.m2 = c(m2%*%a1, m2%*%a2)
cc.m3 = c(m3%*%a1, m3%*%a2)

points(cc.m1[1], cc.m1[2], pch=4,col='red' , lwd=2, cex=1.5)
points(cc.m2[1], cc.m2[2], pch=4,col='green' , lwd=2, cex=1.5)
points(cc.m3[1], cc.m3[2], pch=4,col='blue' , lwd=2, cex=1.5)
```

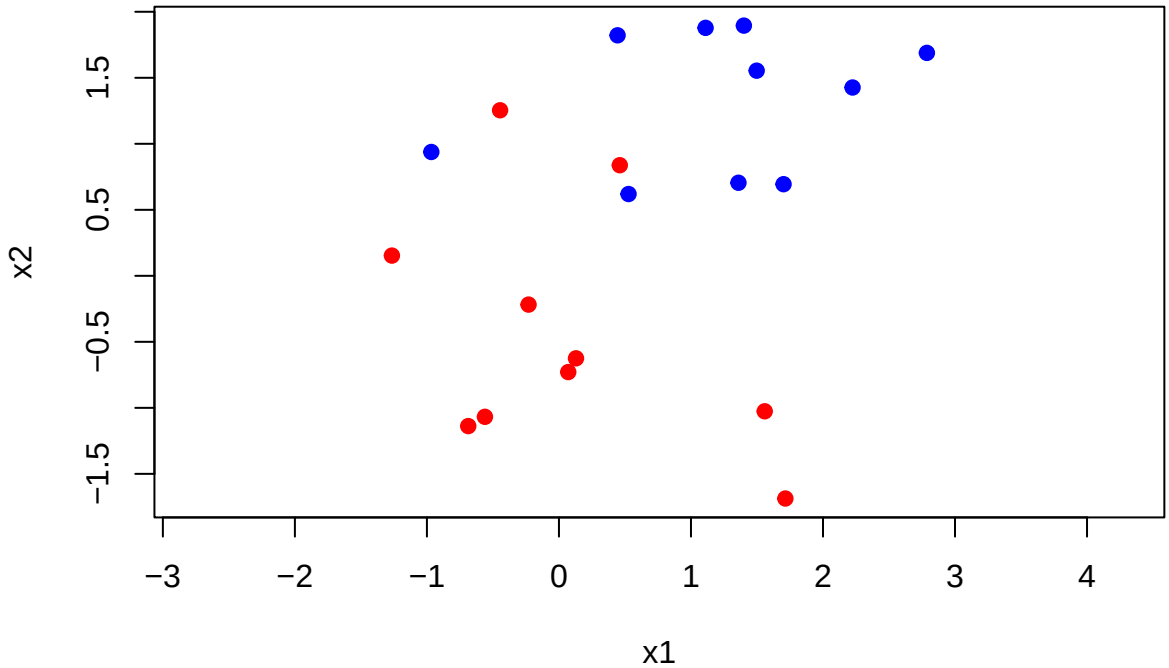


7.5 SVM

```
### Support Vector Machines

library(e1071)
# help(svm)
```

7.5.1 Linear case

```
### Linear case

# Generate the data
set.seed (123)
ndata = 20
#! try more samples

x = matrix (rnorm (ndata*2) , ncol =2)
y = c(rep (-1,10) , rep (1 ,10) )
x[y==1,] = x[y==1,] + 1

# The classes are not separable

plot(x, col =ifelse(y==1, 'blue', 'red'),
     pch=19, xlab='x1', ylab='x2', asp=1)
```

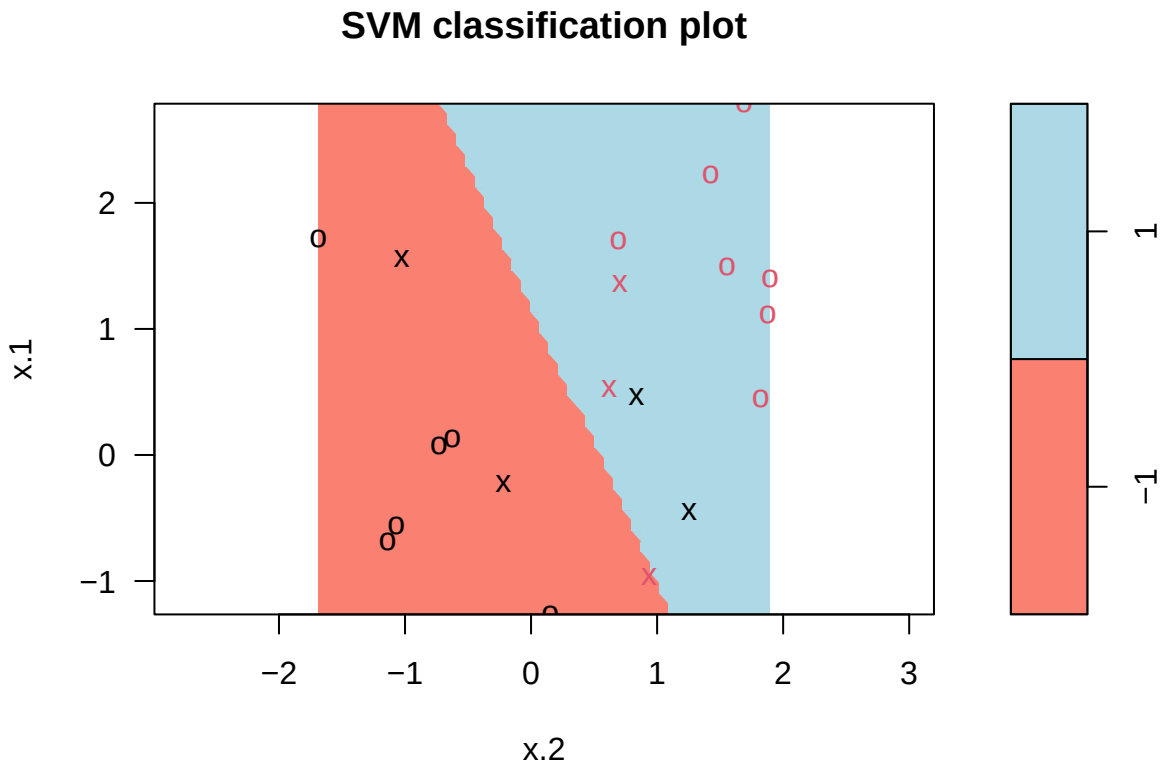


7.5.2 Fit the model

```
# Fit the Support Vector Classifier (kernel = "linear")
# given a cost C
dat = data.frame(x=x, y=as.factor (y))
svmfit = svm(y~., data=dat , kernel ='linear', cost =10, scale =FALSE )
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

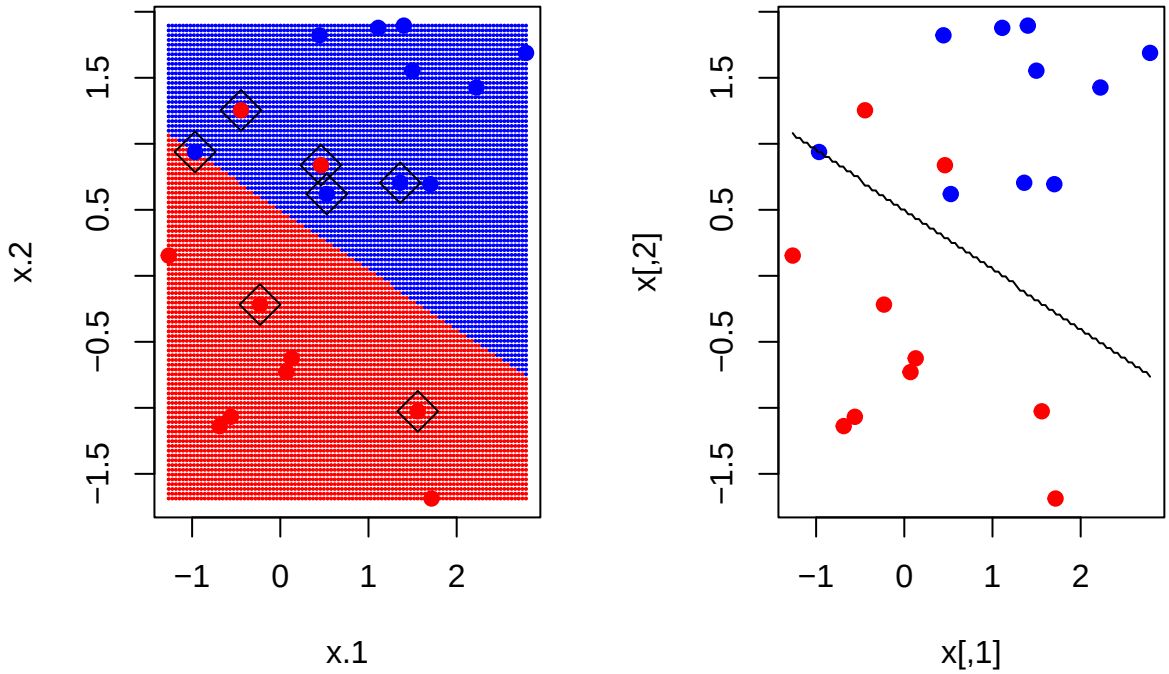
```
#! The cost parameter is the one that regulates the possibility of having points
#! misclassified, ie trepassing the separating boundary (when there is no perfect
#! separation)


par(mfrow=c(1,2))
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```




```
# support vectors are indicated with crosses. Here they are:
svmfit$index
```

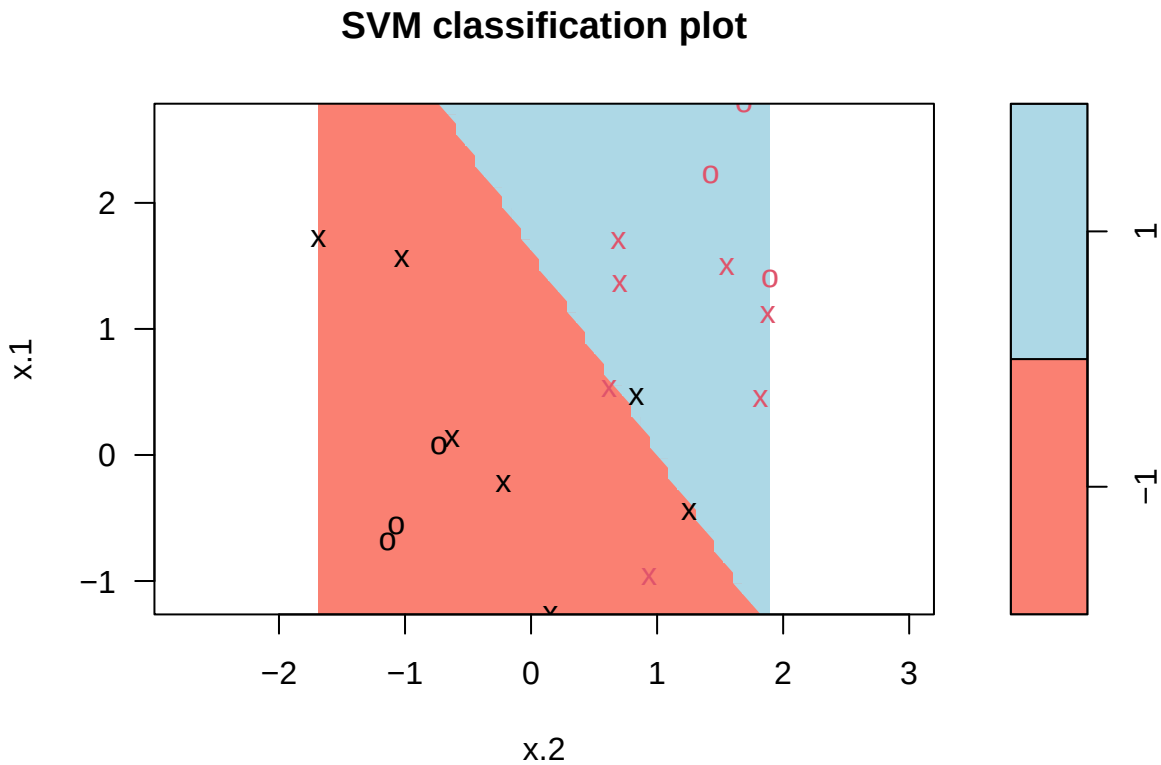
```
## [1]  2  3  7 10 12 18 20
```

```
#! Support vectors are the points which, if moved, will change the separating
#! hyperplane, as are the ones close to the boundary

###
n.g = 100

xgrid = expand.grid(x.1=seq(from=range(dat$x.1)[1],to=range(dat$x.1)[2],length=n.g),
                    x.2=seq(from=range(dat$x.2)[1],to=range(dat$x.2)[2],length=n.g))
ygrid = predict(svmfit,xgrid)

plot(xgrid,col=c("red","blue")[as.numeric(ygrid)],pch=20,cex=.2)
points(x,col=c("red","blue")[as.numeric(dat$y)],pch=19)
points(x[svmfit$index,],pch=5,cex=2)


plot(x,col=c("red","blue")[as.numeric(dat$y)],pch=19)
contour(seq(from=range(dat$x.1)[1],to=range(dat$x.1)[2],length=n.g),
        seq(from=range(dat$x.2)[1],to=range(dat$x.2)[2],length=n.g),
        matrix(as.numeric(ygrid),n.g,n.g),level=1.5,add=TRUE,
        drawlabels=F)
```

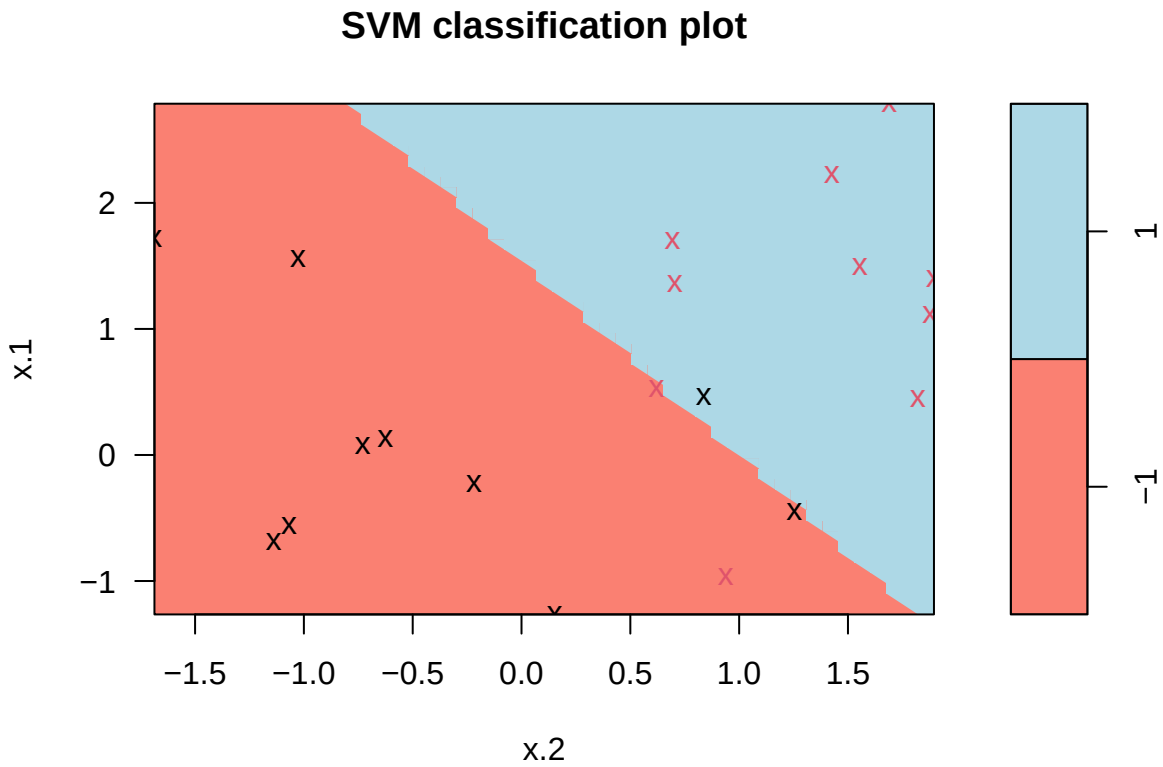


7.5.3 Changing the cost parameter

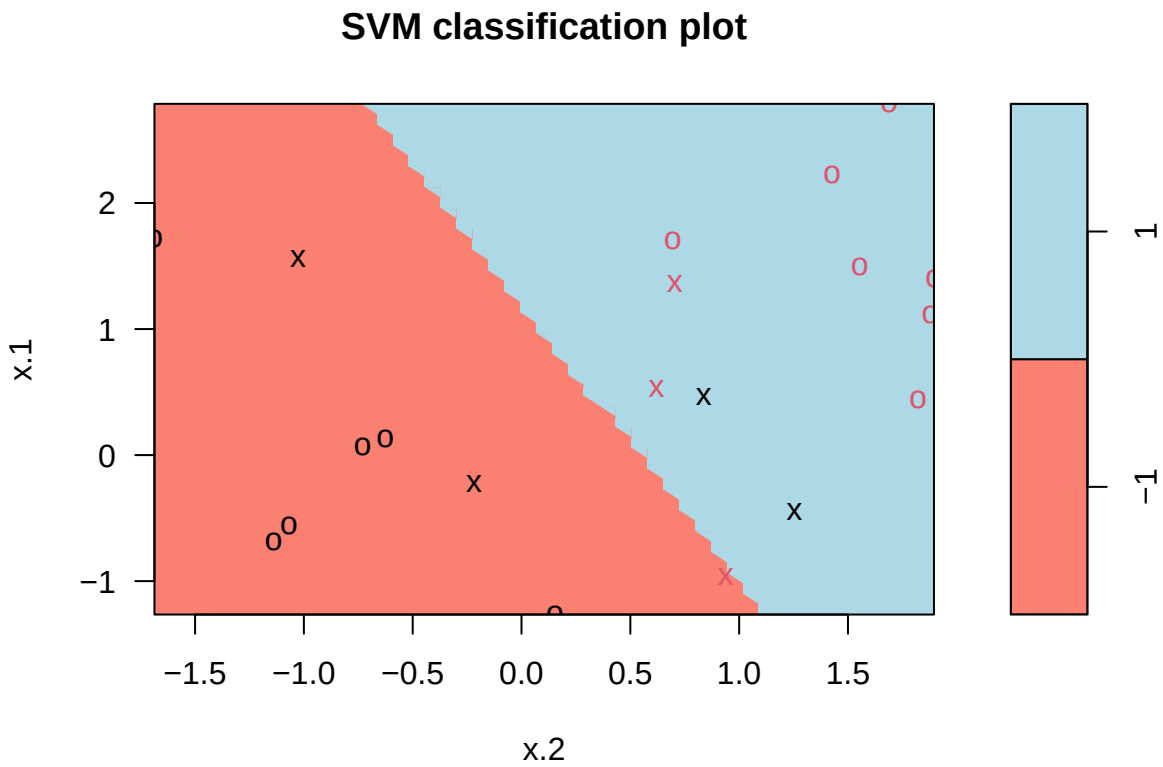
```
# If we try to change the cost parameter we get more support points
# (higher bias, lower variance)
svmfit = svm(y~., data=dat , kernel ='linear', cost =0.1, scale =FALSE )

plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```




```
#! Lower cost will give more SV (more crosses) and a boundary closer to a point,
#! as we impose to have a smaller error

#! cost=0.01, lots of crosses
plot(svm(y~., data=dat , kernel ='linear', cost =0.01, scale =FALSE ),
     dat, col =c('salmon', 'light blue'), pch=19)
```




```
#! cost=20, fewer crosses
plot(svm(y~., data=dat , kernel ='linear', cost =20, scale =FALSE ),
     dat, col =c('salmon', 'light blue'), pch=19)
```



```r
#! - smaller cost: we look at more points
#! - higher cost: we look at points just close to the boundary
#! So also higher cost is less sensible to outliers (we "skip" them)
```

7.5.4 Tune to find the best cost

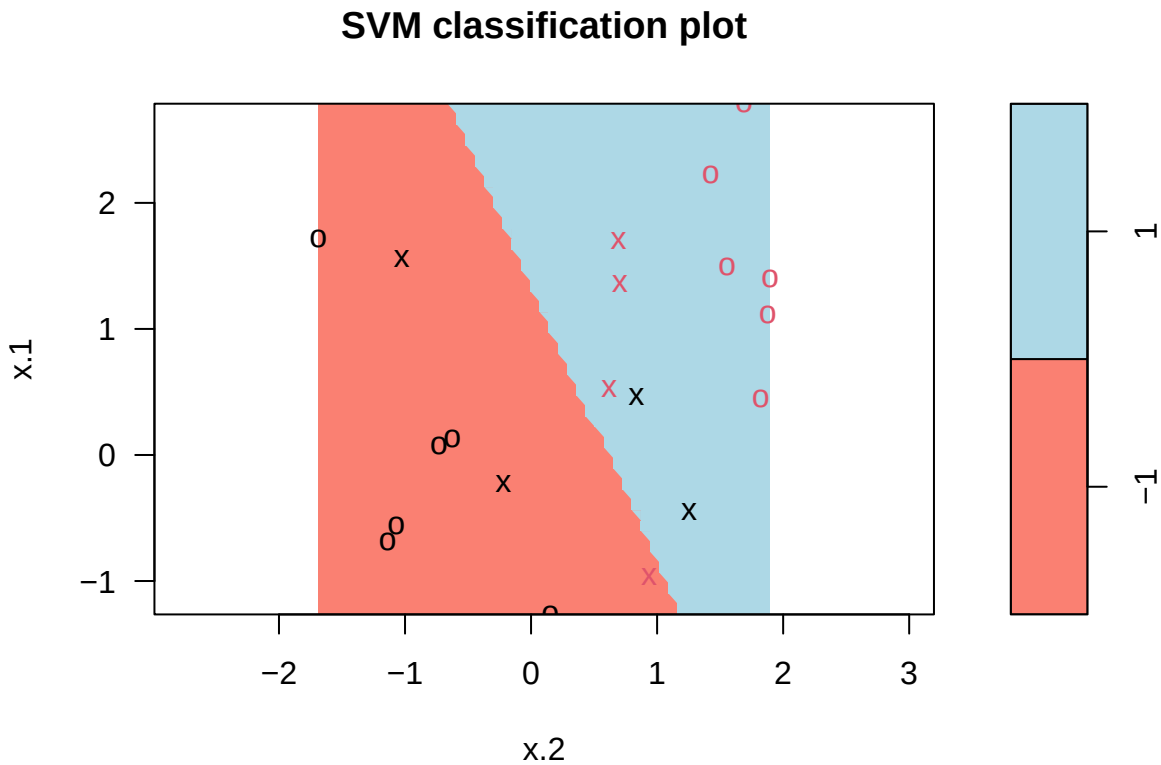
```r
# To set the parameter C we can use the function tune(),
# which is based on cross-validation (10-fold)
set.seed (1)
tune.out = tune(svm,y~.,data=dat ,kernel = 'linear',
                ranges =list(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.15
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.20  0.2581989
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.20  0.2581989
## 6 1e+01  0.20  0.2581989
## 7 1e+02  0.20  0.2581989
```

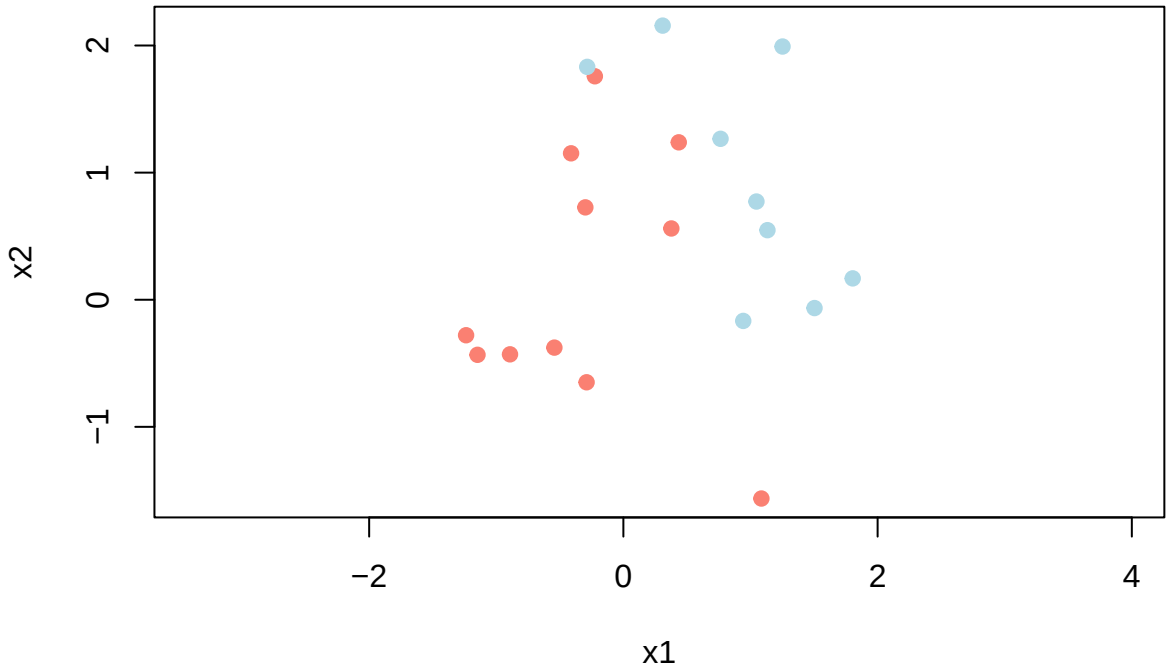
```r
# Extract the best model from the result of tune
bestmod = tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  8
##
##  ( 4 4 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
plot(bestmod , dat, col =c('salmon', 'light blue'), pch=19, asp=1)
```




```
# Prediction for a new observation (command predict())
xtest = matrix(rnorm (20*2) , ncol =2)
ytest = sample(c(-1,1) , 20, rep=TRUE)
xtest[ytest ==1 ,] = xtest[ytest ==1,] + 1
testdat = data.frame(x=xtest , y=as.factor (ytest))

plot(xtest, col =ifelse(ytest==1, 'light blue', 'salmon'),
     pch=19, xlab='x1', ylab='x2', asp=1)
```

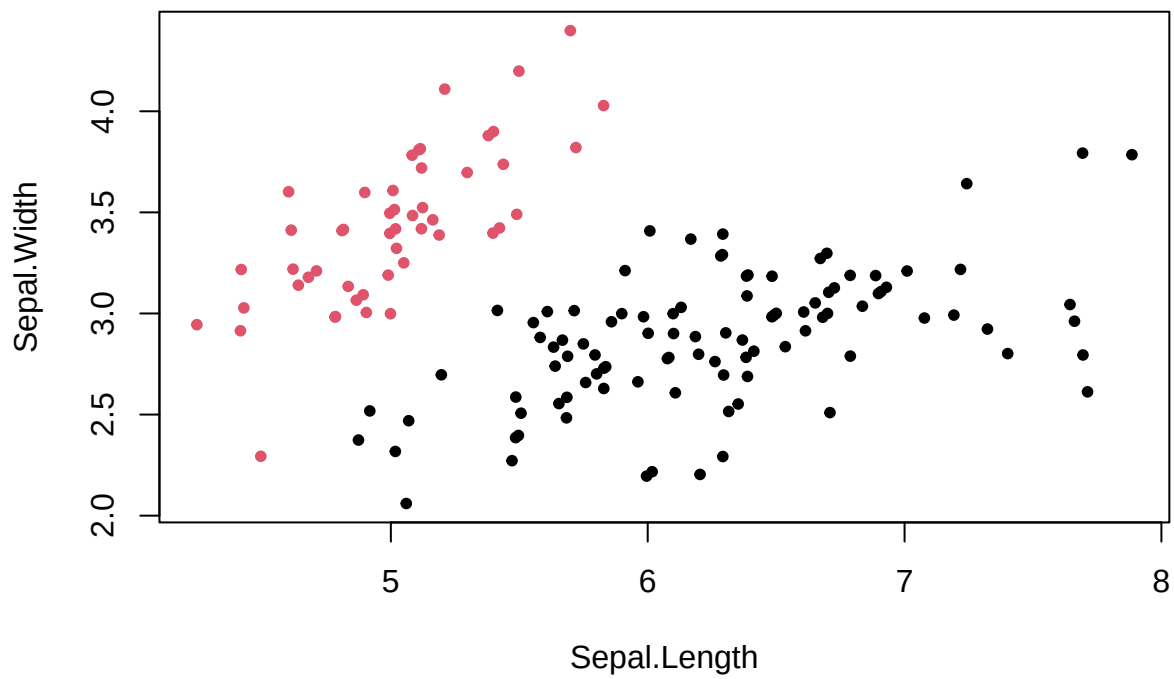



```
ypred = predict(bestmod,testdat)
table(true.label=testdat$y, assigned.label =ypred )
```

```
##           assigned.label
## true.label -1 1
##         -1  6 5
##         1   1 8
```

7.5.5 On iris dataset

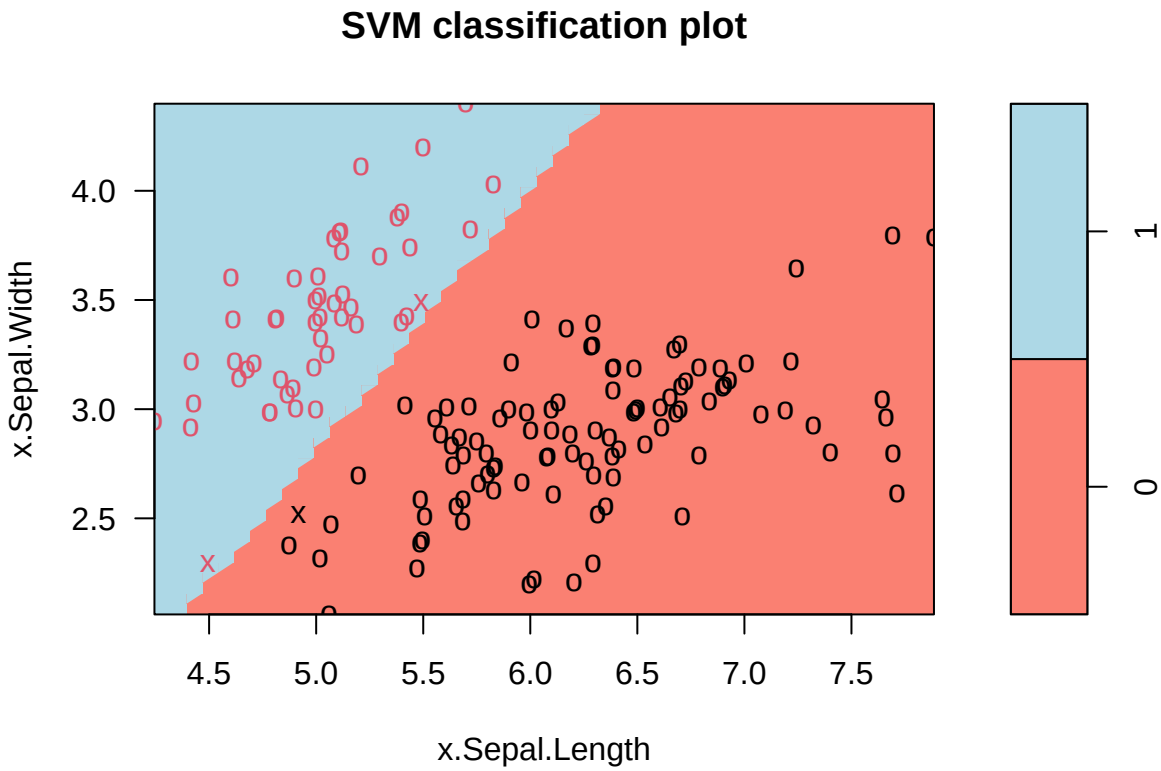
```
# If the classes are separable, setting a high value for the cost function
# leads to the maximal margin classifier (ie, it returns the classification
# provided by the best separating hyperplane)

species.name = factor(iris$Species, labels=c('setosa','versicolor','virginica'))
set.seed(1)
iris2 = iris[,1:2] + cbind(rnorm(150, sd=0.025))    # jittering

# setosa VS versicolor+virginica
y = rep(0,150)
y[which(species.name=='setosa')] = 1


plot(iris2[,1], iris2[,2], xlab='Sepal.Length', ylab='Sepal.Width',
     pch=20, col=as.character(y+1))
```




```
dat = data.frame(x=iris2[,c(2,1)], y=as.factor (y))
svmfit = svm(y~., data=dat , kernel ='linear', cost =100, scale =FALSE )
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 100, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##
## Number of Support Vectors:  3
##
##  ( 2 1 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

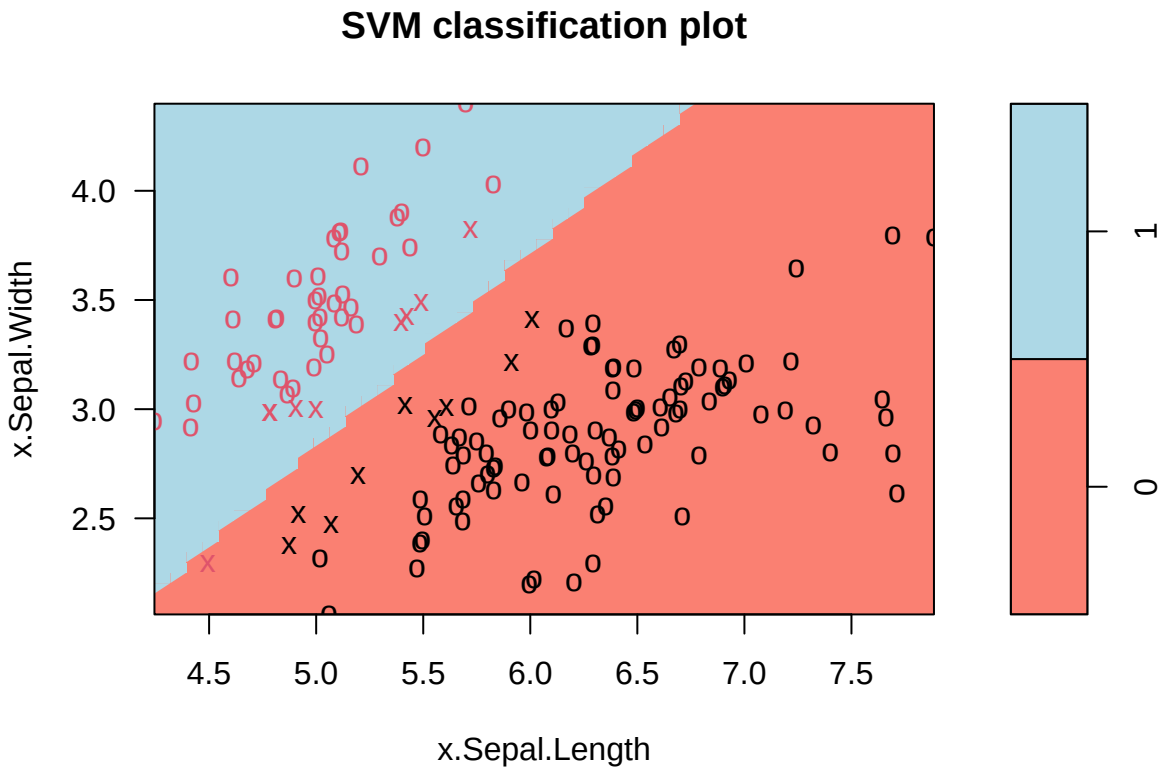
```
par(mfrow=c(1,2))
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19)
```




```
dat = data.frame(x=iris2[,c(2,1)], y=as.factor (y))
svmfit = svm(y~., data=dat , kernel ='linear', cost =1, scale =FALSE )
summary(svmfit)
```

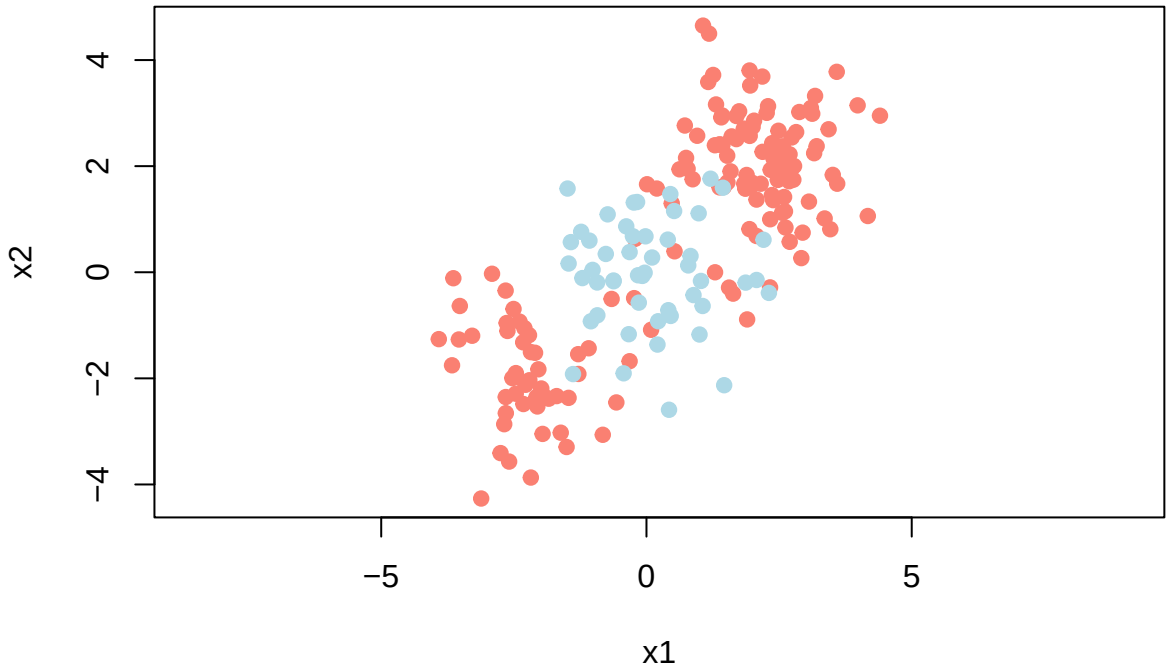
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  18
##
##  ( 9 9 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
par(mfrow=c(1,2))
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19)
```



7.5.6 Non linear case

```
### Non linear case

#! To achieve non linearity we can still use linear methods, but on transformed data.
#! But in SVM we can let it do all the work by changing the kernel, ie changing how it
#! is the defined the scalar product (ie the similarity function) between our points

#! Like also we could work on a large dataset with mul

# Generate the data
set.seed (1)
x = matrix (rnorm (200*2) , ncol =2)
x[1:100 ,] = x[1:100 ,]+2
x[101:150 ,] = x[101:150 ,] -2
y = c(rep (1 ,150) ,rep (2 ,50) )
dat = data.frame(x=x,y=as.factor (y))

# The classes are not separable

plot(x, col =ifelse(y==1, 'salmon', 'light blue'), pch=19, xlab='x1', ylab='x2', asp=1)
```



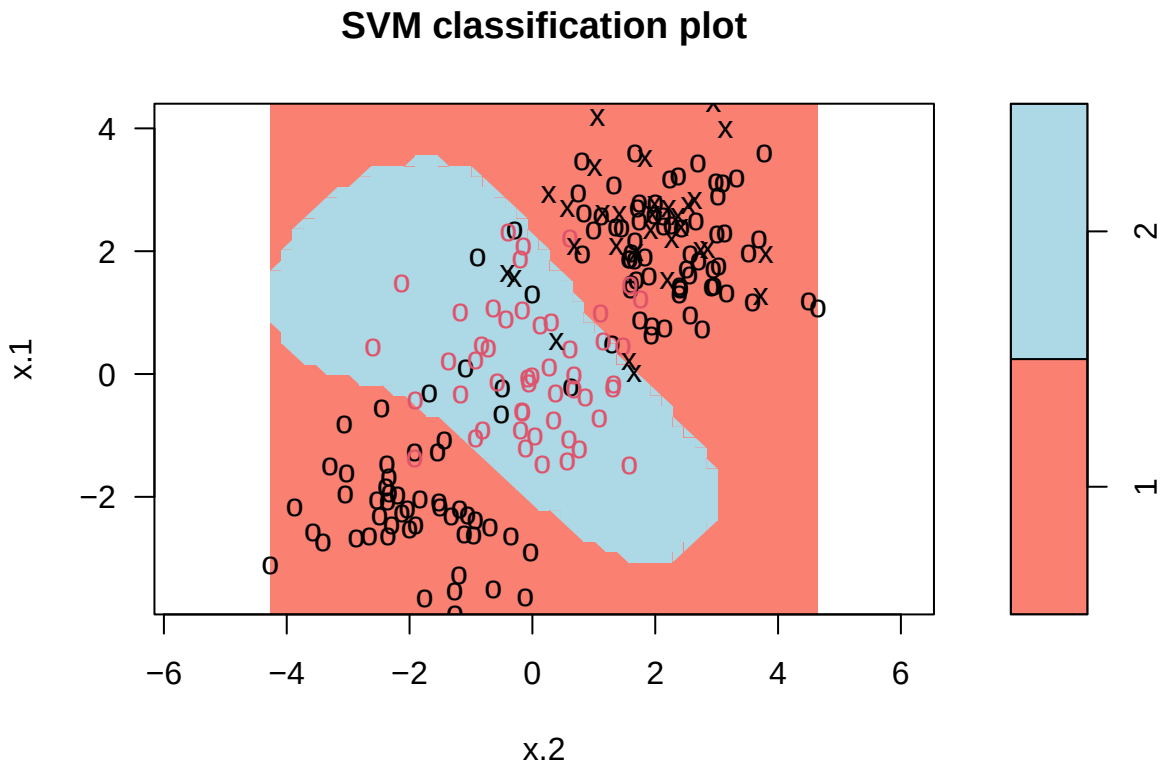

```
# Randomly split in train and test
train = sample (200 ,100)
```

7.5.7 Fit the model

```
# Fit a Support Vector Machine (kernel = "radial") given a cost C
svmfit = svm(y~., data=dat [train ,], kernel ='radial', gamma =1, cost =1)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

```
# Plot the SVM
plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1, main="cost 1")
```




```
# Misclassification error on the training set
table(true=dat[train ,"y"], pred=predict (svmfit ,
                                          newdata =dat[train ,]))
```

```
##     pred
## true  1  2
##    1 69  4
##    2  3 24
```

```
(3+4)/100
```

```
## [1] 0.07
```

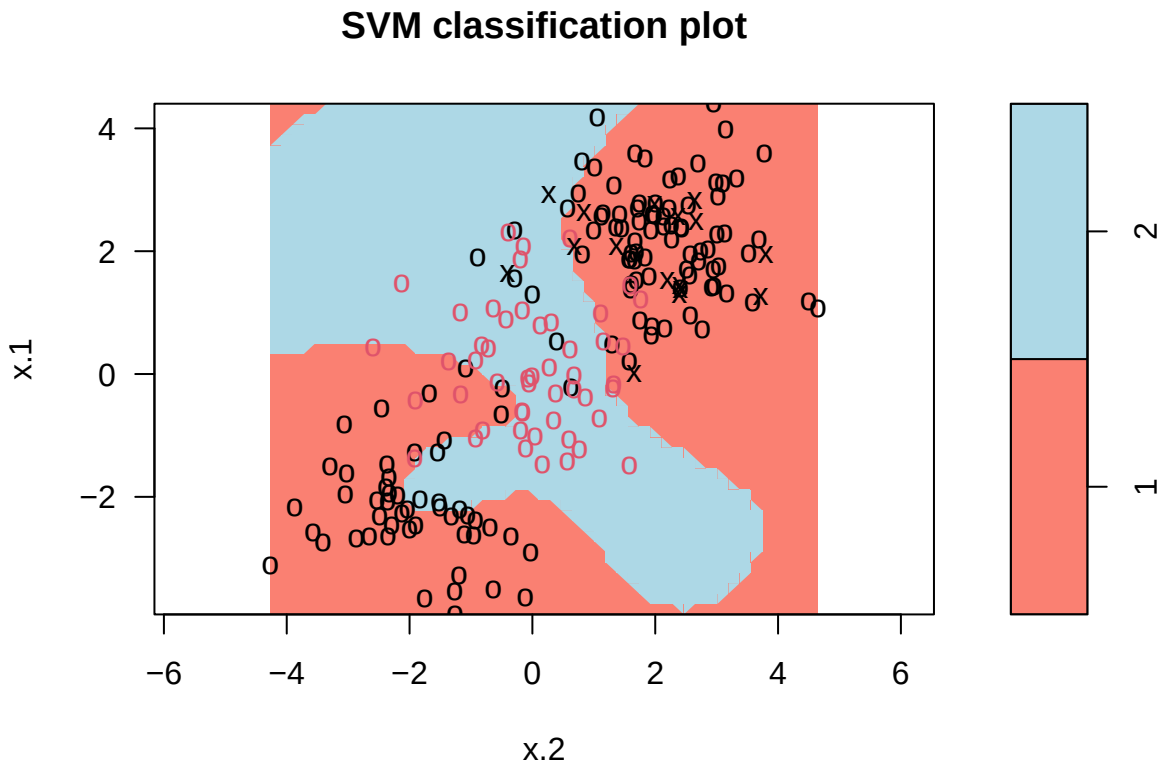
```
# Misclassification error on the test set
table(true=dat[-train ,"y"], pred=predict (svmfit ,
                                           newdata =dat[-train ,]))
```

```
##     pred
## true  1  2
##    1 67 10
##    2  3 20
```

```
(3+10)/100
```

```
## [1] 0.13
```

```
# Increasing the cost decreases the errors on the training set,
# at the expense of a more irregular boundary
svmfit = svm(y~., data=dat [train ,], kernel ='radial',gamma =1,cost=1e5)


plot(svmfit , dat, col =c('salmon', 'light blue'), pch=19, asp=1, main="cost 1e5")
```




```
# Misclassification error on the training set
table(true=dat[train ,"y"], pred=predict (svmfit ,
                                          newdata =dat[train ,]))
```

```
##     pred
## true  1  2
##    1 73  0
##    2  0 27
```

```
0
```

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

```
# Misclassification error on the test set
table(true=dat[-train ,"y"], pred=predict (svmfit ,
                                           newdata =dat[-train ,]))
```

```
##     pred
## true  1  2
##    1 57 20
##    2  5 18
```

```
(5+20)/100
```

```
## [1] 0.25
```

```
# Set parameters via CV:
set.seed (1)
tune.out = tune(svm , y~., data=dat[train ,], kernel ='radial',
                ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),
                             gamma=c(0.5,1,2,3,4) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.07
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.26 0.15776213
## 2  1e+00   0.5  0.07 0.08232726
## 3  1e+01   0.5  0.07 0.08232726
## 4  1e+02   0.5  0.14 0.15055453
## 5  1e+03   0.5  0.11 0.07378648
## 6  1e-01   1.0  0.22 0.16193277
## 7  1e+00   1.0  0.07 0.08232726
## 8  1e+01   1.0  0.09 0.07378648
## 9  1e+02   1.0  0.12 0.12292726
## 10 1e+03   1.0  0.11 0.11005049
## 11 1e-01   2.0  0.27 0.15670212
## 12 1e+00   2.0  0.07 0.08232726
## 13 1e+01   2.0  0.11 0.07378648
## 14 1e+02   2.0  0.12 0.13165612
## 15 1e+03   2.0  0.16 0.13498971
## 16 1e-01   3.0  0.27 0.15670212
## 17 1e+00   3.0  0.07 0.08232726
## 18 1e+01   3.0  0.08 0.07888106
## 19 1e+02   3.0  0.13 0.14181365
## 20 1e+03   3.0  0.15 0.13540064
## 21 1e-01   4.0  0.27 0.15670212
## 22 1e+00   4.0  0.07 0.08232726
## 23 1e+01   4.0  0.09 0.07378648
## 24 1e+02   4.0  0.13 0.14181365
## 25 1e+03   4.0  0.15 0.13540064
```

```r
# Misclassification error with best model on the training set
table(true=dat[train ,"y"], pred=predict (tune.out$best.model ,
                                          newdata =dat[train ,]))
```

```
##     pred
## true  1  2
##    1 69  4
##    2  2 25
```

```r
(2+4)/100
```

```
## [1] 0.06
```

```r
# Misclassification error with best model on the test set
table(true=dat[-train ,"y"], pred=predict (tune.out$best.model ,
                                           newdata =dat[-train ,]))
```

```
##     pred
## true  1  2
##    1 67 10
##    2  2 21
```

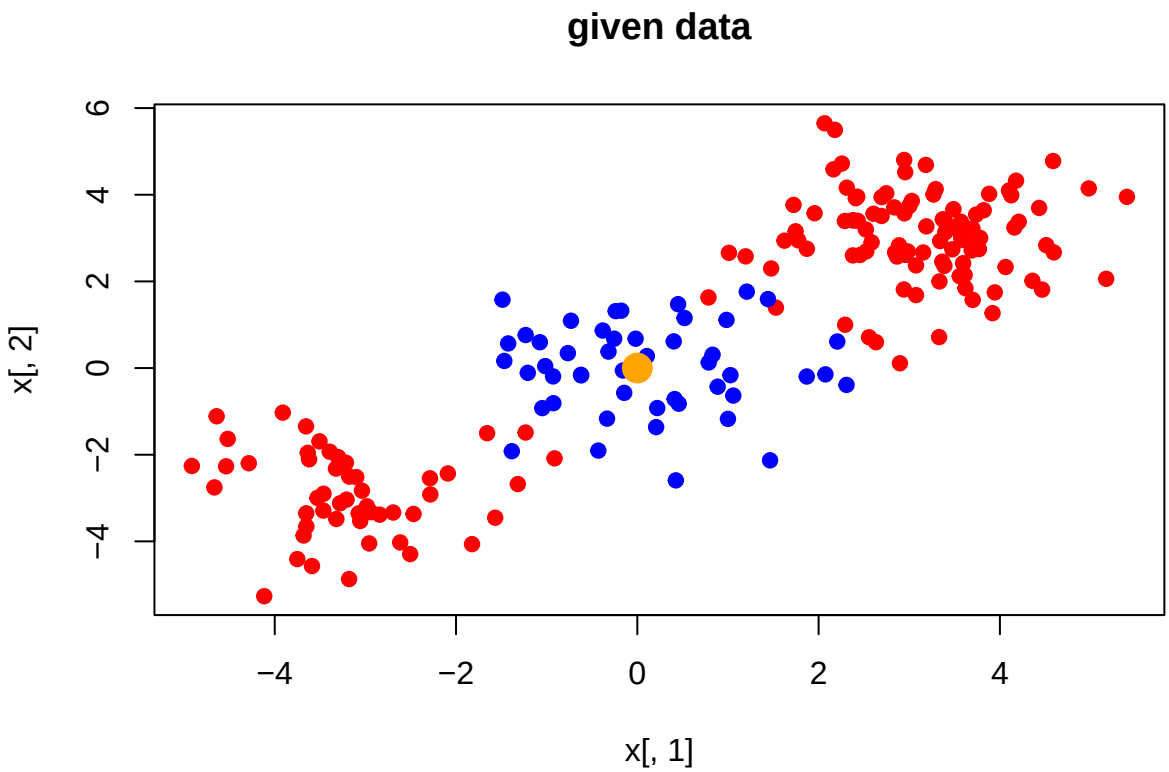
```r
(2+10)/100
```

```
## [1] 0.12
```

7.6 Trying polar transformation

```r
# Generate the data
set.seed (1)
x = matrix (rnorm (200*2) , ncol=2)
x[1:100 ,] = x[1:100 ,]+3
x[101:150 ,] = x[101:150 ,]-3
y = c(rep (1 ,150) ,rep (2 ,50) )
```

```r
dat = data.frame(x=x,y=as.factor (y))

# Polar transform (attempt)
origin = c(0,0)
plot(x[,1],x[,2],col=ifelse(y==1, 'red', 'blue'),pch=19, main="given data")
points(origin[1],origin[2],col="orange",pch=20,cex=3)
```



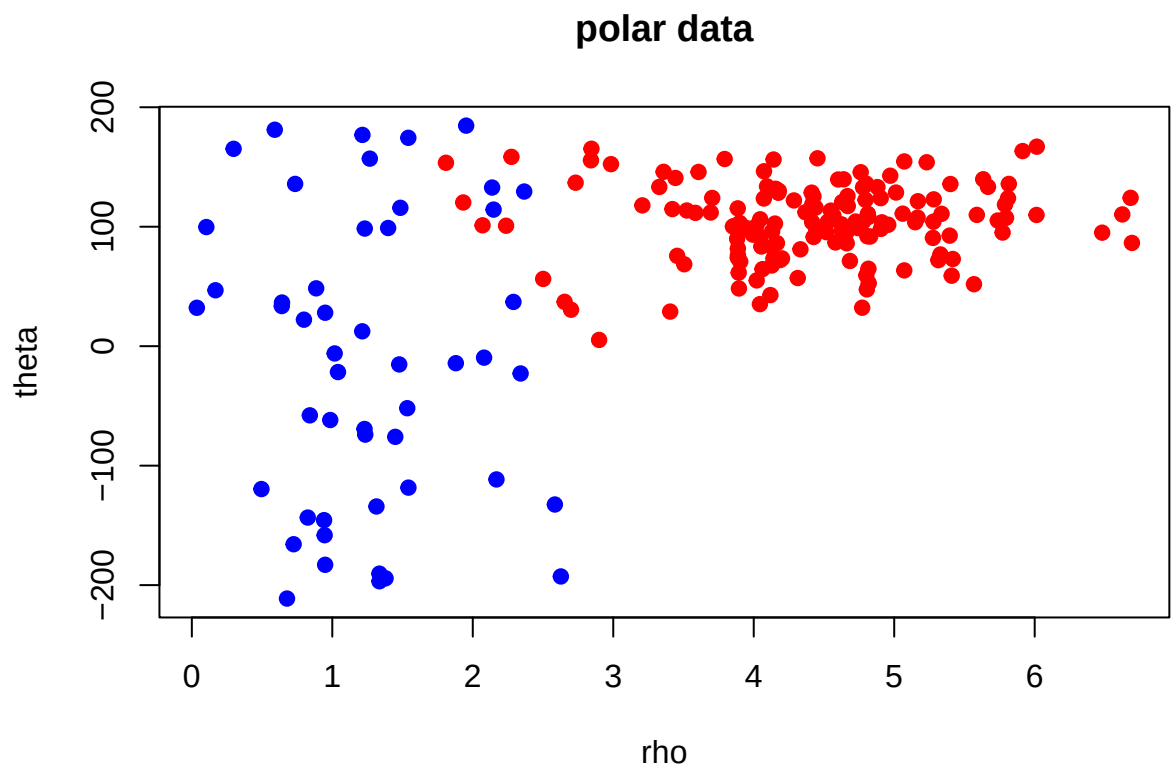

```r
head(dat)
```

```
##        x.1       x.2 y
## 1 2.373546 3.4094018 1
## 2 3.183643 4.6888733 1
## 3 2.164371 4.5865884 1
## 4 4.595281 2.6690922 1
## 5 3.329508 0.7147645 1
## 6 2.179532 5.4976616 1
```

```r
df_pol=dat
for(i in 1:dim(df_pol)[1]){
    # df_pol[i,1]=sqrt((dat[i,1]-center[1])^2+(dat[2,1]-center[2])^2)
    # df_pol[i,2]=acos((dat[i,1]*center[1]+dat[i,2]*center[2])/
                        # (norm(t(x[i,1:2]))*norm(as.matrix(center))))
    df_pol[i,1] = sqrt(dat[i,1]^2+dat[i,2]^2)
    df_pol[i,2] = atan(dat[i,2]/dat[i,1])*215 / (pi/2)
}
head(df_pol)
```

```
##        x.1       x.2 y
## 1 4.154244 131.75979 1
## 2 5.667550 133.35816 1
## 3 5.071617 154.65137 1
## 4 5.314194  72.02368 1
## 5 3.405365  28.94406 1
## 6 5.913936 163.33860 1
```

```
plot(df_pol[,1:2],col=ifelse(y==1, 'red', 'blue'),pch=19, xlab="rho",ylab="theta",
     main="polar data")
```




```
labeled1 = df_pol[which(df_pol[,3]==1),]
labeled2 = df_pol[which(df_pol[,3]==2),]
```

7.6.1 Example on multiclass classification

```
### Application to Gene Expression Data (multiclass classification)
#! Example of a dataset with small n, large p. So having lots of parameters may
#! induce a risk of overfit (seeable by 0/very small empirical error)
library (ISLR)

# help(Khan)

is(Khan)
```

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

```
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```
dim(Khan$xtest)
```

```
## [1]   20 2308
```

```
length (Khan$ytrain)
```

```
## [1] 63
```

```r
length (Khan$ytest)
```

```
## [1] 20
```

```r
table(Khan$ytrain)
```

```
##
##  1  2  3  4
##  8 23 12 20
```

```r
table(Khan$ytest)
```

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

```r
dat = data.frame(x=Khan$xtrain , y=as.factor(Khan$ytrain))
out = svm(y~., data=dat , kernel ="linear",cost =10)
summary (out)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  58
##
##  ( 20 20 11 7 )
##
##
## Number of Classes:  4
##
## Levels:
##  1 2 3 4
```

```r
table(out$fitted , dat$y)
```

```
##
##      1  2  3  4
##   1  8  0  0  0
##   2  0 23  0  0
##   3  0  0 12  0
##   4  0  0  0 20
```

```r
dat.te = data.frame(x=Khan$xtest , y=as.factor (Khan$ytest ))
pred.te = predict (out , newdata =dat.te)
table(pred.te , dat.te$y)
```

```
##
## pred.te 1 2 3 4
##       1 3 0 0 0
##       2 0 6 2 0
##       3 0 0 4 0
##       4 0 0 0 5
```

8 Lab 08 - Clustering (HClust, KMeans, DBScan)

```r
###--------------------###
### LAB 8 (28/04/2023) ###
###--------------------###

### TOPICS:
```

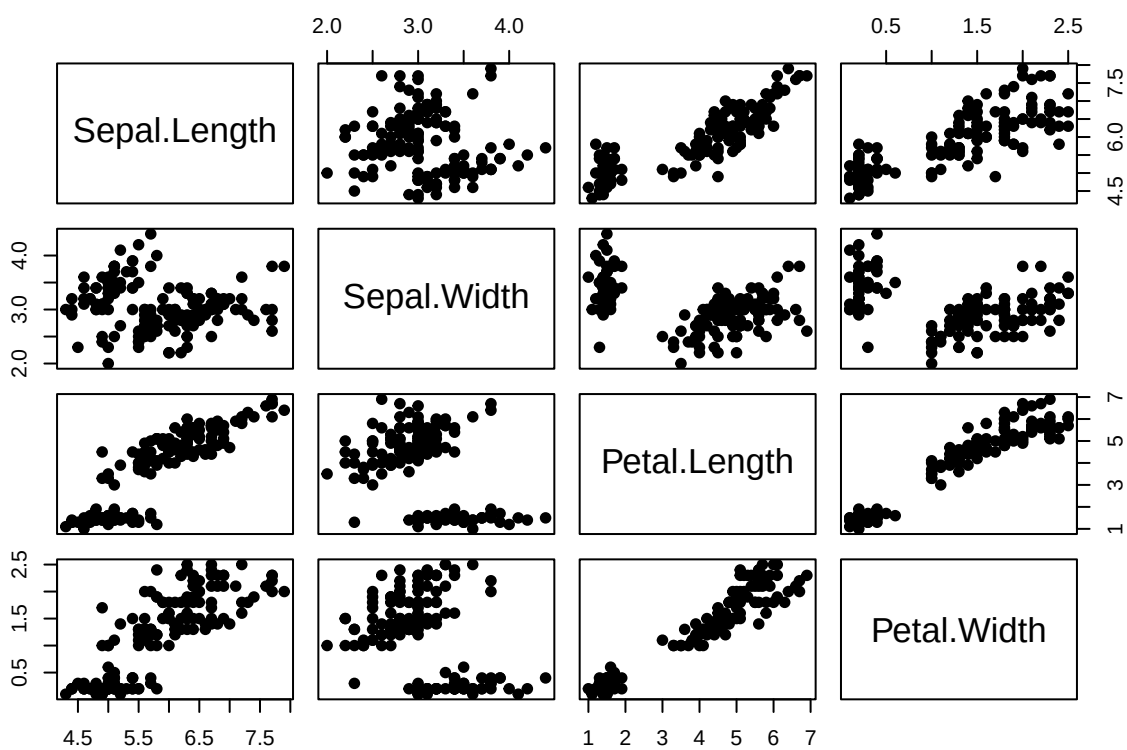
```r
### Hierarchical clustering
### K-means clustering
### DBSCAN
### Exercises
### Multidimensional Scaling

windows.options(height=6.5, width=11, reset=FALSE)
# options(rgl.printRglwidget = TRUE)

library(mvtnorm)
library(MVN)
library(rgl)
library(car)
library(dbscan)
library(cluster)
library(fields)
```

8.1 On iris dataset

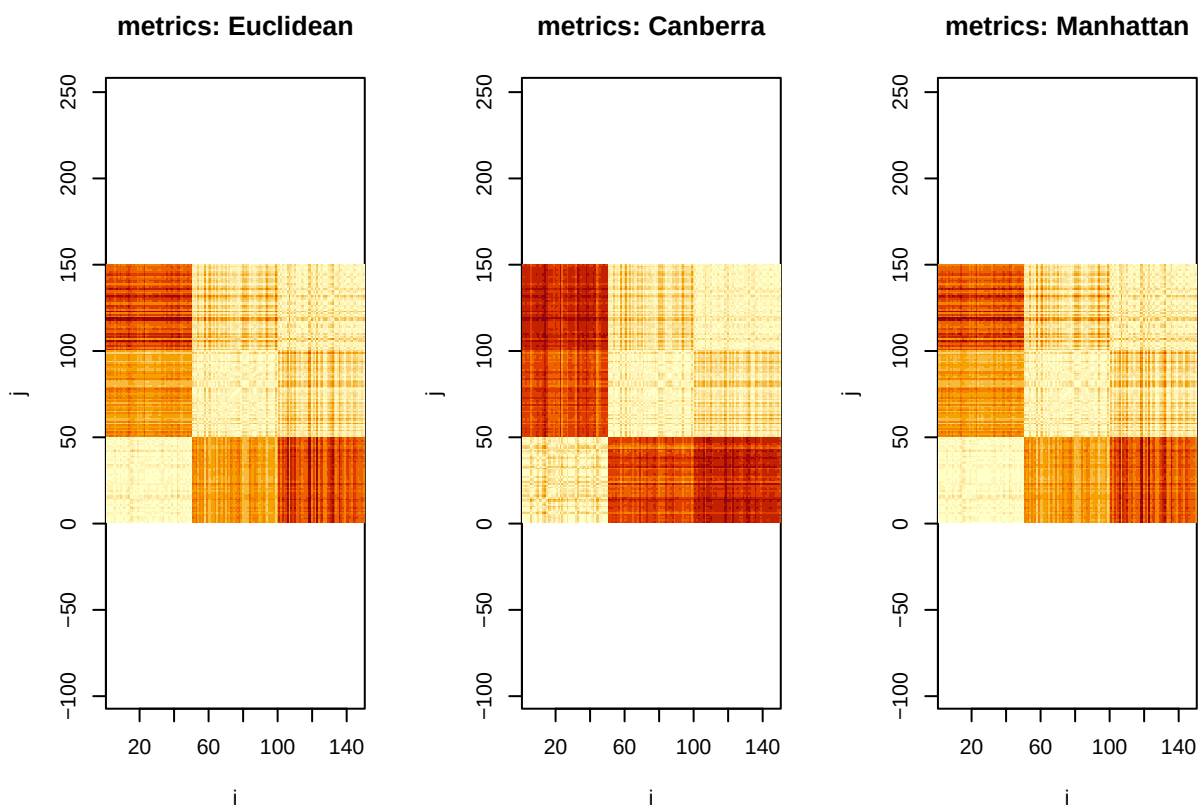
```r
#-------------------------------------------------------------------------------
##### Hierarchical Clustering

##### Example 1: iris dataset
#####-------------------------

# let's forget the labels (we perform a cluster analysis, not a discriminant one)

species.name = iris[,5]
iris4        = iris[,1:4]
```

8.1.1 Dissimilarity matrix

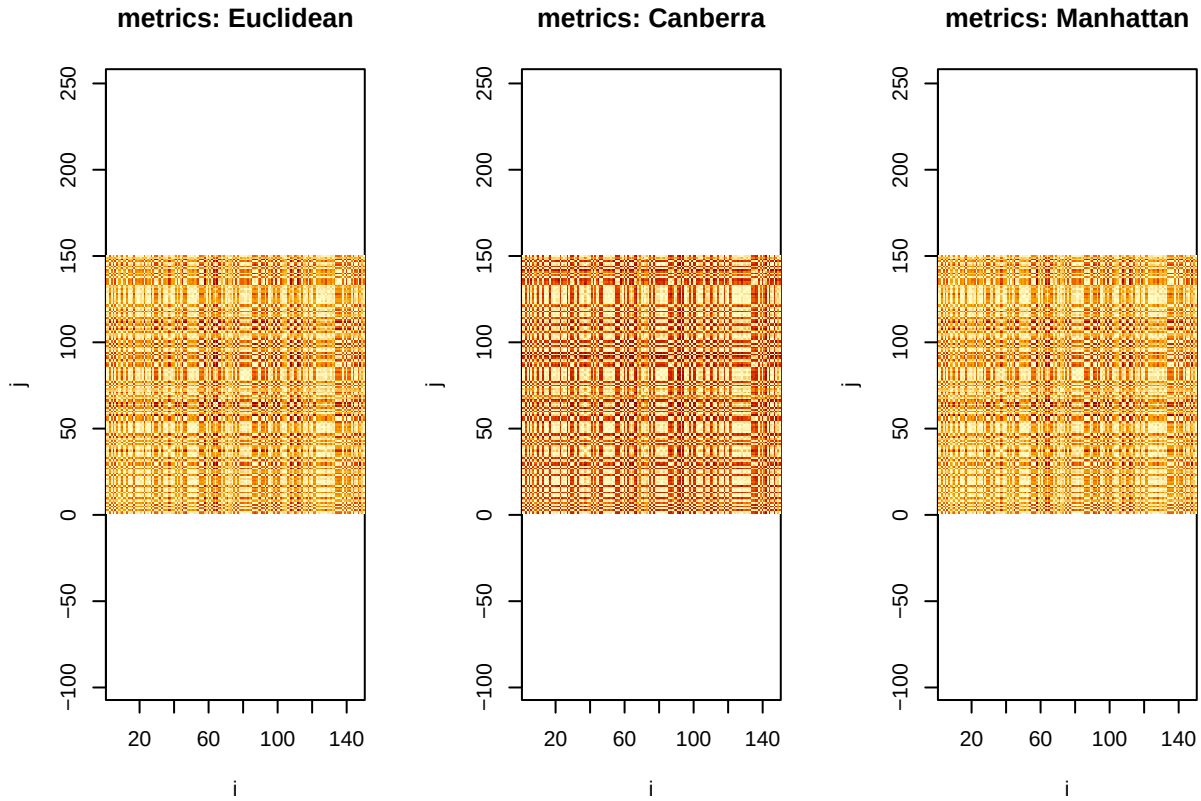
```r
pairs(iris4, pch=19)
```



```r
# compute the dissimilarity matrix of the data
# we choose the Euclidean metric (and then we look at other metrics)
# help(dist)
iris.e = dist(iris4, method='euclidean')

image(1:150,1:150,as.matrix(iris.e), main='metrics: Euclidean', asp=1, xlab='i', ylab='j')
#! We see clear objects, as obs are already divided into groups in the iris dataset
#! So we can see already a 3 clustering structure, well, because the dataset obs are
#! already ordered (all Setosa, then all Virginica, ecc). So maybe with other datasets
#! we need to do some permutation to better see a possible clustering

# with other metrics:
iris.m = dist(iris4, method='manhattan')
iris.c = dist(iris4, method='canberra')

par(mfrow=c(1,3))
image(1:150,1:150,as.matrix(iris.e), main='metrics: Euclidean', asp=1, xlab='i', ylab='j' )
image(1:150,1:150,as.matrix(iris.c), main='metrics: Canberra', asp=1, xlab='i', ylab='j' )
image(1:150,1:150,as.matrix(iris.m), main='metrics: Manhattan', asp=1, xlab='i', ylab='j' )
```




```r
# actually, the data are never ordered according to (unknown) labels
misc = sample(150)
iris4 = iris4[misc,]

iris.e = dist(iris4, method='euclidean')
iris.m = dist(iris4, method='manhattan')
iris.c = dist(iris4, method='canberra')


par(mfrow=c(1,3))
image(1:150,1:150,as.matrix(iris.e), main='metrics: Euclidean', asp=1, xlab='i', ylab='j' )
image(1:150,1:150,as.matrix(iris.c), main='metrics: Canberra', asp=1, xlab='i', ylab='j' )
image(1:150,1:150,as.matrix(iris.m), main='metrics: Manhattan', asp=1, xlab='i', ylab='j' )
```




```
#! These plots are the usual one we see. Before was because we already had the obs splitted
#! into groups and ordered, in the dataset. Here we shuffled them instead


# we now aim to perform hierarchical clustering of the dataset iris
# we limit to Euclidean distance

# Command hclust()
# help(hclust)
```

8.1.2 Hierarchical clustering

```
#! hclust takes a distance object (not a dataset) and the method for the linkage
iris.es = hclust(iris.e, method='single')
iris.ea = hclust(iris.e, method='average')
iris.ec = hclust(iris.e, method='complete')

# if we want more detailed information on euclidean-complete clustering:
names(iris.ec)
```

```
## [1] "merge"       "height"      "order"       "labels"      "method"
## [6] "call"        "dist.method"
```

```
iris.ec$merge  # order of aggregation of statistical units / clusters
```

```
##        [,1] [,2]
##   [1,]  -42 -150
##   [2,]  -77 -118
##   [3,]  -33 -106
##   [4,]  -60 -144
##   [5,]   -6 -128
##   [6,]  -55 -137
##   [7,]  -92 -101
##   [8,]   -3  -88
##   [9,]   -5  -23
##  [10,]  -31  -44
##  [11,]  -73 -149
```

```
##  [12,]  -85 -146
##  [13,]  -47  -66
##  [14,]   -8  -52
##  [15,]  -25 -105
##  [16,]  -29 -110
##  [17,]  -53  -82
##  [18,]  -56 -100
##  [19,]  -98 -134
##  [20,]  -26 -127
##  [21,]  -70  -84
##  [22,]  -86    3
##  [23,]  -20 -115
##  [24,]  -46    2
##  [25,]  -83 -119
##  [26,]  -48 -129
##  [27,]  -96   14
##  [28,]  -11  -15
##  [29,]   -7 -122
##  [30,]  -41    4
##  [31,]  -27   19
##  [32,]  -39 -125
##  [33,]  -78 -116
##  [34,]   -1  -24
##  [35,]  -90 -145
##  [36,]  -16 -109
##  [37,] -123 -133
##  [38,]  -13    8
##  [39,]  -19   20
##  [40,]  -38  -71
##  [41,]  -62   16
##  [42,]  -64   12
##  [43,]  -32  -79
##  [44,]  -35 -103
##  [45,] -104   10
##  [46,]   22   24
##  [47,]  -61    1
##  [48,]  -69   29
##  [49,] -131 -143
##  [50,]  -12  -89
##  [51,]  -45  -99
##  [52,] -108 -112
##  [53,]  -34   15
##  [54,] -121 -139
##  [55,]  -57    9
##  [56,]  -87   41
##  [57,]  -49   17
##  [58,]   30   31
##  [59,]   18   46
##  [60,] -102   35
##  [61,]  -59 -130
##  [62,]  -76    5
##  [63,]  -14  -95
##  [64,]  -18   47
##  [65,]  -30   56
##  [66,]  -43 -111
##  [67,]   -2   49
##  [68,]   26   27
##  [69,]  -17 -142
##  [70,]  -67   13
##  [71,] -147    6
##  [72,] -113 -140
##  [73,] -138   57
##  [74,]    7   59
##  [75,]  -81   37
##  [76,]   -4  -22
##  [77,]  -80   34
##  [78,] -107   11
##  [79,] -120   39
##  [80,]  -36 -124
##  [81,]   40   60
##  [82,] -141   38
##  [83,]  -63  -65
##  [84,]  -51   63
##  [85,]  -75   55
##  [86,]   54   71
```

```
##  [87,]   33   45
##  [88,]   23   84
##  [89,]  -68  -72
##  [90,]  -10   58
##  [91,] -135   69
##  [92,]  -50   50
##  [93,]   53   68
##  [94,]   25   43
##  [95,]   21   75
##  [96,]  -28   64
##  [97,]   28   61
##  [98,]   42   62
##  [99,]  -93   72
## [100,]   32   79
## [101,]  -58   52
## [102,] -114 -136
## [103,]  -74   67
## [104,]   51   99
## [105,]   77   94
## [106,]   48   82
## [107,]   80   93
## [108,]   76   88
## [109,]   66   86
## [110,]   65   70
## [111,] -132   87
## [112,]  -97   36
## [113,]   74   85
## [114,]   95  103
## [115,]   91  102
## [116,] -126   44
## [117,]   -9   78
## [118,]  -54   98
## [119,]   90  113
## [120,]   92  100
## [121,]   89  111
## [122,]   73   96
## [123,]   81  112
## [124,]  -37 -148
## [125,]  106  109
## [126,]  -94  110
## [127,]   97  107
## [128,] -117  108
## [129,]   83  124
## [130,]  -40  121
## [131,]  105  118
## [132,]  114  120
## [133,]  119  126
## [134,]  122  128
## [135,]  123  131
## [136,]  127  130
## [137,]  115  125
## [138,]  101  129
## [139,]  -91  133
## [140,]  116  132
## [141,]  104  138
## [142,]  -21  136
## [143,]  134  140
## [144,]  117  142
## [145,]  135  143
## [146,]  137  139
## [147,]  141  145
## [148,]  144  146
## [149,]  147  148
```

```
#! Each row correspond to a merge step, which involves 2 elements. When we have a minus it
#! means that the merge occurred between single obs (otherwise for sub-clusters already
#! created, which are indicated as a number saying at which step that cluster was formed)

iris.ec$height # distance at which we have aggregations
```

```
##   [1] 0.0000000 0.1000000 0.1000000 0.1000000 0.1000000 0.1000000 0.1414214
##   [8] 0.1414214 0.1414214 0.1414214 0.1414214 0.1414214 0.1414214
##  [15] 0.1414214 0.1414214 0.1414214 0.1414214 0.1414214 0.1414214 0.1414214
##  [22] 0.1732051 0.1732051 0.1732051 0.1732051 0.1732051 0.1732051 0.2000000
```

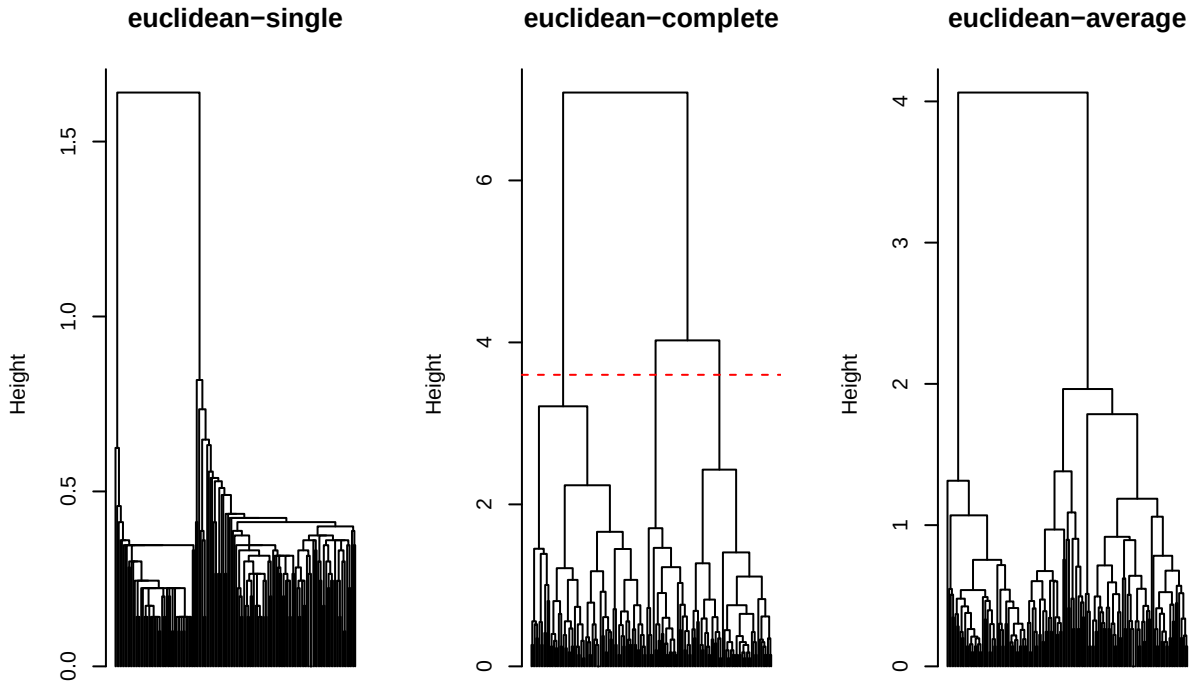
```
##  [29] 0.2000000 0.2000000 0.2000000 0.2000000 0.2000000 0.2236068 0.2236068
##  [36] 0.2449490 0.2449490 0.2449490 0.2449490 0.2449490 0.2449490 0.2449490
##  [43] 0.2449490 0.2645751 0.2645751 0.2645751 0.2645751 0.2645751 0.2645751
##  [50] 0.2645751 0.2645751 0.2645751 0.2828427 0.2828427 0.3000000 0.3000000
##  [57] 0.3000000 0.3000000 0.3000000 0.3162278 0.3162278 0.3162278 0.3316625
##  [64] 0.3316625 0.3316625 0.3316625 0.3316625 0.3316625 0.3464102 0.3464102
##  [71] 0.3464102 0.3464102 0.3605551 0.3741657 0.3741657 0.3741657 0.3872983
##  [78] 0.3872983 0.3872983 0.4000000 0.4000000 0.4123106 0.4123106 0.4242641
##  [85] 0.4242641 0.4242641 0.4358899 0.4472136 0.4472136 0.4472136 0.4582576
##  [92] 0.4582576 0.4690416 0.4690416 0.5099020 0.5196152 0.5196152 0.5196152
##  [99] 0.5196152 0.5196152 0.5477226 0.5477226 0.5567764 0.5567764 0.5567764
## [106] 0.5916080 0.6000000 0.6082763 0.6082763 0.6164414 0.6164414 0.6480741
## [113] 0.6480741 0.6782330 0.6855655 0.7071068 0.7211103 0.7348469 0.7549834
## [120] 0.7549834 0.7615773 0.7874008 0.8062258 0.8062258 0.8246211 0.8306624
## [127] 0.9055385 0.9219544 1.0049876 1.0295630 1.0677078 1.0677078 1.1090537
## [134] 1.1747340 1.2124356 1.2247449 1.2727922 1.3892444 1.4071247 1.4491377
## [141] 1.4525839 1.4628739 1.6613248 1.7058722 2.2360680 2.4289916 3.2109189
## [148] 4.0249224 7.0851958
```

```
iris.ec$order # ordering that allows to avoid intersections in the dendrogram
```

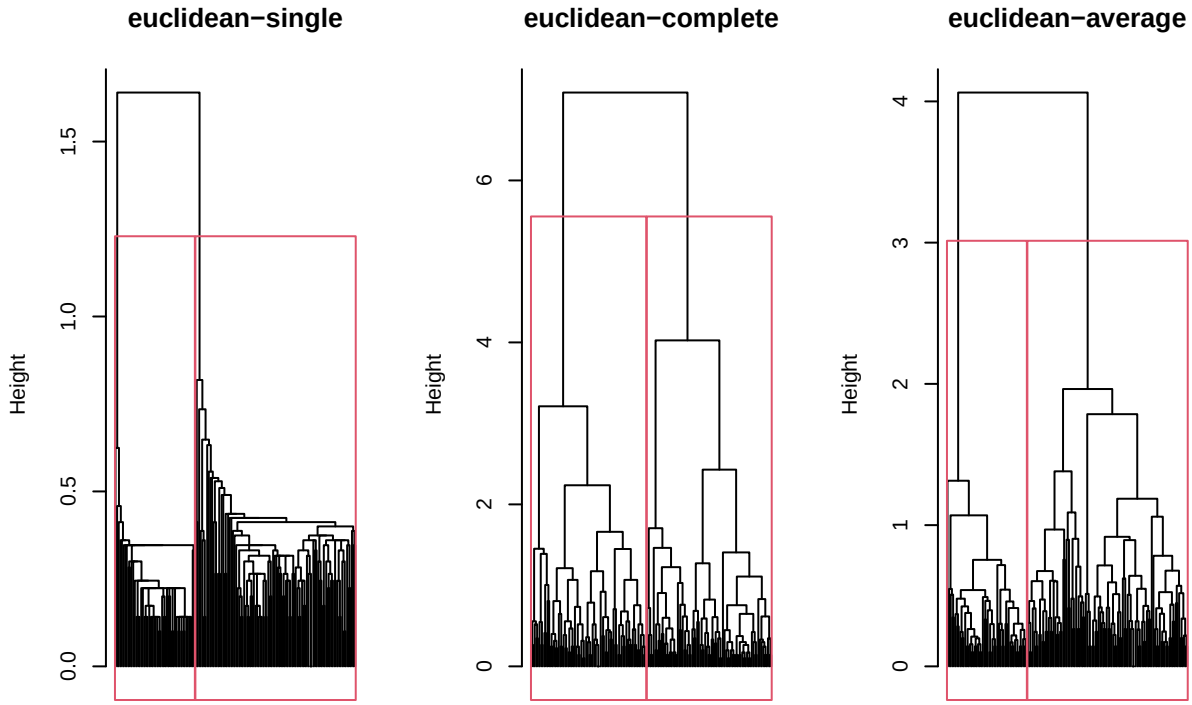
```
##   [1]  45  99  93 113 140  58 108 112  63  65  37 148  38  71 102  90 145  97
##  [19]  16 109  80   1  24  83 119  32  79  54  64  85 146  76   6 128 138  49
##  [37]  53  82  28  18  61  42 150 117   4  22  20 115  51  14  95 126  35 103
##  [55]  70  84  81 123 133  74   2 131 143  50  12  89  39 125 120  19  26 127
##  [73]   9 107  73 149  21  11  15  59 130  36 124  34  25 105  48 129  96   8
##  [91]  52  40  68  72 132  78 116 104  31  44 135  17 142 114 136  69   7 122
## [109] 141  13   3  88  43 111 121 139 147  55 137  91  10  41  60 144  27  98
## [127] 134  92 101  56 100  86  33 106  46  77 118  75  57   5  23  94  30  87
## [145]  62  29 110  67  47  66
```

8.1.3 Dendrogram plots

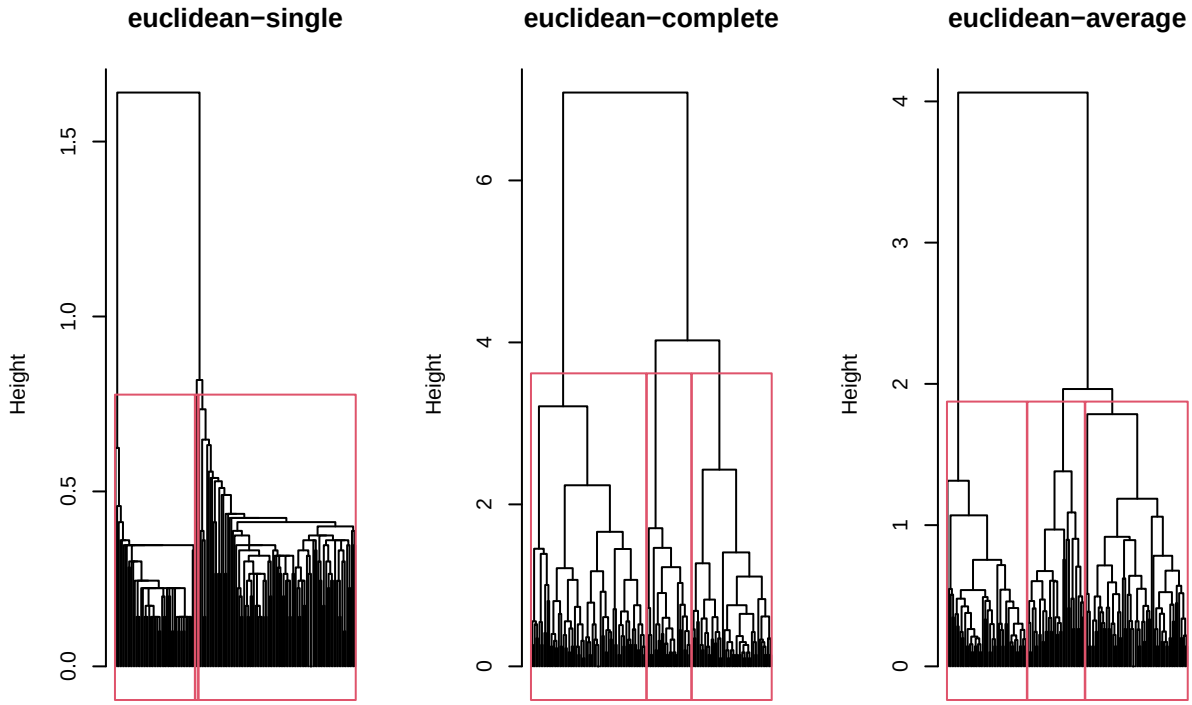
```
# plot of the dendrograms
par(mfrow=c(1,3))
#! We can call directly plot on the hclust object
plot(iris.es, main='euclidean-single', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
plot(iris.ec, main='euclidean-complete', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
abline(h=3.6, col="red",lty="dashed")
plot(iris.ea, main='euclidean-average', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
```




```
#! We want to look for inbalances in the distances at which the merge occurs
#! Single linkage will look always like that due to the chain effect

#! Interpretation:
#! - forget about the lower part, always messy
#! - we want to obtain clusters, so we have to cut the dendrogram at a certain level/height,
#!   and see which clusters we get
#! - the height of the cut can tell us how much confidence we have in saying there are those
#!   clusters: più siamo stretti meno siamo confidenti, più c'è spazio pi il cut è netto
#! Ie we would cut in places where doesnt change a lot if we move up or down


# plot dendrograms (2 clusters)
par(mfrow=c(1,3))
plot(iris.es, main='euclidean-single', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
rect.hclust(iris.es, k=2)
plot(iris.ec, main='euclidean-complete', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
rect.hclust(iris.ec, k=2)
plot(iris.ea, main='euclidean-average', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
rect.hclust(iris.ea, k=2)
```




```r
# plot dendrograms (3 clusters)
par(mfrow=c(1,3))
plot(iris.es, main='euclidean-single', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
rect.hclust(iris.es, k=3)
plot(iris.ec, main='euclidean-complete', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
rect.hclust(iris.ec, k=3)
plot(iris.ea, main='euclidean-average', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
rect.hclust(iris.ea, k=3)
```



8.1.4 Get the clusters by cutting the dendrogram

```
# How to cut a dendrogram?
# We generate vectors of labels through the command cutree()
# help(cutree)

# Fix k=2 clusters:
cluster.ec = cutree(iris.ec, k=2) # euclidean-complete
cluster.ec
```

```
## 148  87  20 112  31 133  27  97  61  36  85  52  47  84  67 137  34 122  79 124
##   1   1   2   1   2   1   2   2   2   2   2   1   2   1   2   1   2   1   1   1
## 107 147  30 111  93  92  13 115  48   7  82 142   1  68  69  62 110 141  98  63
##   2   1   2   1   2   1   2   1   2   2   2   1   2   2   1   2   1   1   1   2
##  26 143  19  81 131  50  39 100  71  86  73  96 139 109  49  28  12 119  56  35
##   2   1   2   2   1   2   2   2   1   1   1   2   1   1   2   2   2   1   2   2
## 114   3 132 104 118   9  14  80  44  76 145  65  94  78  25 105   8  54 146 116
##   1   2   1   1   1   2   2   2   2   1   1   2   2   1   2   1   2   2   1   1
##  77 128 140  66 117  41  43  22  57 144  42  38 103  23 134  89 101   2 108  29
##   1   1   1   1   1   2   2   2   1   1   2   2   1   2   1   2   1   2   1   2
##   5 125  88  70  83  18  99 106 149   4   6 123 130  16 127  90 135  40 113  74
##   2   1   1   2   2   2   2   1   1   2   2   1   1   2   1   2   1   2   1   1
##  21  24  55  72  75 120  64 129  95  91  51  60  59  46  17  15  11 150  32 126
##   2   2   1   2   1   1   1   1   2   2   1   2   1   2   2   2   2   1   2   1
##  45  33  53  10 121 138  37 136  58 102
##   2   2   1   2   1   1   2   1   2   1
```

```
cluster.es = cutree(iris.es, k=2) # euclidean-single
cluster.ea = cutree(iris.ea, k=2) # euclidean-average

# interpret the clusters
table(label.true = species.name[misc], label.cluster = cluster.es)
```

```
##             label.cluster
## label.true    1  2
##   setosa      0 50
##   versicolor 50  0
##   virginica  50  0
```

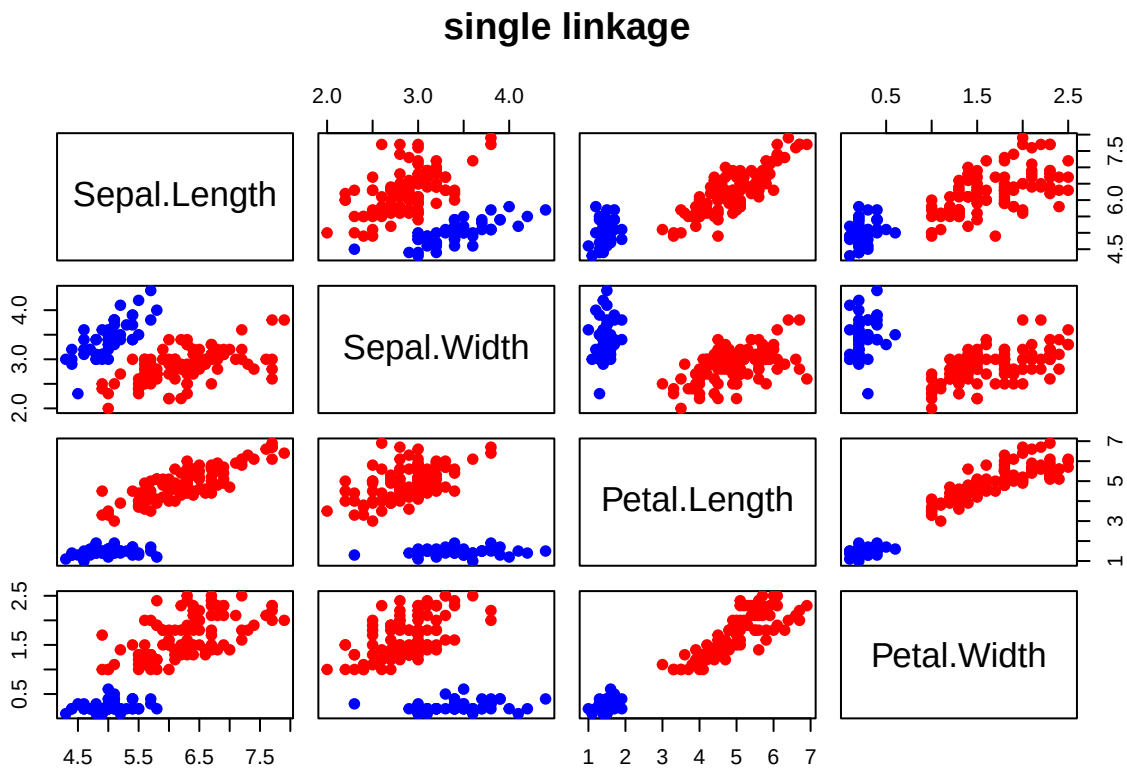
```r
table(label.true = species.name[misc], label.cluster = cluster.ec)
```

```
##             label.cluster
## label.true    1  2
##   setosa      0 50
##   versicolor 23 27
##   virginica  49  1
```

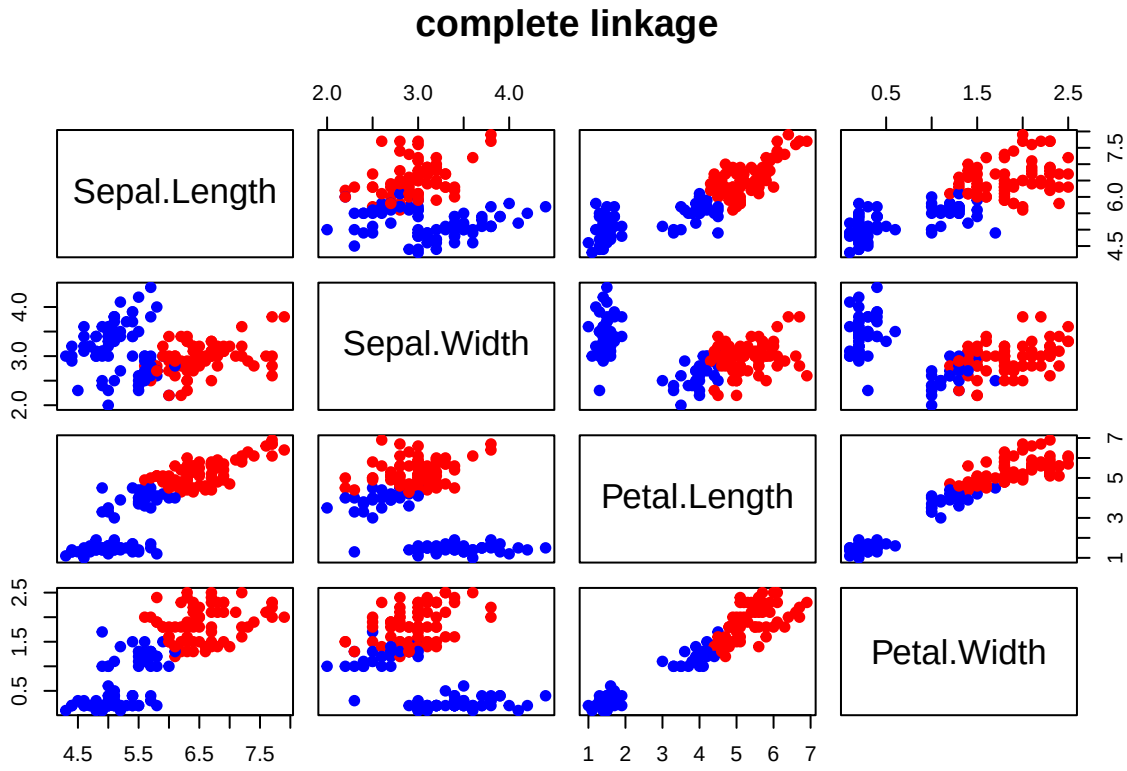
```r
table(label.true = species.name[misc], label.cluster = cluster.ea)
```

```
##             label.cluster
## label.true    1  2
##   setosa      0 50
##   versicolor 50  0
##   virginica  50  0
```

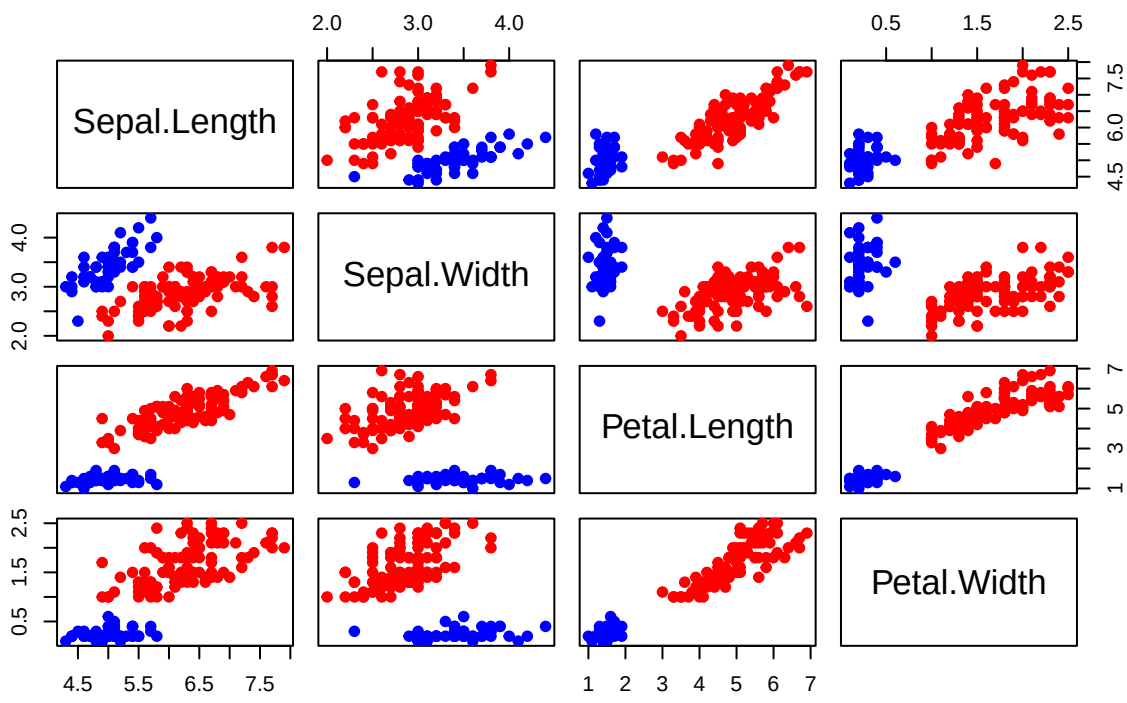
```r
plot(iris4, col=ifelse(cluster.es==1,'red','blue'), pch=19, main="single linkage")
```




```r
plot(iris4, col=ifelse(cluster.ec==1,'red','blue'), pch=19, main="complete linkage")
```

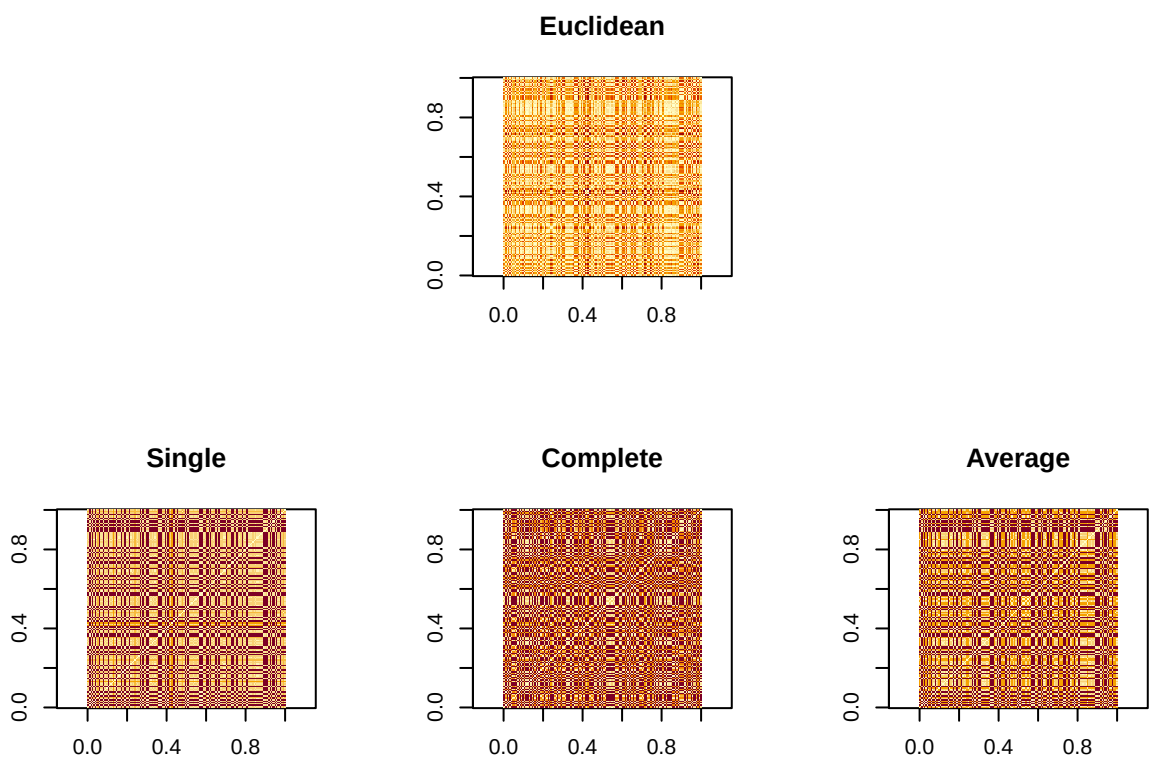



```
#! Complete does not seem really precise, since cant recognize "elongated clusters", it
#! tends to merge them as he looks to the larges distance

plot(iris4, col=ifelse(cluster.ea==1,'red','blue'), pch=19, main="average linkage")
```



8.1.5 Cophenetic matrix

```
# Let's give a mark to the algorithms: did they aggregate coherently with
# the dissimilarity matrix or not?

# compute the cophenetic matrices
#! It answers to: at which distance two obs are merged into the same cluster?
#! So we want them to be the closest possible to the original ones
coph.es = cophenetic(iris.es)
coph.ec = cophenetic(iris.ec)
coph.ea = cophenetic(iris.ea)

# compare with dissimilarity matrix (Euclidean distance)

layout(rbind(c(0,1,0),c(2,3,4)))
image(as.matrix(iris.e), main='Euclidean', asp=1 )
image(as.matrix(coph.es), main='Single', asp=1 )
image(as.matrix(coph.ec), main='Complete', asp=1 )
image(as.matrix(coph.ea), main='Average', asp=1 )
```




```
#! Not very useful
```

8.1.6 Copehenetic coefficients

```
# compute cophenetic coefficients
#! The higher the better, in fact for complete linkage is not really good
es = cor(iris.e, coph.es)
ec = cor(iris.e, coph.ec)
ea = cor(iris.e, coph.ea)

c("Eucl-Single"=es,"Eucl-Compl."=ec,"Eucl-Ave."=ea)
```

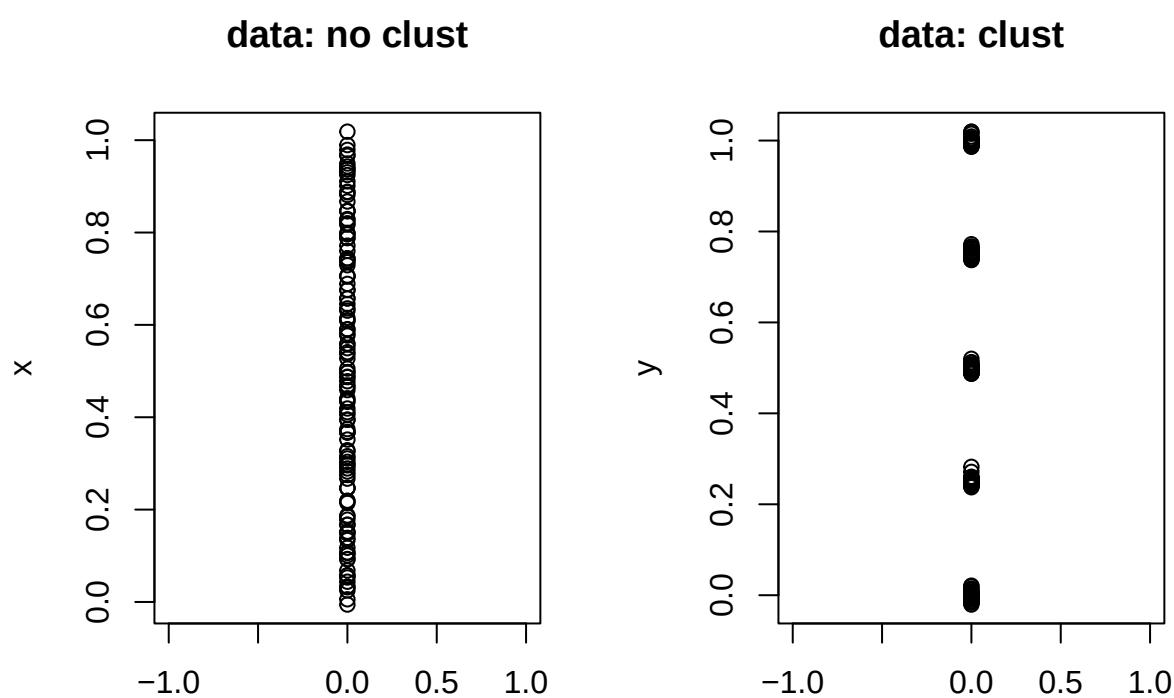
```
## Eucl-Single Eucl-Compl.   Eucl-Ave.
##   0.8638787   0.7276283   0.8769561
```

```
#! So CPCC are a way to choose at the end which linkage to use

# Exercise: repeat the analysis with other metrics
```

8.2 On simulated data

8.2.1 Univariate case

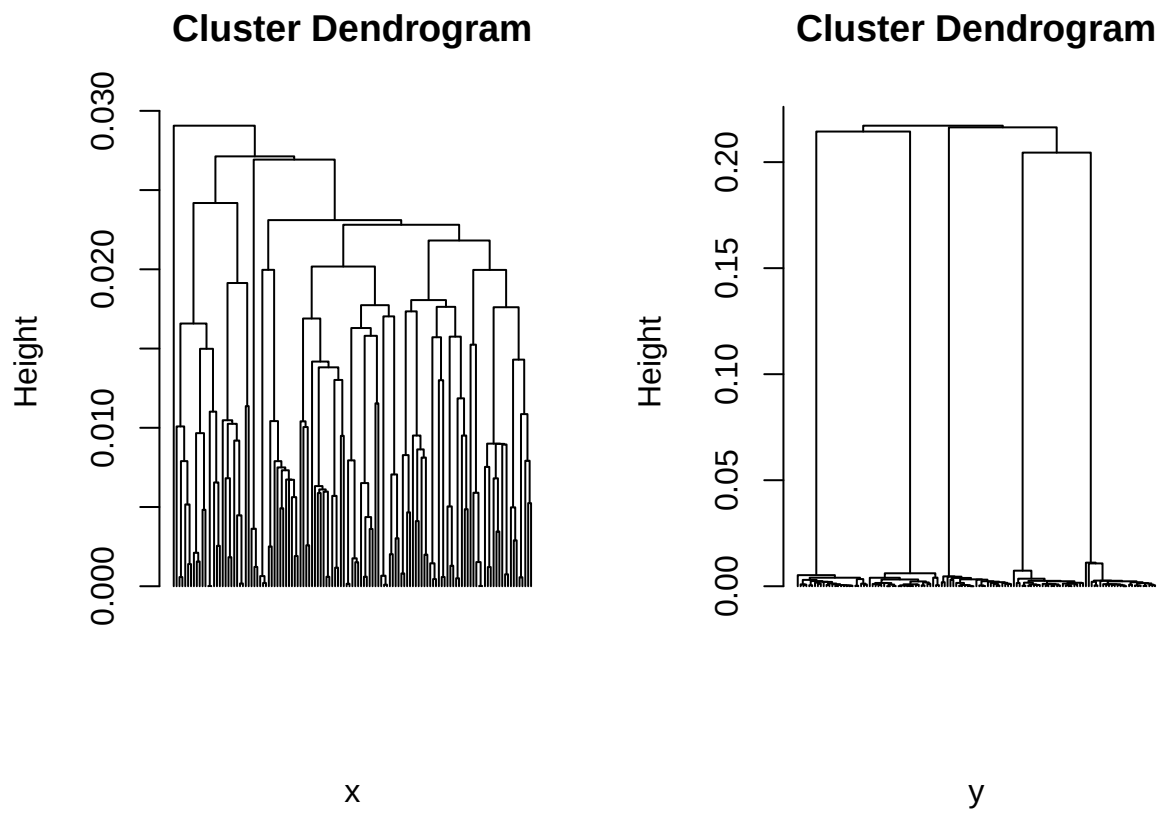
```
#_____________________________________________________________________________
##### Example 2: simulated data
#####-------------------------------------------------------------

### Univariate case
###-----------------
set.seed(123)
# x : vector of NON clustered data
x = 0:124/124 + rnorm(125, sd=0.01)
# y : vector of clustered data (5 clusters)
y = c(rnorm(25, mean=0, sd=0.01), rnorm(25, mean=0.25, sd=0.01),
      rnorm(25, mean=0.5, sd=0.01), rnorm(25, mean=0.75, sd=0.01),
      rnorm(25, mean=1, sd=0.01))

x = sample(x)
y = sample(y)


par(mfrow=c(1,2))
plot(rep(0,125),x, main='data: no clust',xlab='')
plot(rep(0,125),y, main='data: clust',xlab='')
```



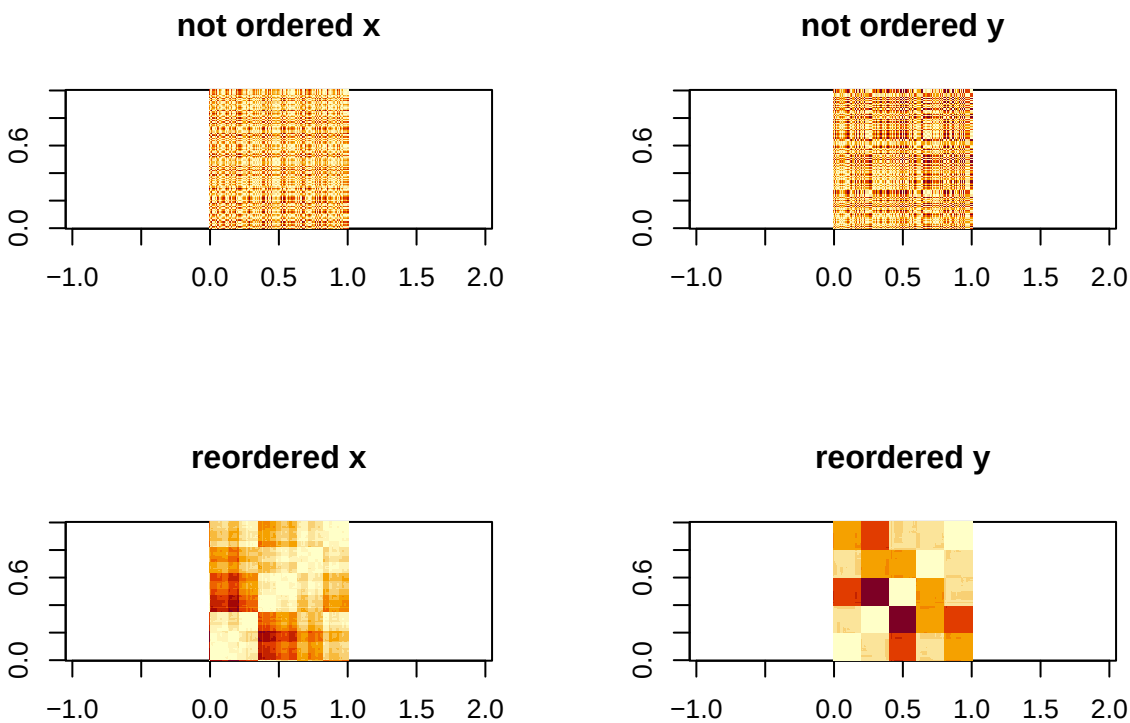

```
dx = dist(x)
dy = dist(y)
hcx= hclust(dx, method='single')
hcy= hclust(dy, method='single')
```

8.2.2 Balanced vs unbalanced trees

```
par(mfrow=c(1,2))
plot(hcx, labels=F, cex=0.5, hang=-0.1, xlab='', sub='x')
plot(hcy, labels=F, cex=0.5, hang=-0.1, xlab='', sub='y')
```



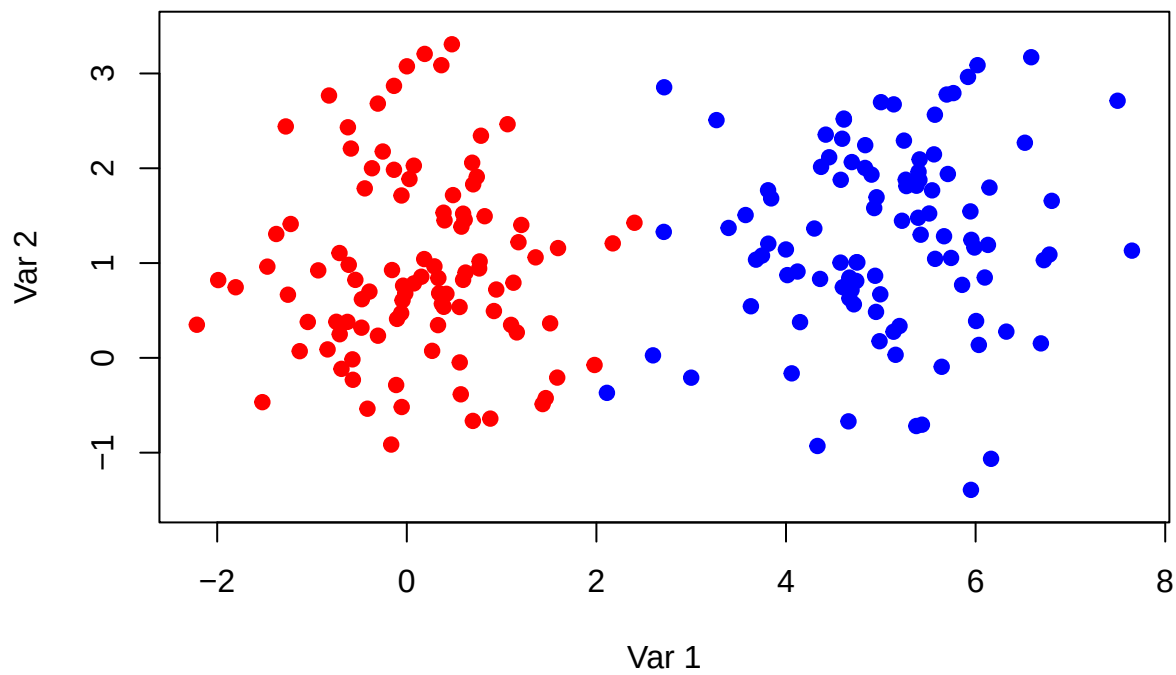

```
#! Unbalance trees (significant higher branches) show already a clustering structure,
#! ie all the found groups are very far away from each other => clear clustering structure
#! Balance trees => not clear clustering structure


par(mfrow=c(2,2))
image(as.matrix(dx), asp=1, main='not ordered x' )
image(as.matrix(dy), asp=1, main='not ordered y' )
image(as.matrix(dx)[hcx$order,hcx$order], asp=1, main='reordered x' )
image(as.matrix(dy)[hcy$order,hcy$order], asp=1, main='reordered y' )
```



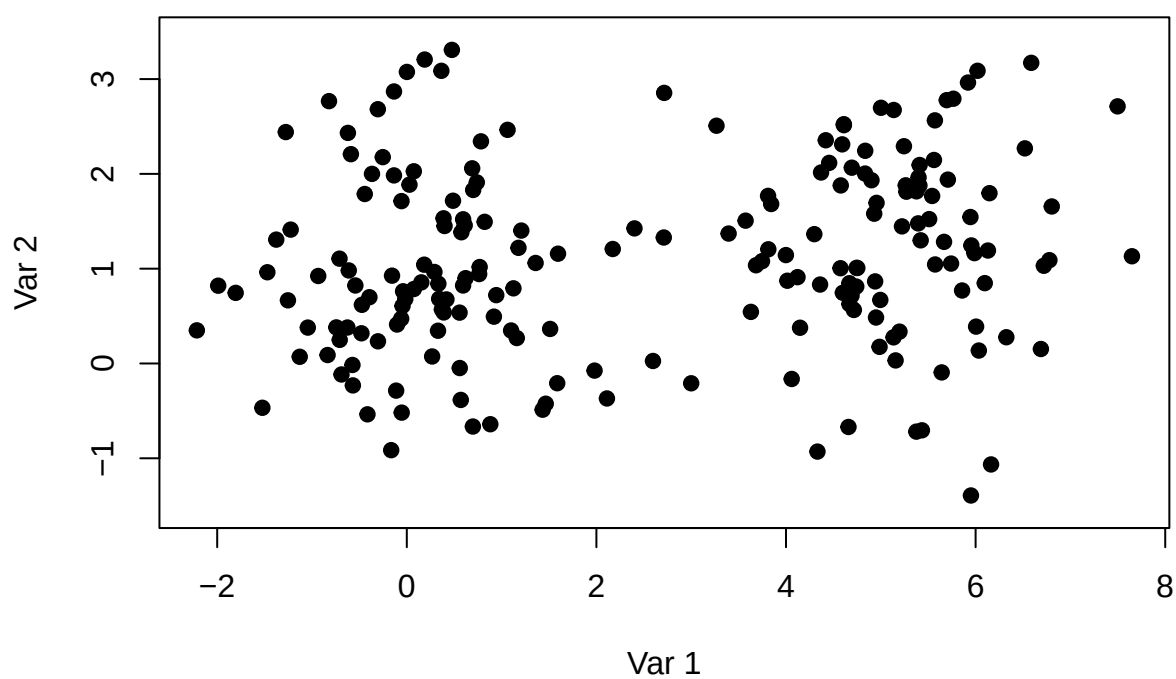

```
#! Units are independent, so we can reorder the obs tranquillamente. The information
#! carried is the same, so we can choose the order more interpretable for us
```

8.2.3 Chaining effect, bad for single linkage

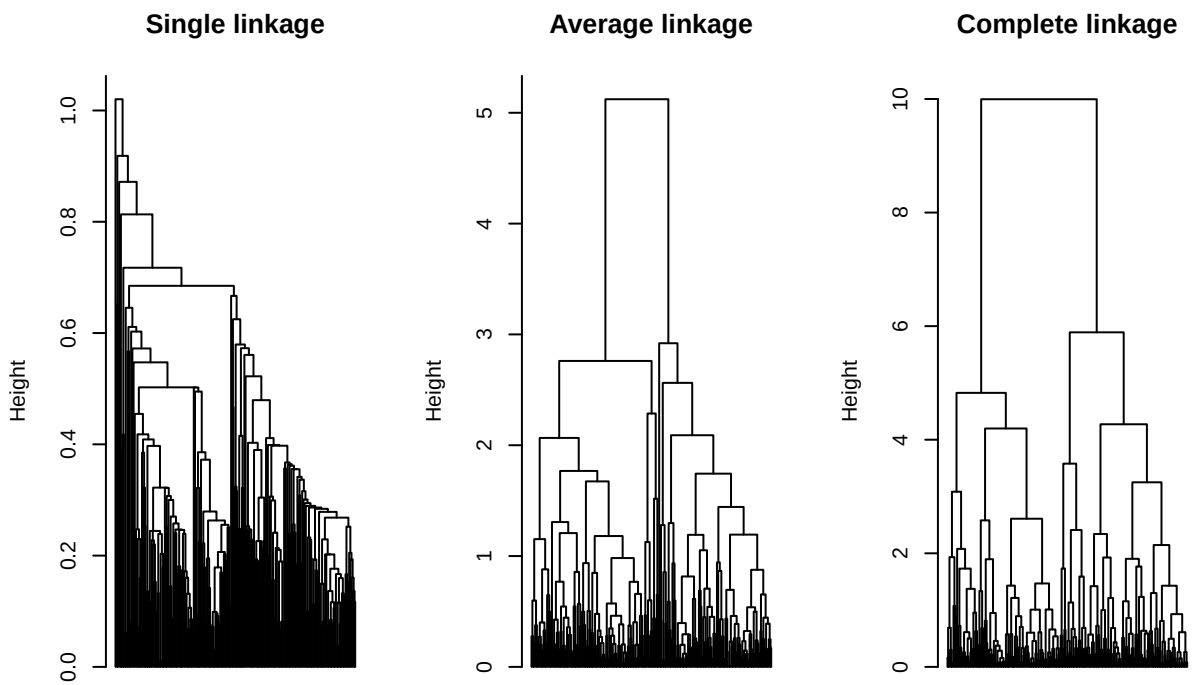
```
### Bivariate case (example of chaining effect)
###-------------------------------------------
p = 2
n = 100

#! Try to experiment separating more and more those means, until single linkage also works
mu1 = c(0,1)
mu2 = c(5,1.2)
sig = diag(rep(1,p))

set.seed(1)
X1 = rmvnorm(n, mu1, sig)
X2 = rmvnorm(n, mu2, sig)

X = rbind(X1, X2)
# If we knew the labels:

plot(X, xlab='Var 1', ylab='Var 2', col=rep(c('red','blue'),each=100), asp=1, pch=19)
```



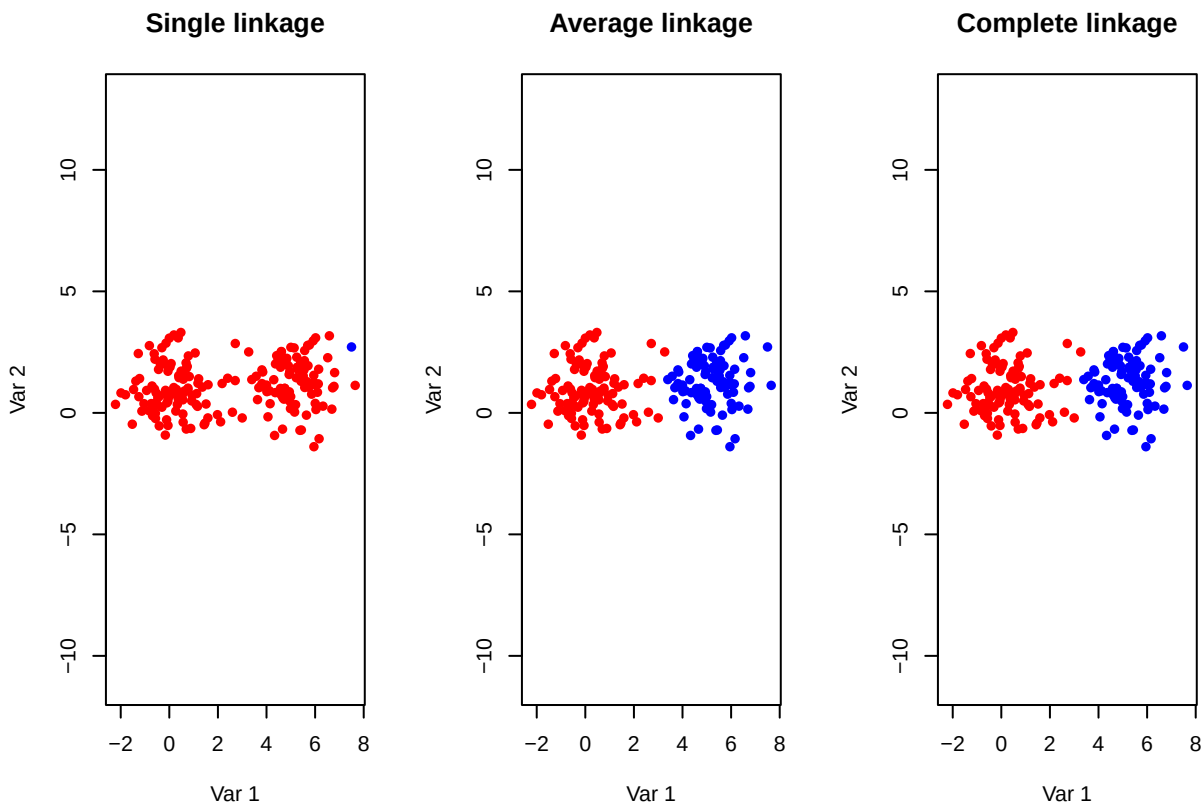

```
# How we actually see the data:
plot(X, xlab='Var 1', ylab='Var 2', asp=1, pch=19)
```



```r
# let's compare the clustering results with Euclidean distance and three
# types of linkage
x.d = dist(X, method = 'euclidean')

x.es = hclust(x.d, method='single')
x.ea = hclust(x.d, method='average')
x.ec = hclust(x.d, method='complete')


par(mfrow=c(1,3))
plot(x.es, main = 'Single linkage'  , hang=-0.1, xlab='', labels=F, sub='')
#! From that scale structure we suspect a chain effect, a merge into larger cluster
plot(x.ea, main = 'Average linkage' , hang=-0.1, xlab='', labels=F, sub='')
plot(x.ec, main = 'Complete linkage', hang=-0.1, xlab='', labels=F, sub='')
```

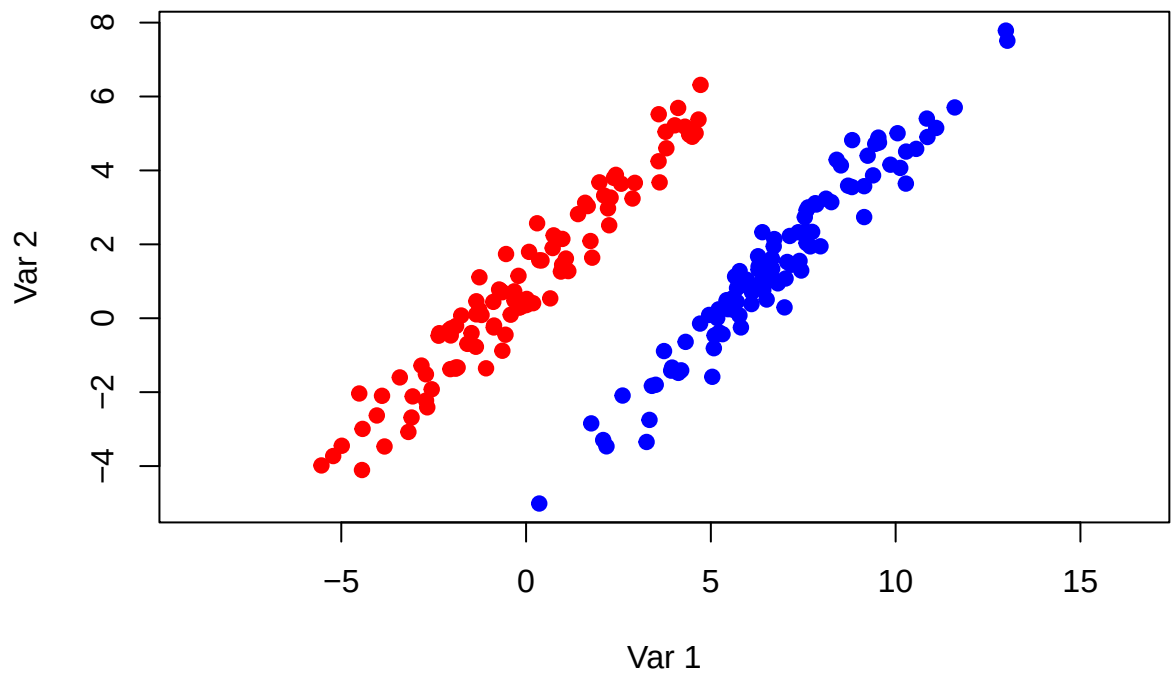



```r
# let's cut the tree as to get 2 clusters
cluster.es = cutree(x.es, k = 2)
cluster.ea = cutree(x.ea, k = 2)
cluster.ec = cutree(x.ec, k = 2)


par(mfrow=c(1,3))
plot(X, xlab='Var 1', ylab='Var 2', main = 'Single linkage',
    col=ifelse(cluster.es==1,'red','blue'), pch=16, asp=1)
plot(X, xlab='Var 1', ylab='Var 2', main = 'Average linkage',
    col=ifelse(cluster.ea==1,'red','blue'), pch=16, asp=1)
plot(X, xlab='Var 1', ylab='Var 2', main = 'Complete linkage',
    col=ifelse(cluster.ec==1,'red','blue'), pch=16, asp=1)
```

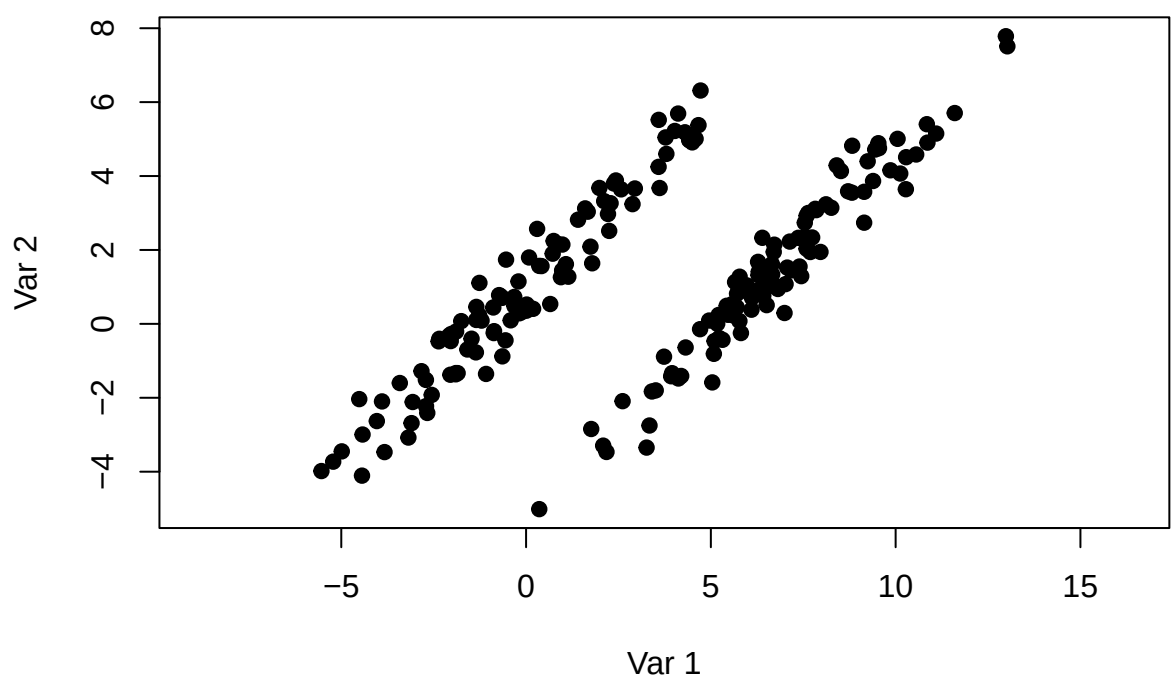


8.2.4 Ellipsoidal clusters, bad for complete linkage

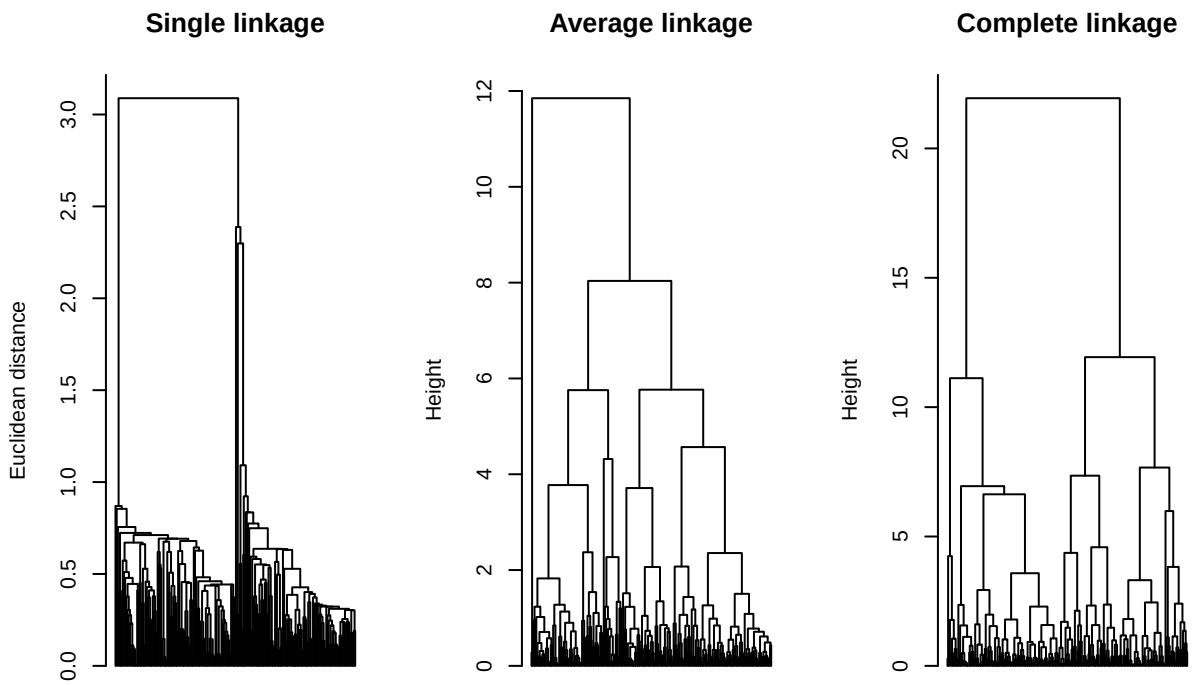
```
### Bivariate case (example of ellipsoidal clusters)
###-------------------------------------------------

p = 2
n = 100

mu1 = c(0,1)
mu2 = c(6.5,1)

e1 = c(1,1)
e2 = c(-1,1)
sig = 5*cbind(e1)%*%rbind(e1)+.1*cbind(e2)%*%rbind(e2)

set.seed(2)
X1 = rmvnorm(n, mu1, sig)
X2 = rmvnorm(n, mu2, sig)
X = rbind(X1, X2)

# If we knew the labels:

plot(X, xlab='Var 1', ylab='Var 2', asp=1, col=rep(c('red','blue'),each=100), pch=19)
```



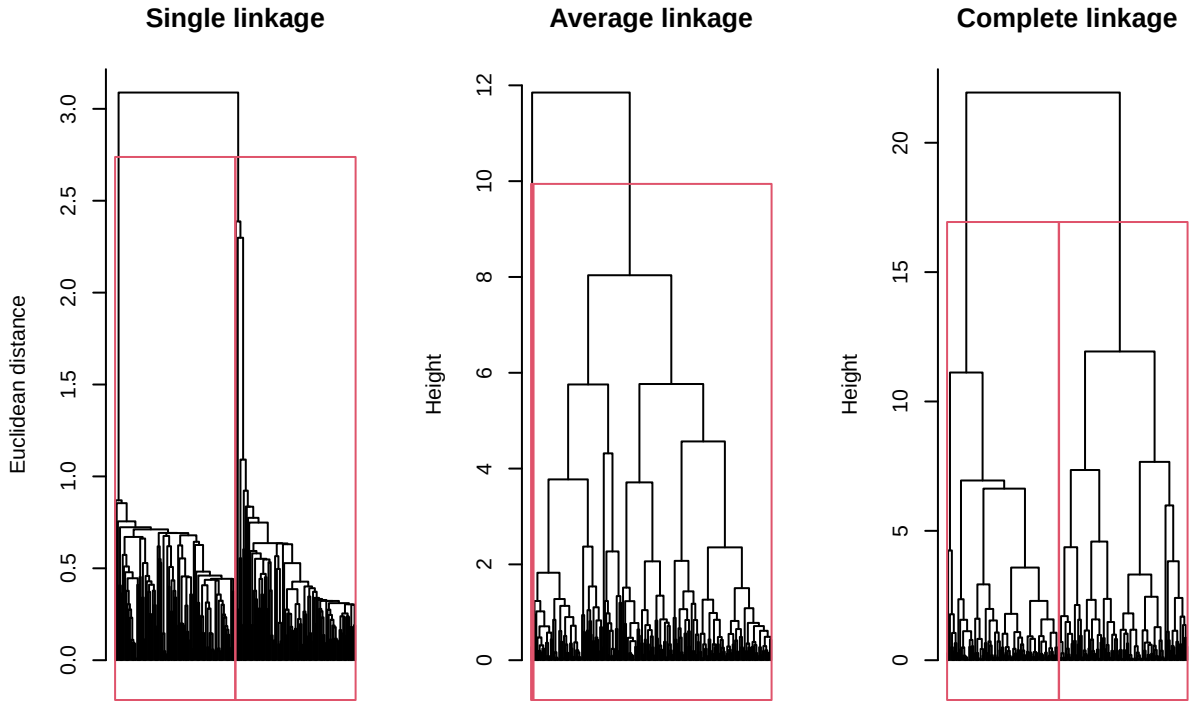

```r
# How we actually see the data:
plot(X, xlab='Var 1', ylab='Var 2', asp=1, pch=19)
```



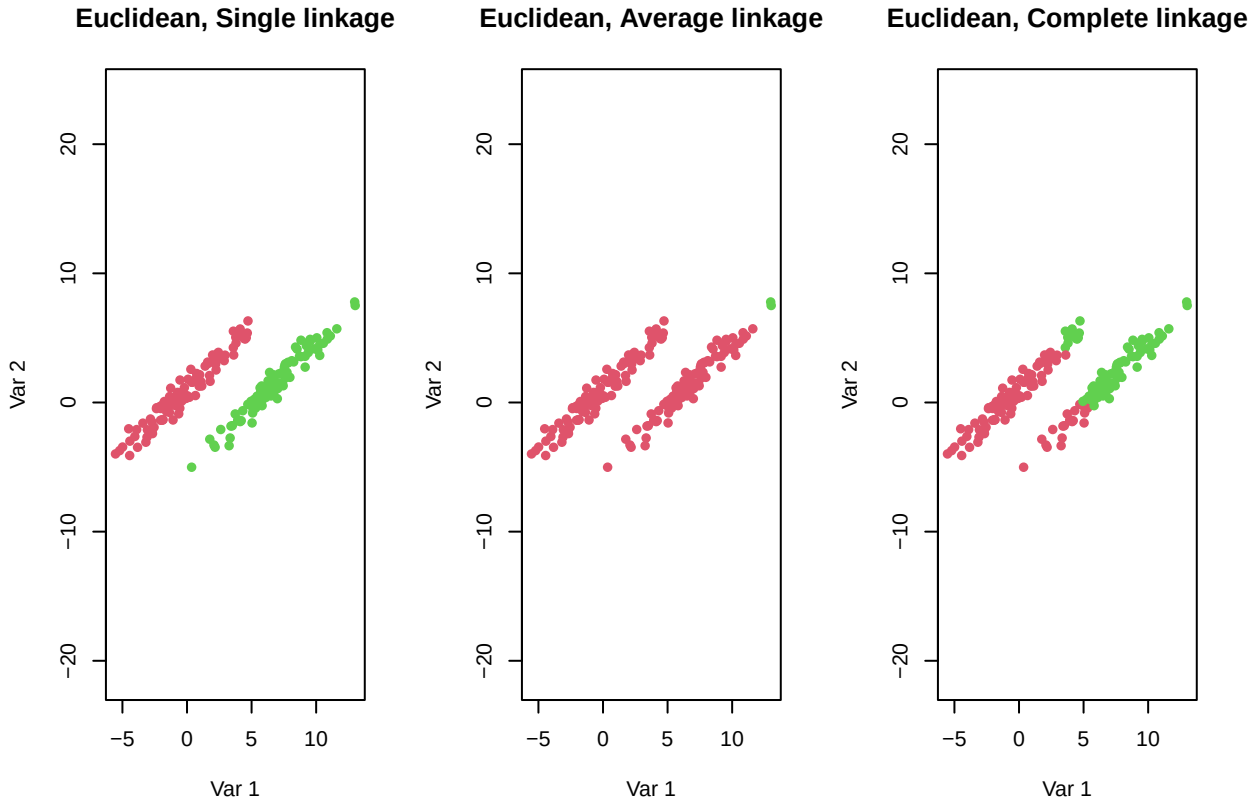
```r
## compare results with different linkage, when Euclidean distances is used
x.d = dist(X, method = 'euclidean')

x.es = hclust(x.d, method='single')
x.ea = hclust(x.d, method='average')
x.ec = hclust(x.d, method='complete')

par(mfrow=c(1,3))
plot(x.es, main = 'Single linkage', ylab='Euclidean distance', hang=-0.1, xlab='', labels=F, sub='')
plot(x.ea, main = 'Average linkage', hang=-0.1, xlab='', labels=F, sub='')
plot(x.ec, main = 'Complete linkage', hang=-0.1, xlab='', labels=F, sub='')
```




```r
#! Seems that the complete one is the best, or also the single
#! And removing those two obs in the middle will produce a even better cluster structure,
#! a final merge really far away, very unbalanced trees
#! Look at the y-scale also, to really see the difference in the cuts

# let's cut the tree to get 2 clusters

par(mfrow=c(1,3))
plot(x.es, main = 'Single linkage', ylab='Euclidean distance', hang=-0.1,
     xlab='', labels=F, sub='')
rect.hclust(x.es, k=2)
plot(x.ea, main = 'Average linkage', hang=-0.1, xlab='', labels=F, sub='')
rect.hclust(x.ea, k=2)
plot(x.ec, main = 'Complete linkage', hang=-0.1, xlab='', labels=F, sub='')
rect.hclust(x.ec, k=2)
```




```
cluster.es = cutree(x.es, k = 2)
cluster.ea = cutree(x.ea, k = 2)
cluster.ec = cutree(x.ec, k = 2)

par(mfrow=c(1,3))
plot(X, xlab='Var 1', ylab='Var 2', main = 'Euclidean, Single linkage', col=cluster.es+1,
     pch=16, asp=1)
plot(X, xlab='Var 1', ylab='Var 2', main = 'Euclidean, Average linkage', col=cluster.ea+1,
     pch=16, asp=1)
plot(X, xlab='Var 1', ylab='Var 2', main = 'Euclidean, Complete linkage', col=cluster.ec+1,
     pch=16, asp=1)
```




```
#! The complete one seemed fine but instead didnt capure the ellipsoidal shape

# Change the mean mu2
# Example:
# mu2 = c(4.5,1.2)
```

8.3 On earthquake dataset

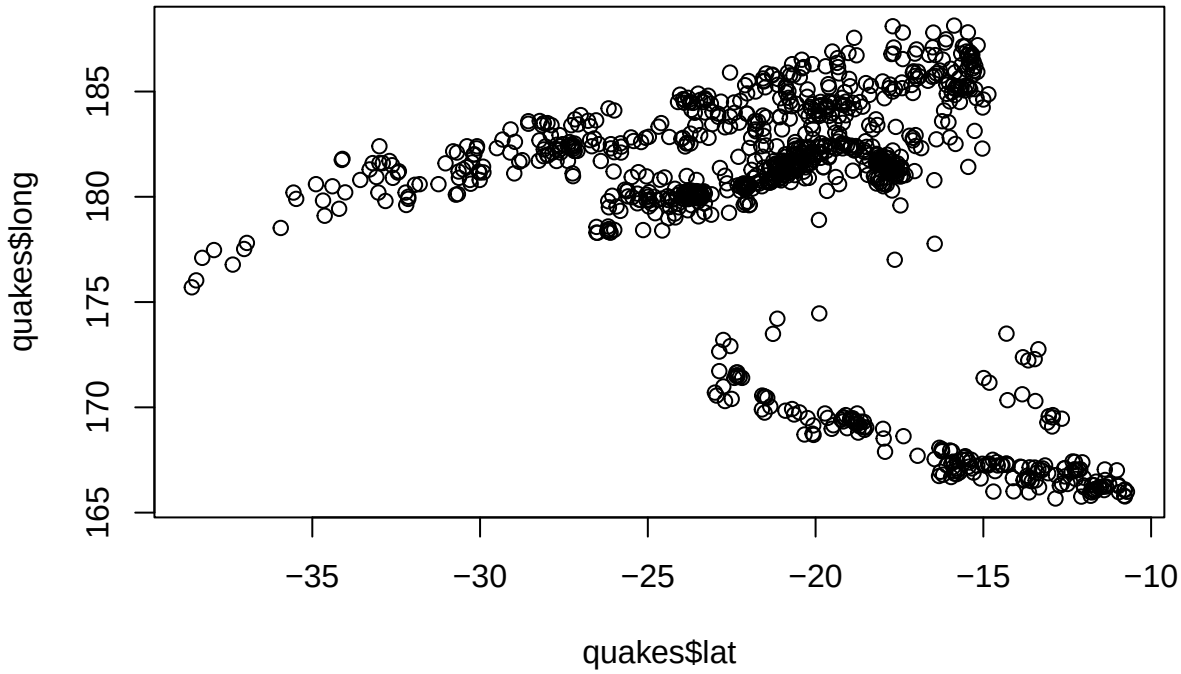
```
#-------------------------------------------------------------------------------
##### Example 3: earthquakes dataset
#####-------------------------------

# help(quakes)

head(quakes)
```

```
##      lat   long depth mag stations
## 1 -20.42 181.62   562 4.8       41
## 2 -20.62 181.03   650 4.2       15
## 3 -26.00 184.10    42 5.4       43
## 4 -17.97 181.66   626 4.1       19
## 5 -20.42 181.96   649 4.0       11
## 6 -19.68 184.31   195 4.0       12
```

```
dim(quakes)
```

```
## [1] 1000    5
```

```
plot(quakes$lat, quakes$long)
```



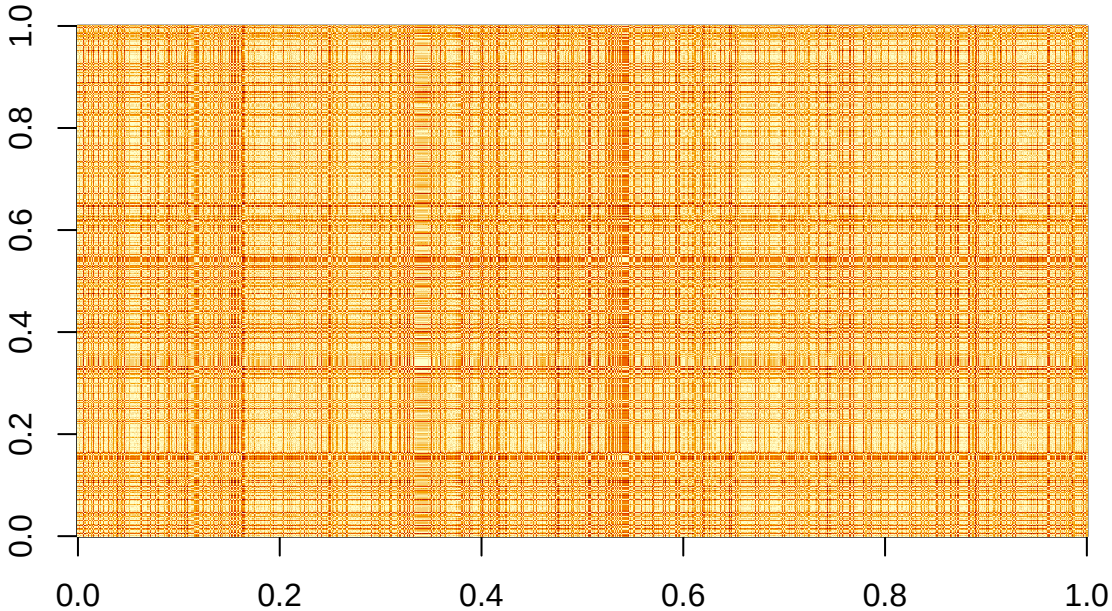

```
Q = cbind(quakes[,1:2], depth = -quakes[,3]/100)
head(Q)
```

```
##      lat   long depth
## 1 -20.42 181.62 -5.62
## 2 -20.62 181.03 -6.50
## 3 -26.00 184.10 -0.42
## 4 -17.97 181.66 -6.26
## 5 -20.42 181.96 -6.49
## 6 -19.68 184.31 -1.95
```

```
plot3d(Q, size=4, col='orange', aspect = T)
#! Seemingly identifiable clusters:
plot3d(Q, size=4, col=ifelse(Q$long<175,'orange',"red"), aspect = T, sub="a eye-sight division")
```

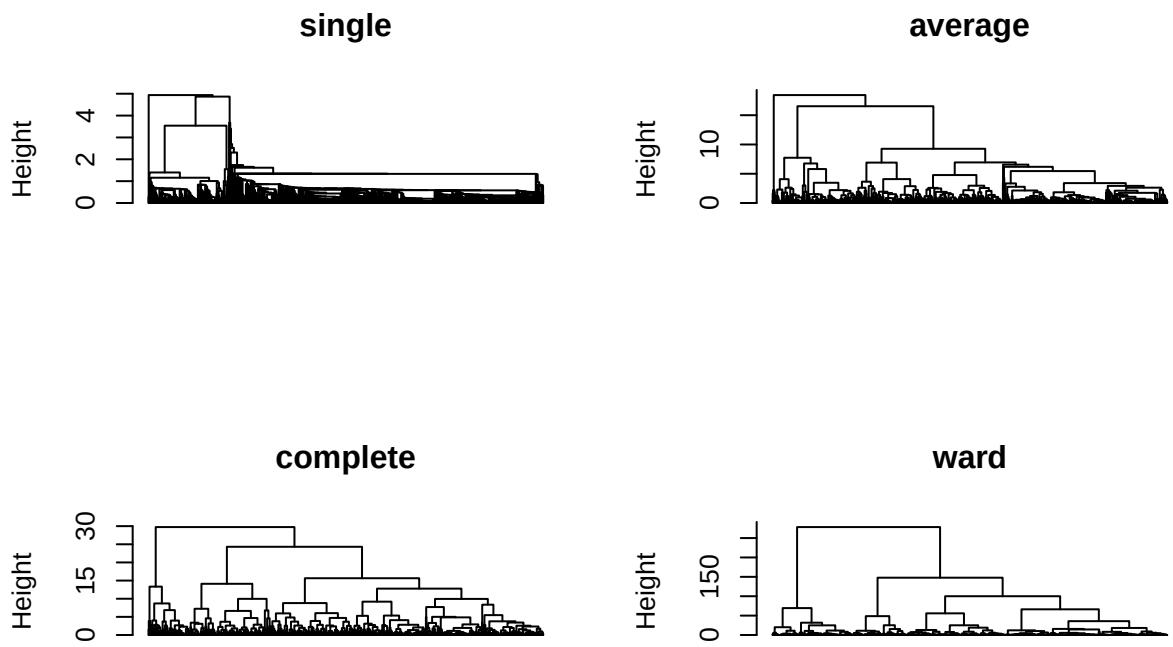
8.3.1 Dissimiliraty matrix

```
# dissimilarity matrix (Euclidean metric)
d = dist(Q)
image(as.matrix(d))
```



8.3.2 Dendrogram plots

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

clusts = hclust(d, method='single')
plot(clusts, hang=-0.1, labels=FALSE, main='single', xlab='', sub='')
# rect.hclust(clusts, k=2)
# rect.hclust(clusts, k=3)

clusta = hclust(d, method='average')
plot(clusta, hang=-0.1, labels=FALSE, main='average', xlab='', sub='')
# rect.hclust(clusta, k=2)
# rect.hclust(clusta, k=3)

clustc = hclust(d, method='complete')
plot(clustc, hang=-0.1, labels=FALSE, main='complete', xlab='', sub='')
# rect.hclust(clustc, k=2)
# rect.hclust(clustc, k=3)

clustw = hclust(d, method='ward.D2')
plot(clustw, hang=-0.1, labels=FALSE, main='ward', xlab='', sub='')
```




```
# rect.hclust(clustw, k=2)
# rect.hclust(clustw, k=3)
```

8.3.3 Ward linkage

```
## Ward-Linkage (see J-W p. 692-693):
# Ward considered hierarchical clustering procedures based on minimizing the
# 'loss of information' from joining two groups. This method is usually implemented
# with loss of information taken to be an increase in an error sum of squares
# criterion, ESS. First for a given cluster k, let ESS[k] be the sum of the
# squared deviations of every item in the cluster from the cluster mean (centroid).
# (ESS[k]=sum_{x.j in cluster k} t(x.j-x.mean[k])%*%(x.j-x.mean[k]), where
# x.mean[k] is the centroid of cluster k).
# If there are currently K clusterts, define ESS as the sum of the ESS[k]
# (ESS=ESS[1]+ESS[2]+...+ESS[K]). At each step in the analysis, the union of
# every possible pair of clusters is considered, and the two clusters whose
# combination results in the smallest increase in ESS (minimum loss of information)
# are joined.
# Initially, each cluster consists of a single item and, if there are N items,
# ESS[k]=0, k=1,2,...,N, so ESS=0. At the other extreme, when all the clusters are
# combined in a single group of N items, the value of ESS is given by
# ESS=sum_j(t(x.j-x.mean)%*%(x.j-x.mean)), where x.j is the multivariate measurement
# associated with the jth item and x.mean is the mean of all items.

open3d()
```

```
## wgl
##   4
```

```
# single linkage
clusters = cutree(clusts, 2)
plot3d(Q, size=4, col=clusters+1, aspect = T, sub="single linkage, cut at 2")

open3d()
```

```
## wgl
##   5
```

```r
clusters = cutree(clusts, 3)
plot3d(Q, size=4, col=clusters+1, aspect = T, sub="single linkage, cut at 3")

open3d()
```

```
## wgl
##   6
```

```r
# average linkage
clustera = cutree(clusta, 2)
plot3d(Q, size=4, col=clustera+1, aspect = T, sub="average linkage, cut at 2")

open3d()
```

```
## wgl
##   7
```

```r
clustera = cutree(clusta, 3)
plot3d(Q, size=4, col=clustera+1, aspect = T, sub="average linkage, cut at 3")

open3d()
```

```
## wgl
##   8
```

```r
# complete linkage
clusterc = cutree(clustc, 2)
plot3d(Q, size=4, col=clusterc+1, aspect = T, sub="complete linkage, cut at 2")

open3d()
```

```
## wgl
##   9
```

```r
clusterc = cutree(clustc, 3)
plot3d(Q, size=4, col=clusterc+1, aspect = T, sub="complete linkage, cut at 3")

open3d()
```

```
## wgl
##  10
```

```r
# ward linkage
clusterw = cutree(clustw, 2)
plot3d(Q, size=4, col=clusterw+1, aspect = T, sub="ward linkage, cut at 2")

open3d()
```

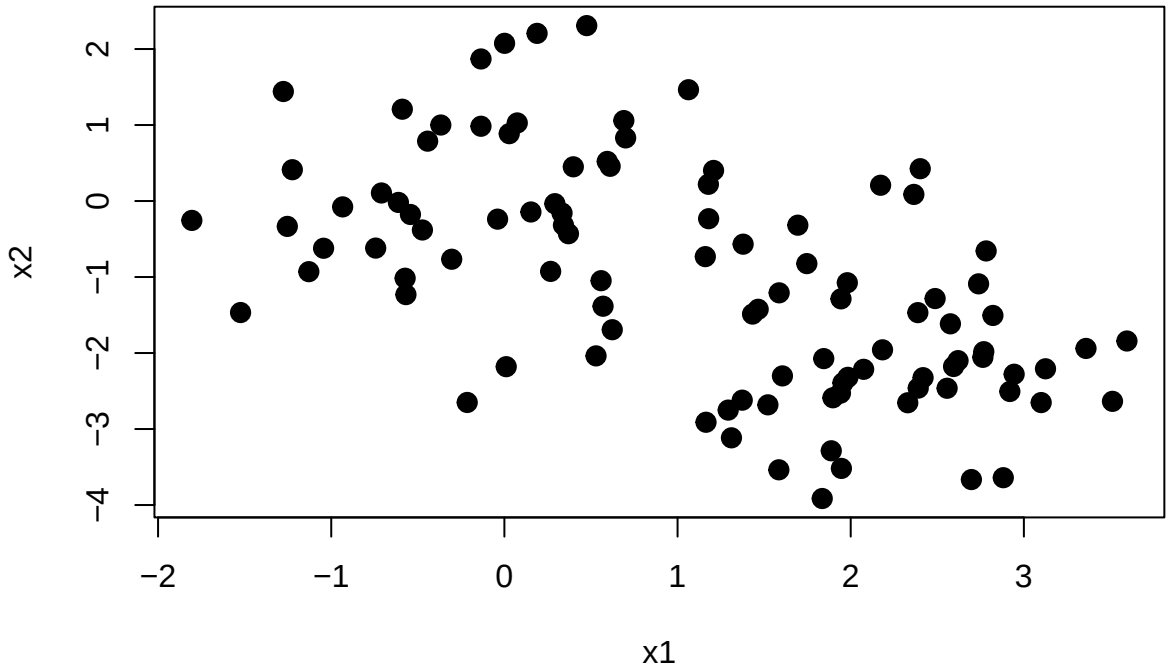
```
## wgl
##  11
```

```r
clusterw = cutree(clustw, 3)
plot3d(Q, size=4, col=clusterw+1, aspect = T, sub="ward linkage, cut at 3")
```

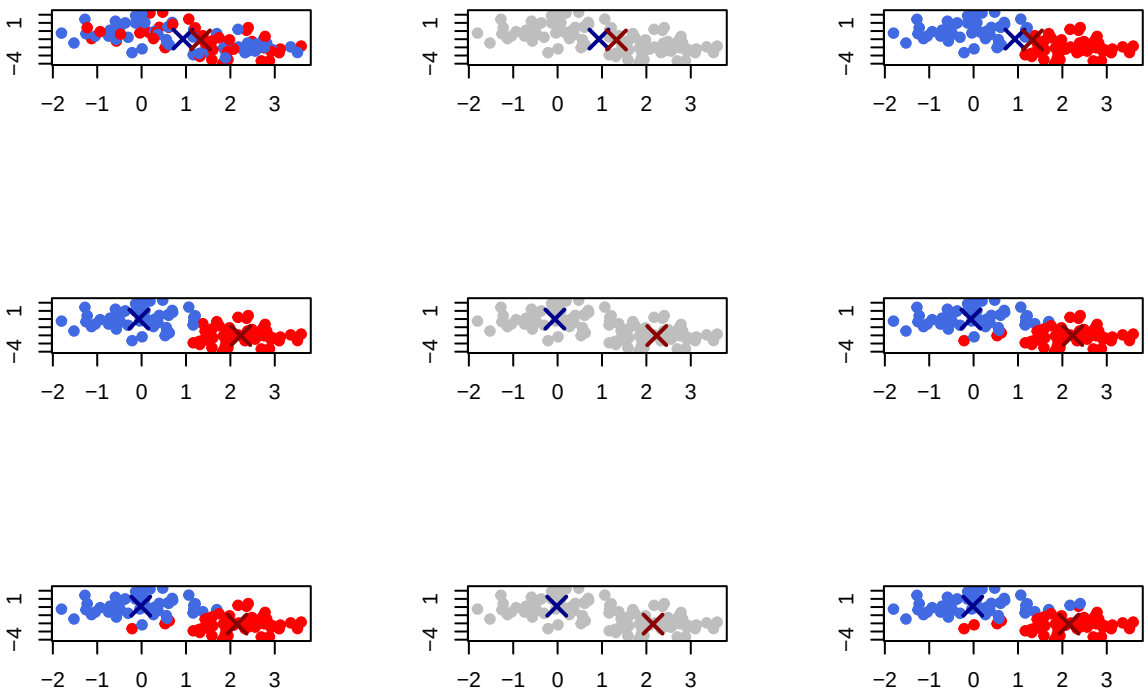
8.4 K-means method

8.4.1 Algorithm simulation

```r
#_____________________________________________________________________________
##### K-means method

#################################################################
# Recall K-means algorithm:
# simulated data
n = 100
set.seed(1)
x = matrix(rnorm(n*2), ncol=2)
x[1:(n/2),1] = x[1:(n/2),1]+2
x[1:(n/2),2] = x[1:(n/2),2]-2

plot(x,pch=20,cex=2,xlab='x1',ylab='x2')
```




```
# k-means algorithm
k = 2
#! We do the initialization by assigning to each point a random label between 1:k
cluster = sample(1:2, n, replace=TRUE)
iter.max = 3

colplot = c('royalblue','red')
colpoints = c('blue4','red4')

# dev.new()
par(mfrow = c(3,3))
# layout(matrix(c(1,1,1,1,1,1,1,1,1),nrow=3))
for(i in 1:iter.max){
    C = NULL
    for(l in 1:k){
      C = rbind(C, colMeans(x[cluster == l,]))
    }

    plot(x, col = colplot[cluster],pch=19, xlab="", ylab="")
    # line = readline()

    points(C, col = colpoints, pch = 4, cex = 2, lwd = 2, xlab="", ylab="")
    # line = readline()

    plot(x, col = 'grey',pch=19, xlab="", ylab="")
    points(C, col = colpoints, pch = 4, cex = 2, lwd = 2, xlab="", ylab="")
    # line = readline()

    QC = rbind(C, x)
    Dist = as.matrix(dist(QC, method = 'euclidean'))[(k+1):(k+n),1:k]
    for(j in 1:n){
      cluster[j] = which.min(Dist[j,])
    }

    plot(x, col = colplot[cluster],pch=19, xlab="", ylab="")
    points(C, col = colpoints, pch = 4, cex = 2, lwd = 2)
    # line = readline()
}
```



8.4.2 K-means on earthquake dataset

```
# back to the earthquakes dataset

### in automatic, command kmeans()
# help(kmeans)

result.k = kmeans(Q, centers=2) # Centers: fixed number of clusters

names(result.k)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
result.k$cluster      # labels of clusters
```

```
##    [1] 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2
##   [38] 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 1
##   [75] 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1
##  [112] 1 1 1 1 1 2 2 2 2 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 2 1 2 1 1 2
##  [149] 1 1 1 2 1 2 2 1 2 1 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [186] 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
##  [223] 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 2 2 1 2 1 1 1 2 1
##  [260] 1 1 1 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
##  [297] 1 1 1 2 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 2 1 2 2 1 1 1 2 1 1 2 1 2 1 1 1
##  [334] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1
##  [371] 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1
##  [408] 2 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1
##  [445] 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1
##  [482] 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 2 1
##  [519] 1 1 1 1 1 2 1 1 2 2 1 1 2 2 1 2 1 2 1 2 2 1 2 2 2 2 2 2 2 1 1 1 1 2 2 1 1
##  [556] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1
##  [593] 2 2 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 2 1 1 1 2
##  [630] 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1
##  [667] 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1
##  [704] 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 1 2 1 1 1 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1
##  [741] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1
```

```
##  [778] 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1
##  [815] 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1
##  [852] 2 2 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1 2 2 1 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2
##  [889] 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 2 2 1 1 1 1 2 1 1 1 1
##  [926] 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
##  [963] 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 2 2 1 1 1 1 1 1 1 1 2 1 1
## [1000] 2
```

```
result.k$centers      # centers of the clusters
```

```
##         lat     long     depth
## 1 -21.87143 182.3506 -3.479182
## 2 -15.87785 168.2598 -1.696390
```

```
result.k$totss        # tot. sum of squares
```

```
## [1] 66706.31
```

```
result.k$withinss     # sum of squares within clusters, the w(k) of the theory
```

```
## [1] 24075.266  3899.542
```

```
result.k$tot.withinss # sum(sum of squares within cluster)
```

```
## [1] 27974.81
```

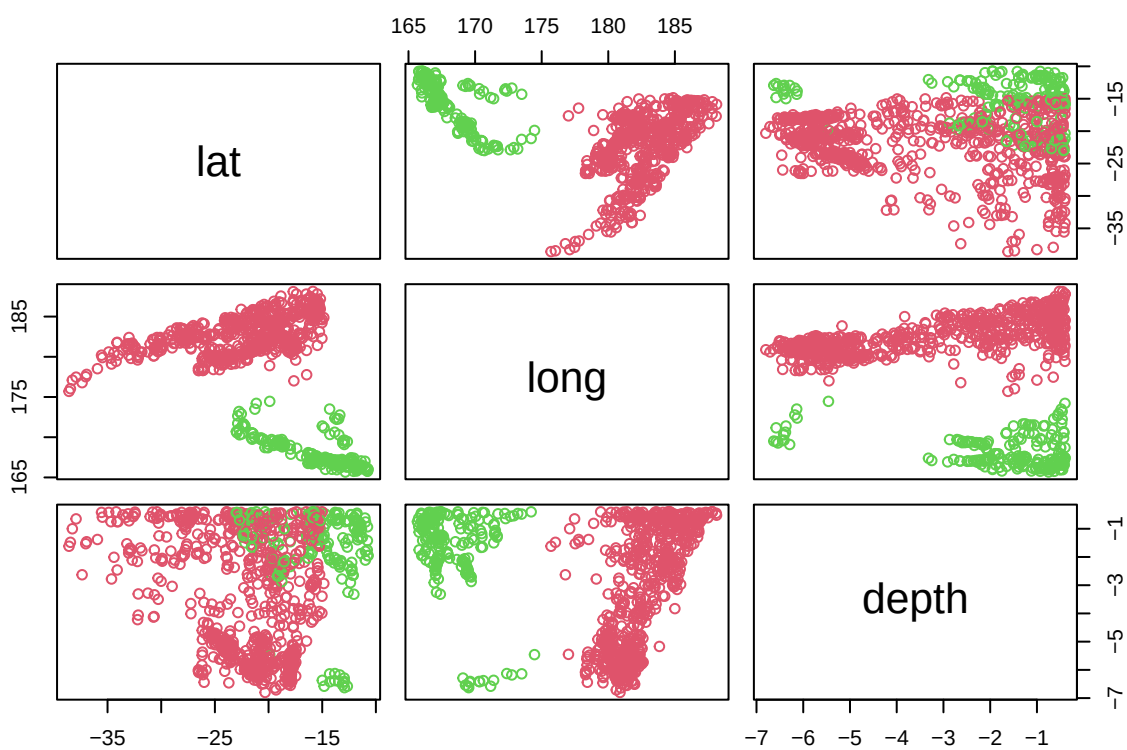
```
result.k$betweenss    # sum of squares between clusters
```

```
## [1] 38731.5
```

```
result.k$size         # dimension of the clusters
```

```
## [1] 795 205
```

```
plot(Q, col = result.k$cluster+1)
```



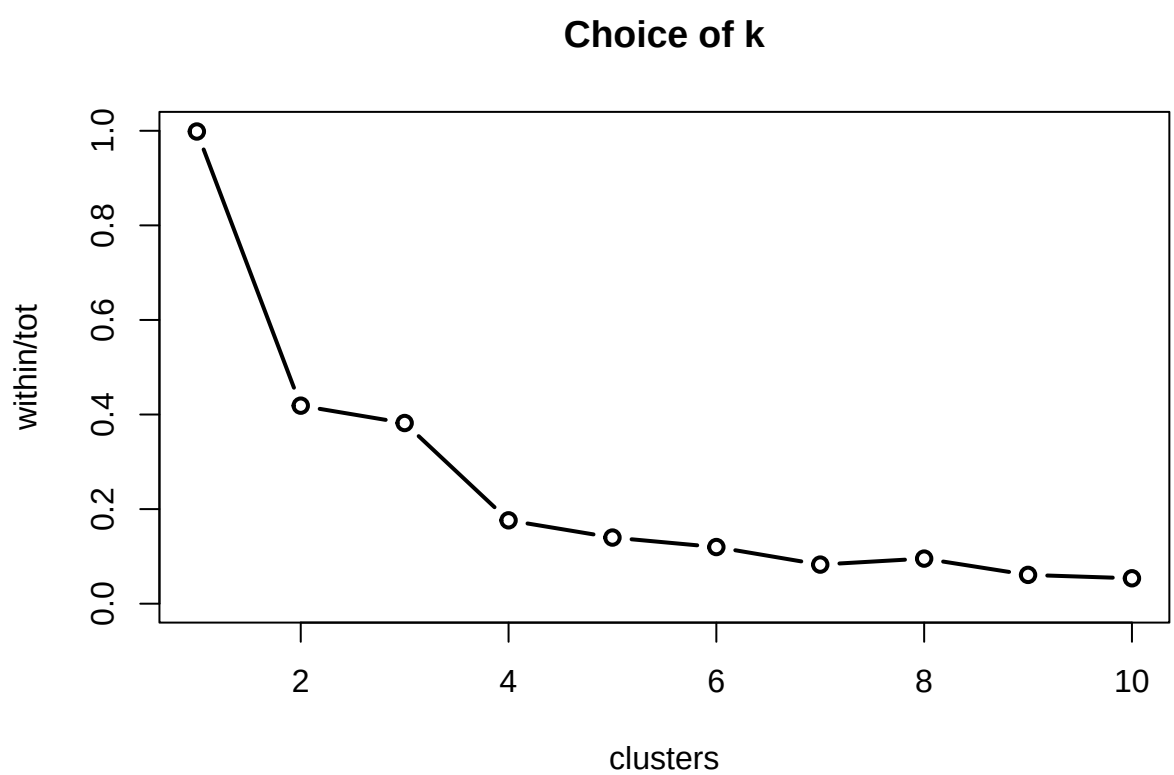
```
open3d()
```

```
## wgl
##  12
```

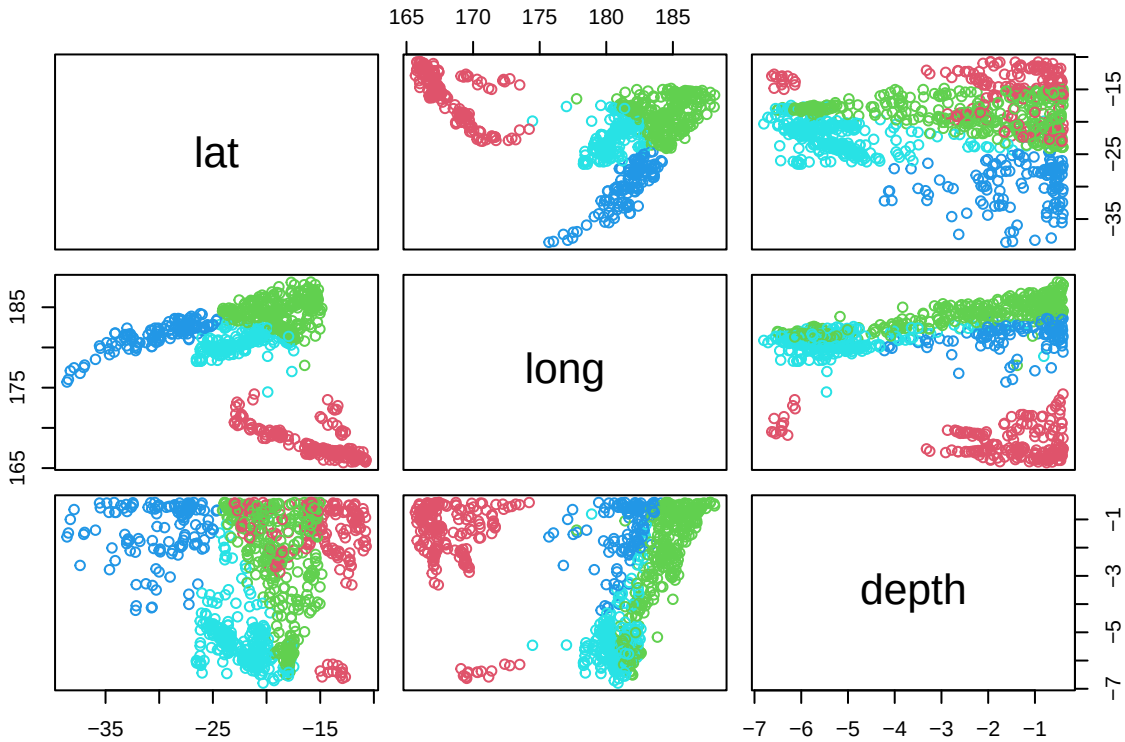
```
plot3d(Q, size=4, col=result.k$cluster+1, aspect = T)
points3d(result.k$centers,size=10)
```

8.4.3 How to choose k

```
### How to choose k:
### 1) evaluate the variability between the groups with respect to
###    the variability withing the groups
### 2) evaluate the result of hierarchical clustering (not recommended,
###    quite computationally expensive)

# (we've just seen method 2) and suggested k = 2)

# method 1)
b = NULL
w = NULL
for(k in 1:10){
  result.k = kmeans(Q, k)
  w = c(w, sum(result.k$wit))
  b = c(b, result.k$bet)
  if(k==3){ # to get the plot the prof got
    b=b+100;
  }
}

#! The elbow method
matplot(1:10, w/(w+b), pch='', xlab='clusters', ylab='within/tot', main='Choice of k', ylim=c(0,1))
lines(1:10, w/(w+b), type='b', lwd=2)
```



```r
# this method seems to suggest k = 2 or 4, definitely not 3
#! (what do we gain moving to 3? nothing the ratio doesnt increment too much)
# let's try also k=4:
result.k = kmeans(Q, 4)

plot(Q, col = result.k$cluster+1)
```




```r
open3d()
```

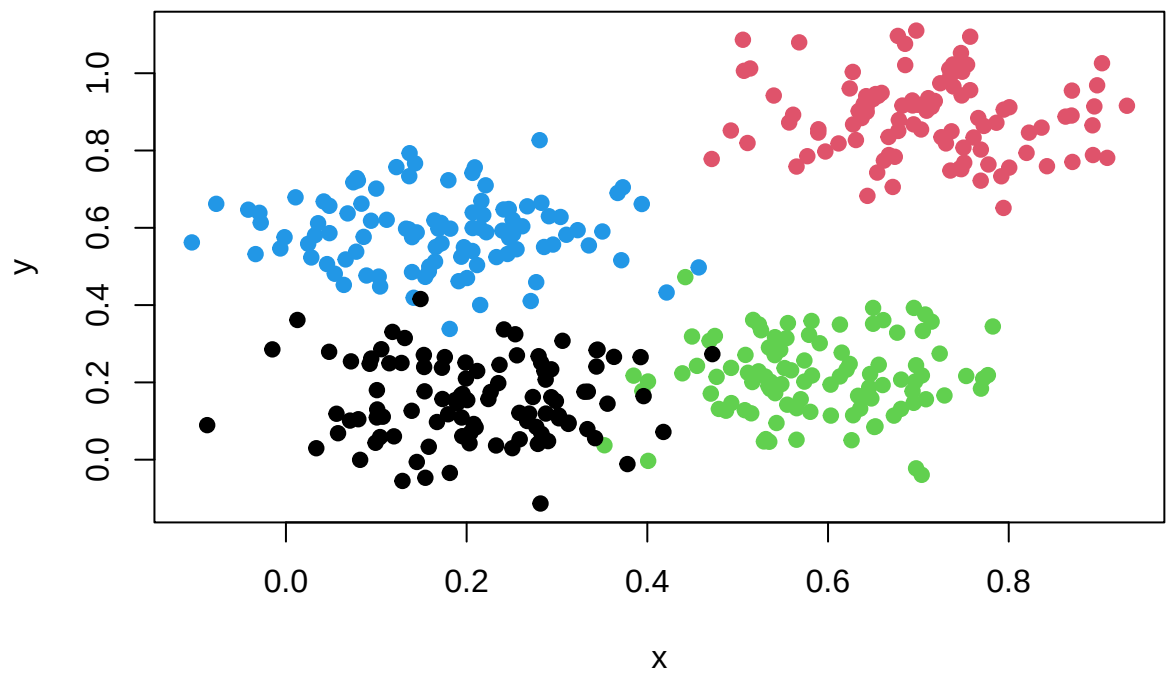
```
## wgl
##  13
```

```r
plot3d(Q, size=4, col=result.k$cluster+1, aspect = T)
points3d(result.k$centers, size = 10)
```

8.5 DBscan

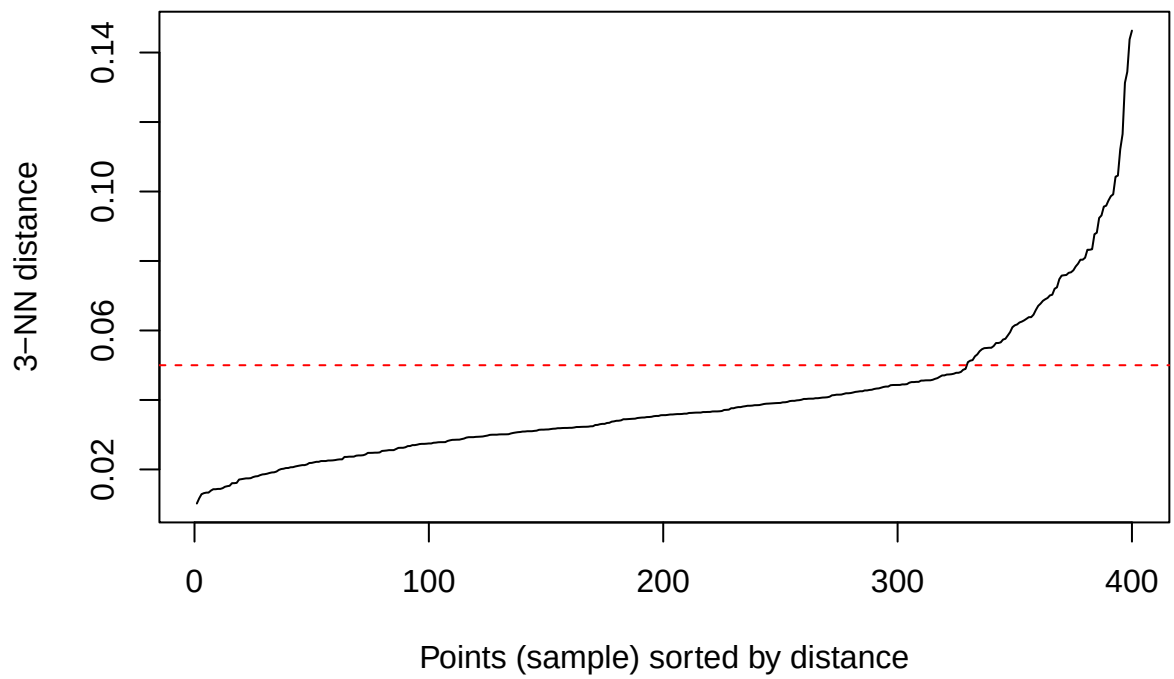
8.5.1 On simulated data

```r
#_____________________________________________________________________________
##### DBSCAN

# Simulate the data
set.seed(2)
n = 400
x = cbind(x = runif(4) + rnorm(n, sd = 0.1), y = runif(4) + rnorm(n, sd = 0.1))
true_clusters = rep(1:4, time = 100)

plot(x, col = true_clusters, pch = 19)
```



8.5.2 Choose the parameters

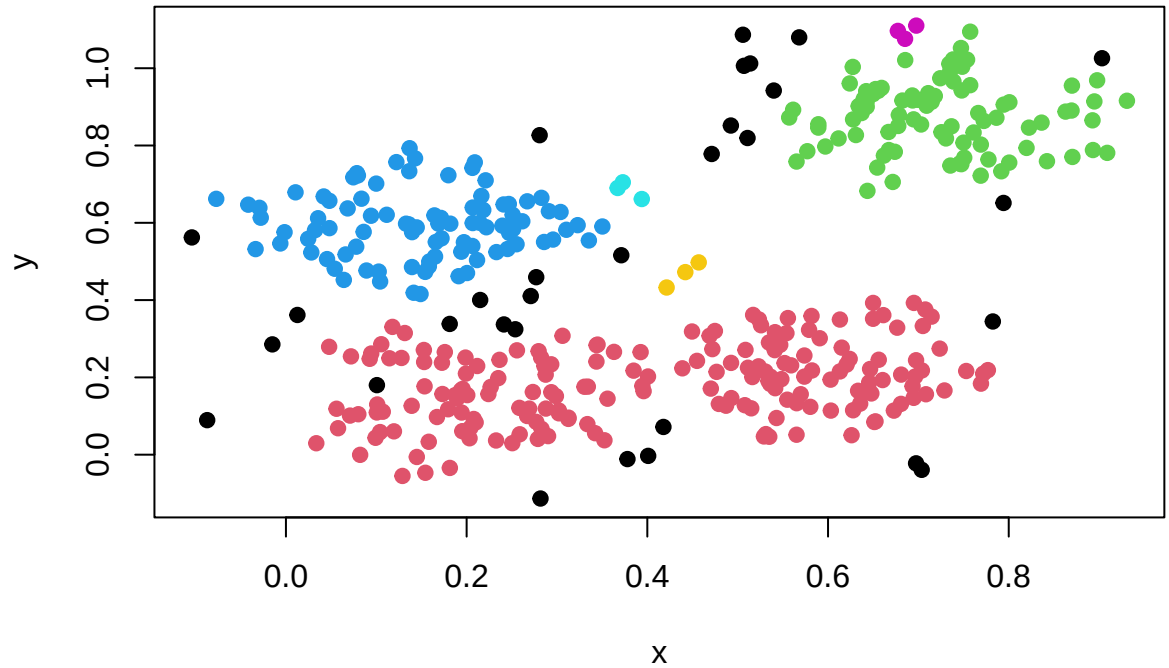
```
# Choice of hyperparameters for DBSCAN
#! - eps: radius of the neighborhood
#! - MinPts: local density around points
# Rule of thumb, minPts = dimensionality+1 = p+1 = 3 here
#! But the choice is difficult, so always tune them with different trials

# How to choose eps from minPts?
# Plot of the distances to the minPts nearest neighbor, for each point of the dataset
kNNdistplot(x, k = 3)
#! Always something like this: we have lots of points which are close, and fewer points
#! which will be in less and less areas (like obs and tails of a distribution)

# Taking eps = 0.05 seems to be a good threshold
#! The higher eps we choose, the more points will be classified as core by DBScan
#! And we are ok to having an higher number of core points
abline(h = 0.05, col = "red", lty = 2)
```




```r
# Run the dbscan
dbs = dbscan(x, eps = 0.05, minPts = 3, borderPoints = TRUE)
#! Core points are assigned to just one cluster. But border points may be assigned to
#! multiple clusters. Which to choose? the algorithm does it. If we want instead to
#! directly discard them we set that option to FALSE
dbs
```

```
## DBSCAN clustering for 400 objects.
## Parameters: eps = 0.05, minPts = 3
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 6 cluster(s) and 30 noise points.
##
##   0   1   2   3   4   5   6
##  30 185  87  89   3   3   3
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

```r
# Plot of the resulting clustering
plot(x, col = dbs$cluster + 1L, pch=19)
```




```
#! Maybe we got unidentified points (azzurri e gialli) che vorremo non fossero singoli
#! cluster, ma assignati. Quindi potremmo aumentare minpts per esempio, per rilassare la
#! richiesta sull'alta densità

#! But DBScan is very hard to tune, is very sensitive to changes in parameters

#! Anyway DBScan is very good when we have complex shapes (spirali, figure concave, ecc)
#! So here maybe DBScan is not very useful wrt other techniques

# How to tune the algorithm and find the "best" eps and minPts?
# We can consider grid search BUT we need an objective function
# In the absence of the true labels (which would be the case in practice),
# we have to use an "intrinsic" clustering quality measure
```

8.5.3 Choose with silhouette score

```
# Silhouette score (from the package "cluster")
# help(silhouette)
#! For each point i in the dataset we compute
#! sil(i) = ( b(i)-a(i) ) / ( max(a(i),b(i)) ) in [-1,1]
#!  - a(i): average distance from unit i to points of his cluster
#!  - b(i): average distance from unit i to points ot the closest cluster

#! Silhouette is useful also when clustering variables and not obs. For obs ie
#! units however it says "how much units are well embedded in their cluster"

# Let's compute the silhouette score on the clustering performed before
# WARNING (specific to DBSCAN): We need to remove the noise points as they do
# not belong to a cluster, before computing the silhouette score
#! However look at how many points are we discarding
print(paste("Removing noise, we are left with",
            sum(dbs$cluster != 0),"obs out of",length(dbs$cluster)))
```

```
## [1] "Removing noise, we are left with 370 obs out of 400"
```

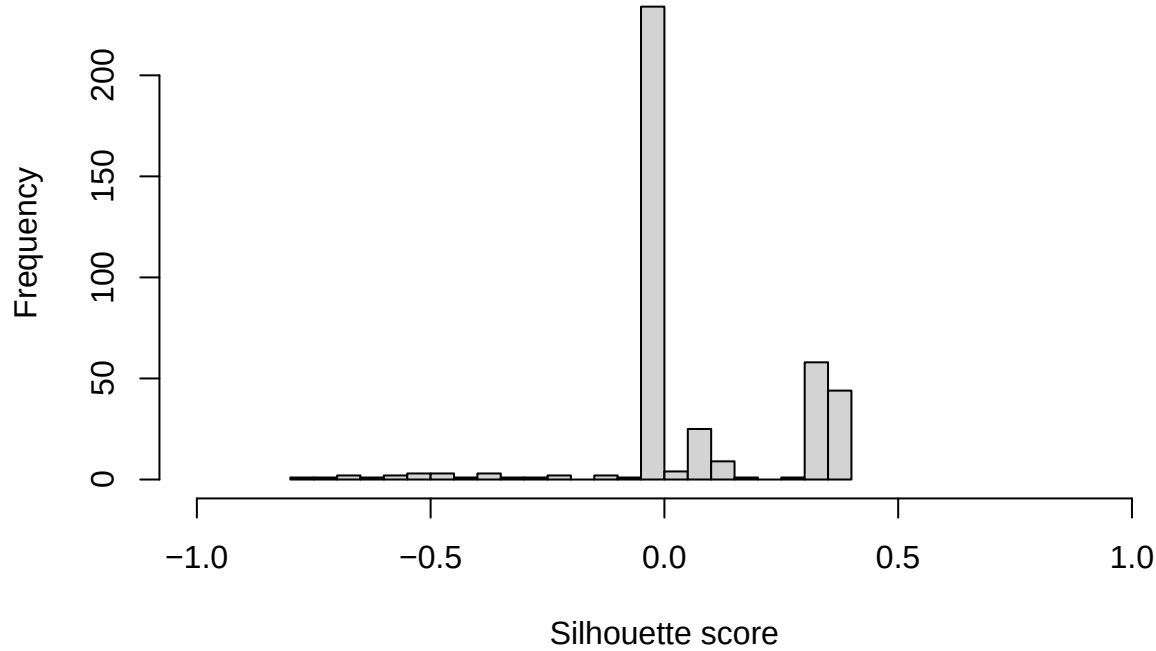
```r
clustered_index = which(dbs$cluster != 0)       # Index of non noise points
clustered_points = x[clustered_index]           # only clustered points
clustered_labels = dbs$cluster[clustered_index] # corresponding labels

sil = silhouette(clustered_labels, dist(clustered_points))
summary(sil)
```

```
## Silhouette of 370 units in 6 clusters from silhouette.default(x = clustered_labels, dist = dist(clustered_p
##  Cluster sizes and average silhouette widths:
##        185         87         89          3          3          3
## -0.6875638 -0.3220144  0.4201064  0.6874261  0.8207990  0.5853787
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.9711 -0.7059 -0.5241 -0.3015  0.1504  0.8662
```

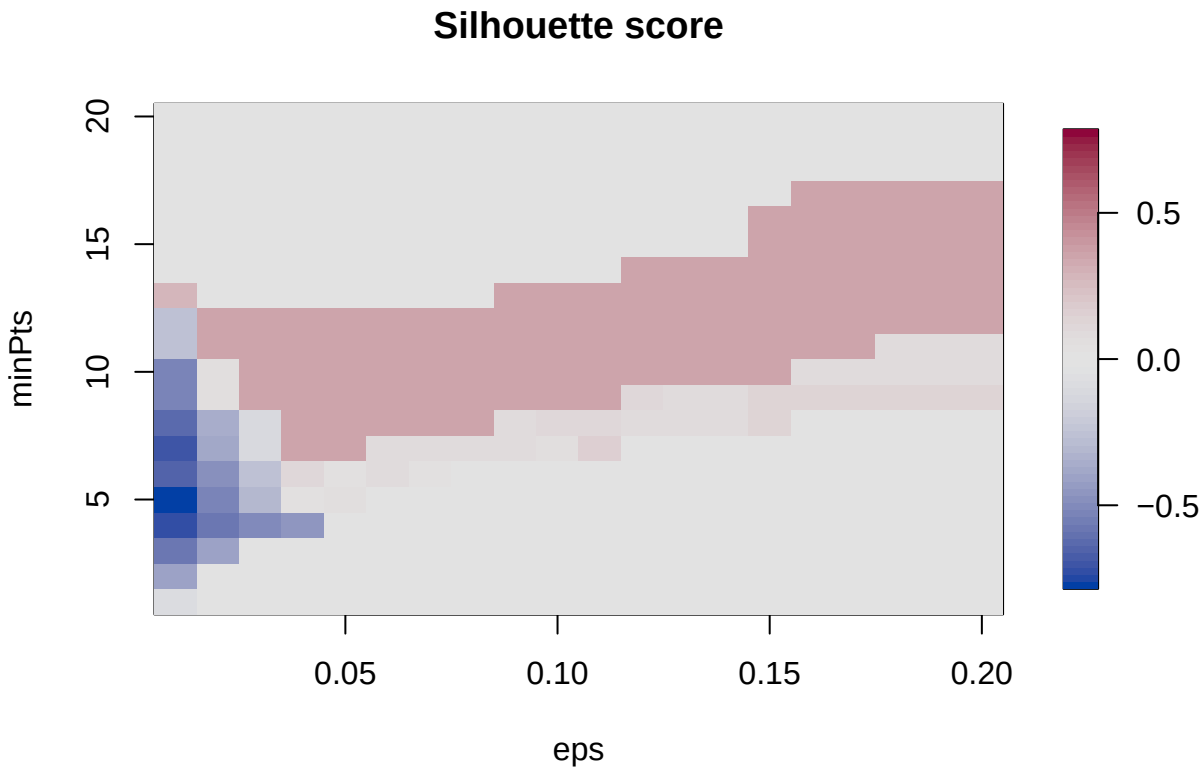
```r
sil_score = function(labels, dist) {
  # Compute the average of the silhouette widths
  sil = silhouette(labels, dist)
  sil_widths = sil[,"sil_width"]
  mean(sil_widths)
}

sil_score(clustered_labels, dist(clustered_points))
```

```
## [1] -0.301471
```

```r
# Grid Search
minPts_grid = 1:20
eps_grid = seq(0.01, 0.2, by = 0.01)

max_share_noise = 0.2

dbscan_perf = function(minPts, eps) {
  # Compute the silhouette score resulting from dbscan clustering
  dbs = dbscan(x, eps, minPts) # Run dbscan

  clustered_index = which(dbs$cluster != 0) # Index of non noise points
  clustered_points = x[clustered_index] # only clustered points
  clustered_labels = dbs$cluster[clustered_index] # corresponding labels
  nb_clusters = length(unique(clustered_labels))

  if ((nb_clusters > 1 & nb_clusters < n) &
      (length(which(dbs$cluster == 0))/n < max_share_noise)) {
    # Silhouette score is defined only if 2 <= nb_clusters <= n-1
    sil_score(clustered_labels, dist(clustered_points))
  }

  else {
    # otherwise we return 0 which would be the approx. value of the silhouette
    # score if the clusters were completely overlapping
    0
  }
}

# We compute the silhouette score for all combinations of minPts and eps
perf_grid = outer(minPts_grid, eps_grid, FUN = Vectorize(dbscan_perf))
dimnames(perf_grid) = list(minPts_grid, eps_grid)

# Histogram of the Silhouette scores

hist(perf_grid, breaks = 20, xlab = "Silhouette score", xlim = c(-1, 1), main = NULL)
```




```
max_score = max(perf_grid)
min_score = min(perf_grid)
max_abs = max(abs(max_score), abs(min_score))


windows.options(height = 6.5, width = 6.5)
image.plot(x = eps_grid, y = minPts_grid, z = perf_grid, xlab = "eps", ylab = "minPts",
           main = 'Silhouette score', col = hcl.colors(64, palette = 'Blue-Red'),
           breaks = c(seq(-max_abs, 0, length=33)[-33], seq(0, max_abs, length=33)))
```




```
#! We can see that in the diagonal, with red color, we have good combinations of values (eps,minpts)

# Retrieve best parameter values
max_score = max(perf_grid)
argmax_score = which(perf_grid == max_score, arr.ind = TRUE)
best_eps = eps_grid[argmax_score[2]]
best_minPts = minPts_grid[argmax_score[1]]
best_eps
```

```
## [1] 0.08
```

```
best_minPts
```

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

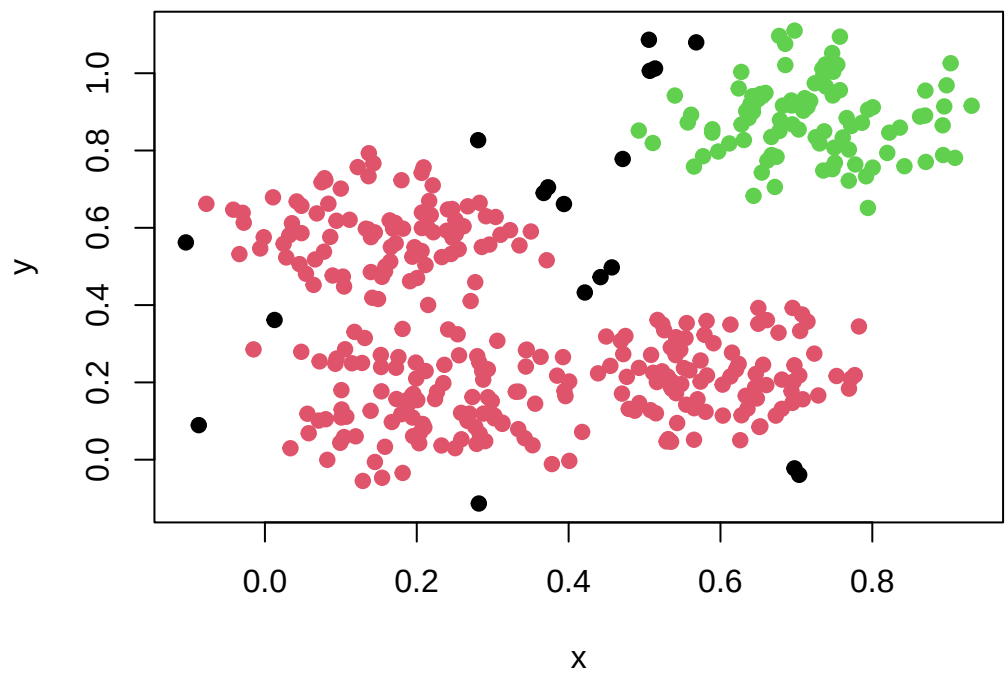
```
max_score
```

```
## [1] 0.3636846
```

```
# Run the dbscan
dbs = dbscan(x, best_eps, best_minPts)
dbs
```

```
## DBSCAN clustering for 400 objects.
## Parameters: eps = 0.08, minPts = 7
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 2 cluster(s) and 18 noise points.
##
##   0   1   2
##  18 287  95
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

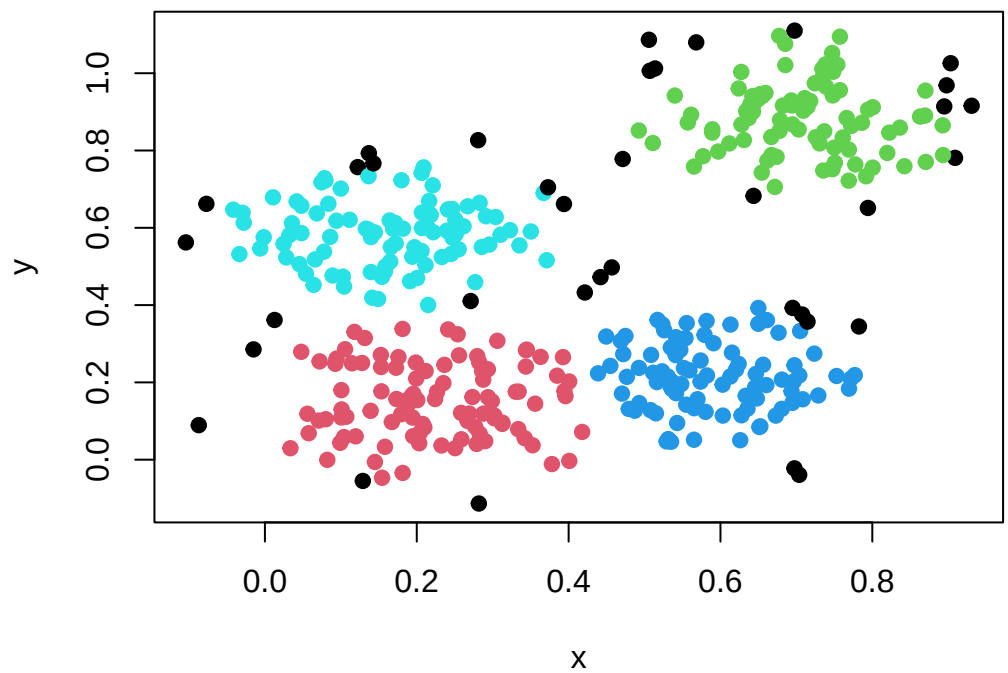
```
plot(x, col = dbs$cluster + 1L, pch=19)
```




```r
#! Well we are not really so happy, maybe we wanted 4 clusters
#! But also two groups clustering actually seems to make sense
#! => Silhouette coeff is not an absolute perfect index of performanca

# Let's try now with eps = 0.09 and minPts = 15
dbs = dbscan(x, eps = 0.09, minPts = 15)
dbs
```

```
## DBSCAN clustering for 400 objects.
## Parameters: eps = 0.09, minPts = 15
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 4 cluster(s) and 36 noise points.
##
##  0  1  2  3  4
## 36 99 87 89 89
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

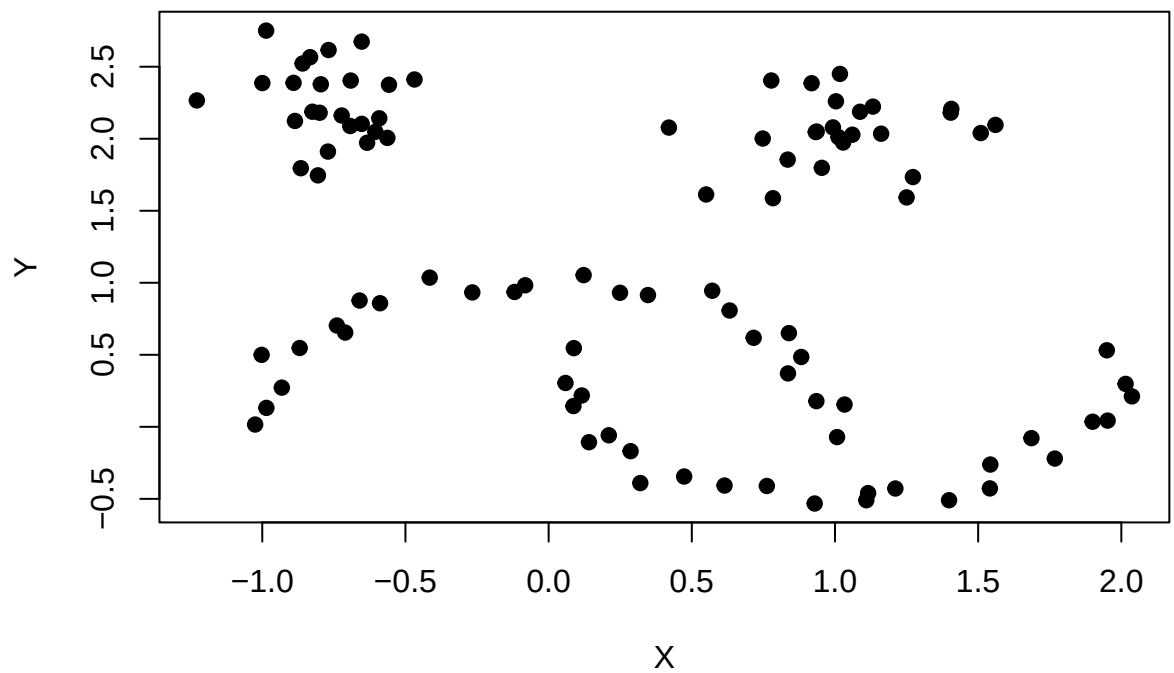
```r
# Recovered the original clusters!

plot(x, col = dbs$cluster + 1L, pch=19)
```



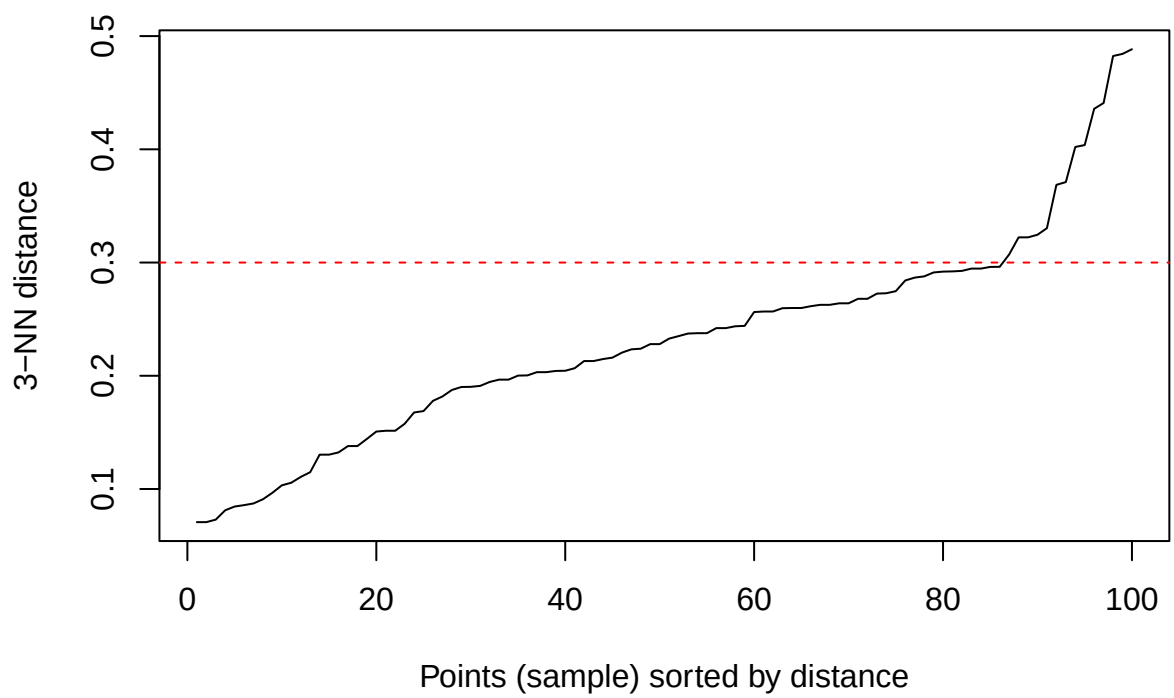

```
#! Try using the previous techniques (linakges, kmeans, ecc) on this dataset
#! which se think will work better
```

8.6 On dataset moon

```
#! (good DBScan, bad kmeans and hclust)
#-------------------------------------------------------------------------------
# Example of dataset where DBSCAN succeed and k-means & hierarchical fail

data("moons")
plot(moons, pch=19)
```



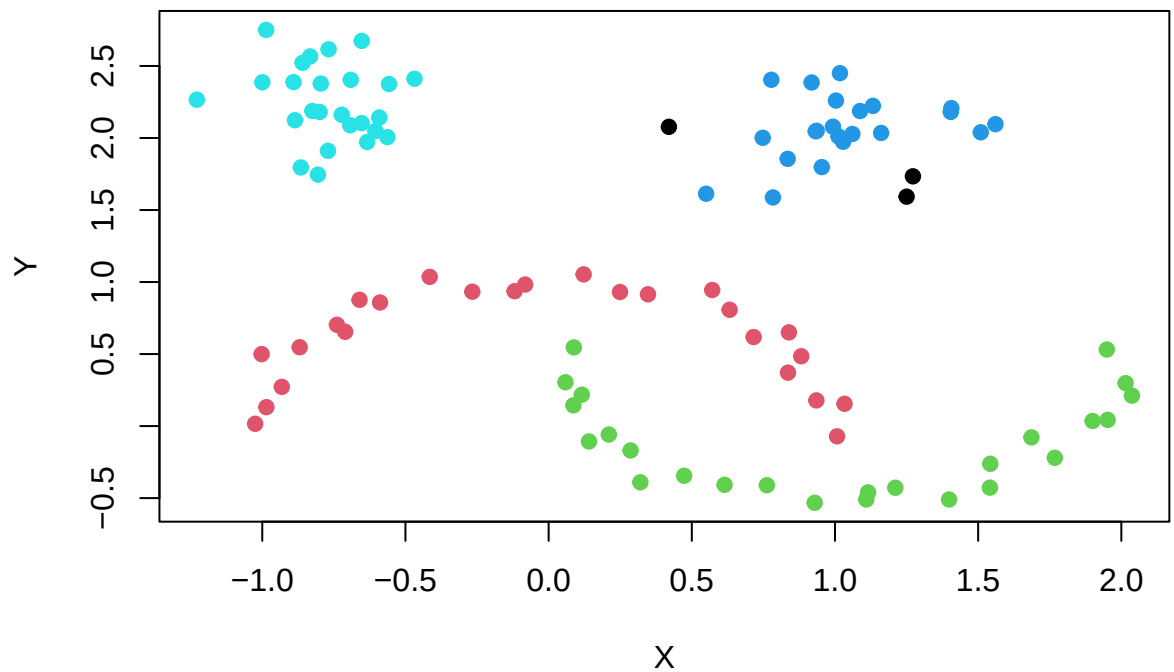

```
minPts = 3 # Dimensionality + 1

kNNdistplot(moons, k = 3)
abline(h = 0.3, col = "red", lty = 2)
```



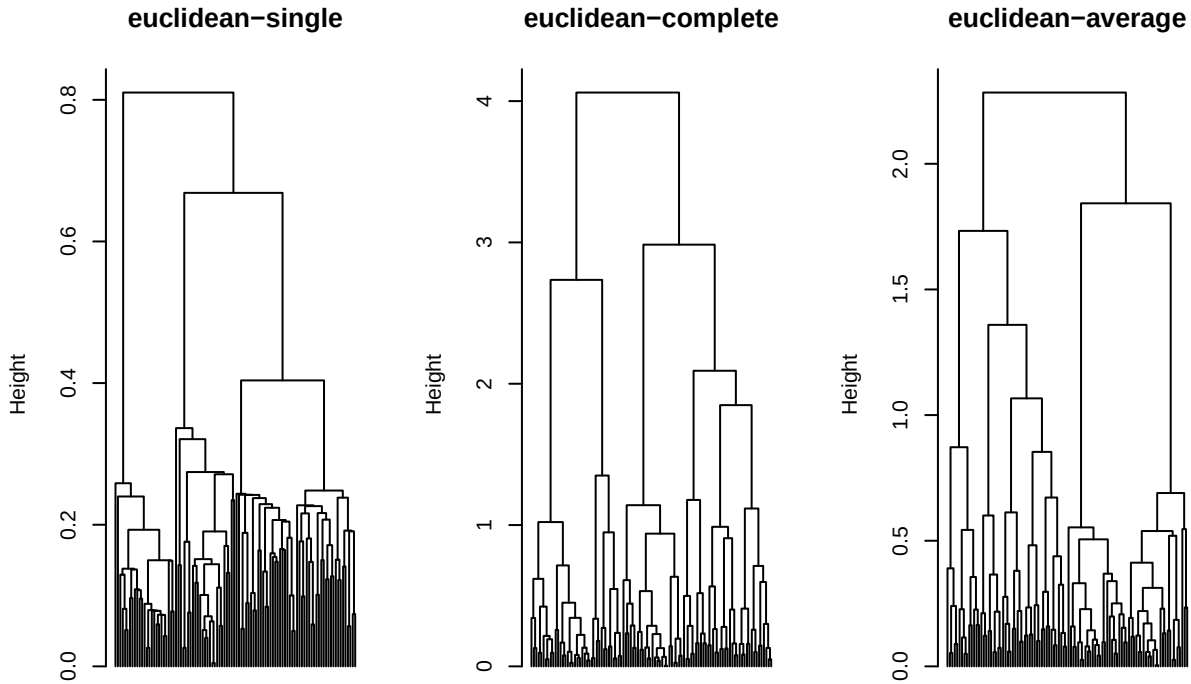
```r
eps = 0.3
minPts = 3

dbs = dbscan(moons, eps, minPts)
dbs
```

```
## DBSCAN clustering for 100 objects.
## Parameters: eps = 0.3, minPts = 3
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 4 cluster(s) and 3 noise points.
##
##  0  1  2  3  4
##  3 25 25 22 25
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

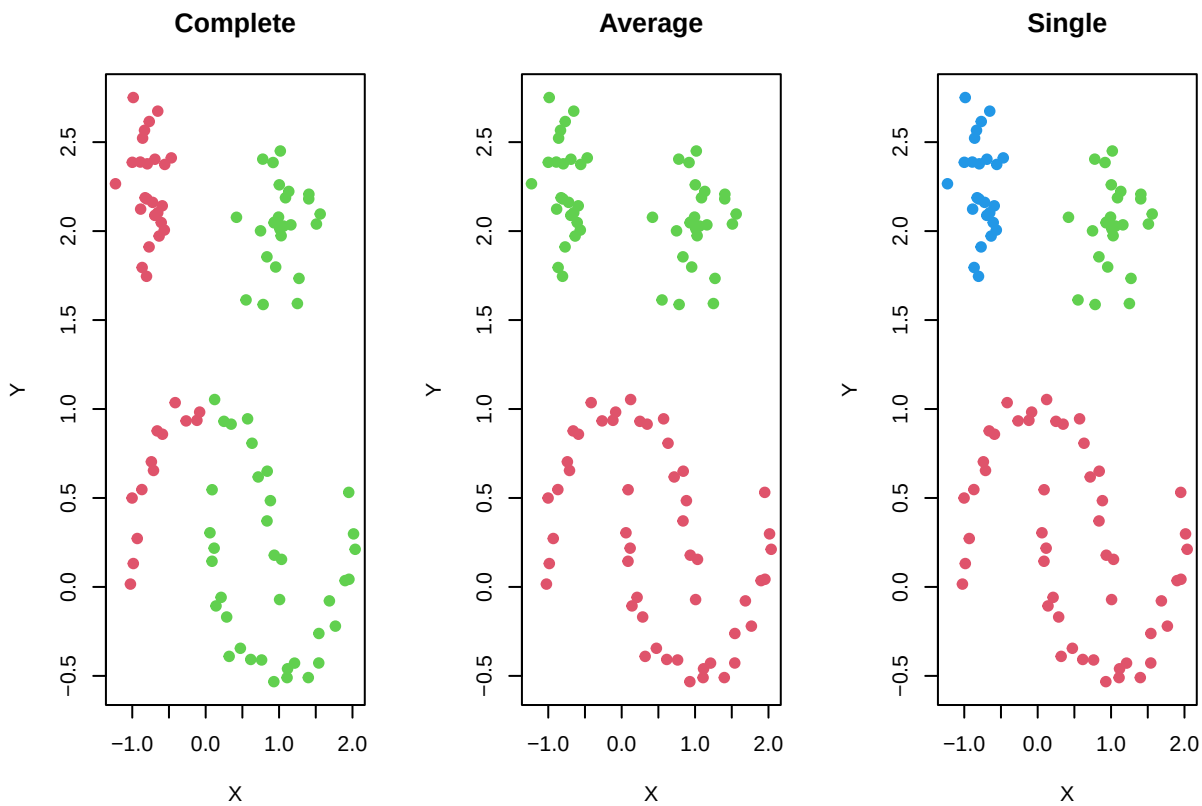
```r
plot(moons, col = dbs$cluster + 1L, pch=19)
```



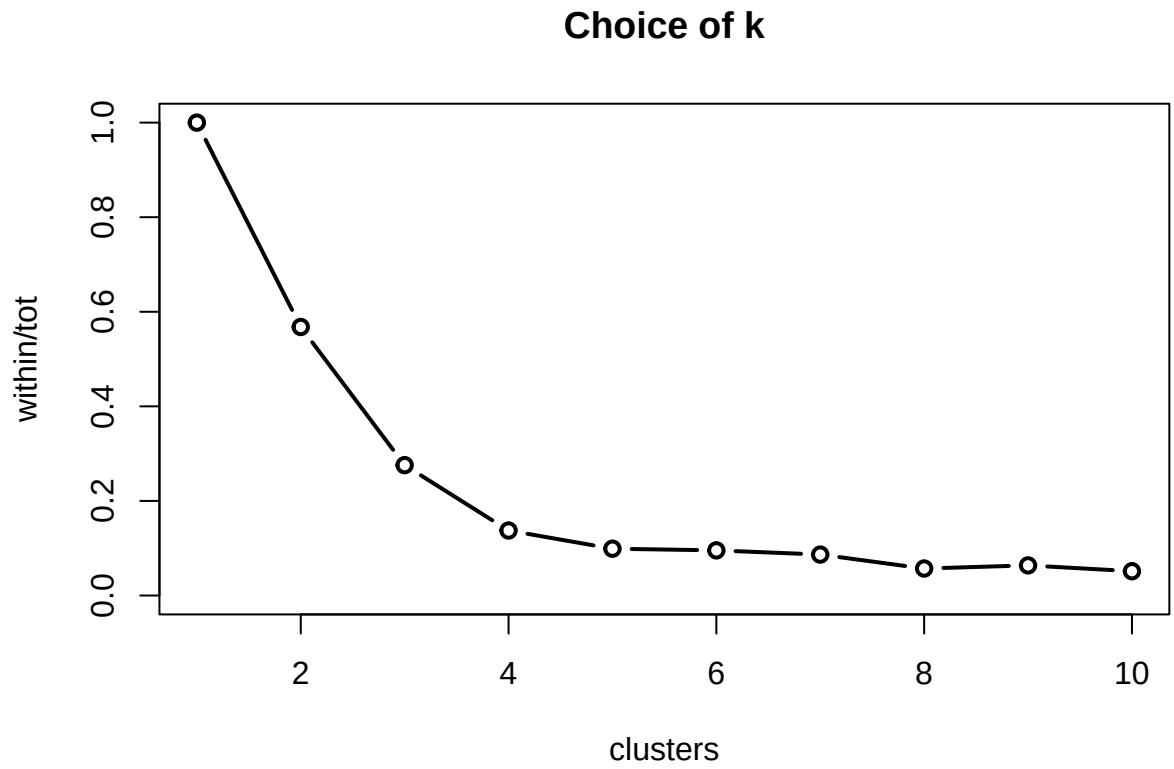

```r
# Let's try the other algorithms that we saw:
hclust.s = hclust(dist(moons), method='single')
hclust.a = hclust(dist(moons), method='average')
hclust.c = hclust(dist(moons), method='complete')


# plot of the dendrograms
par(mfrow=c(1,3))
plot(hclust.s, main='euclidean-single', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
plot(hclust.c, main='euclidean-complete', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
plot(hclust.a, main='euclidean-average', hang=-0.1, xlab='', labels=F, cex=0.6, sub='')
```



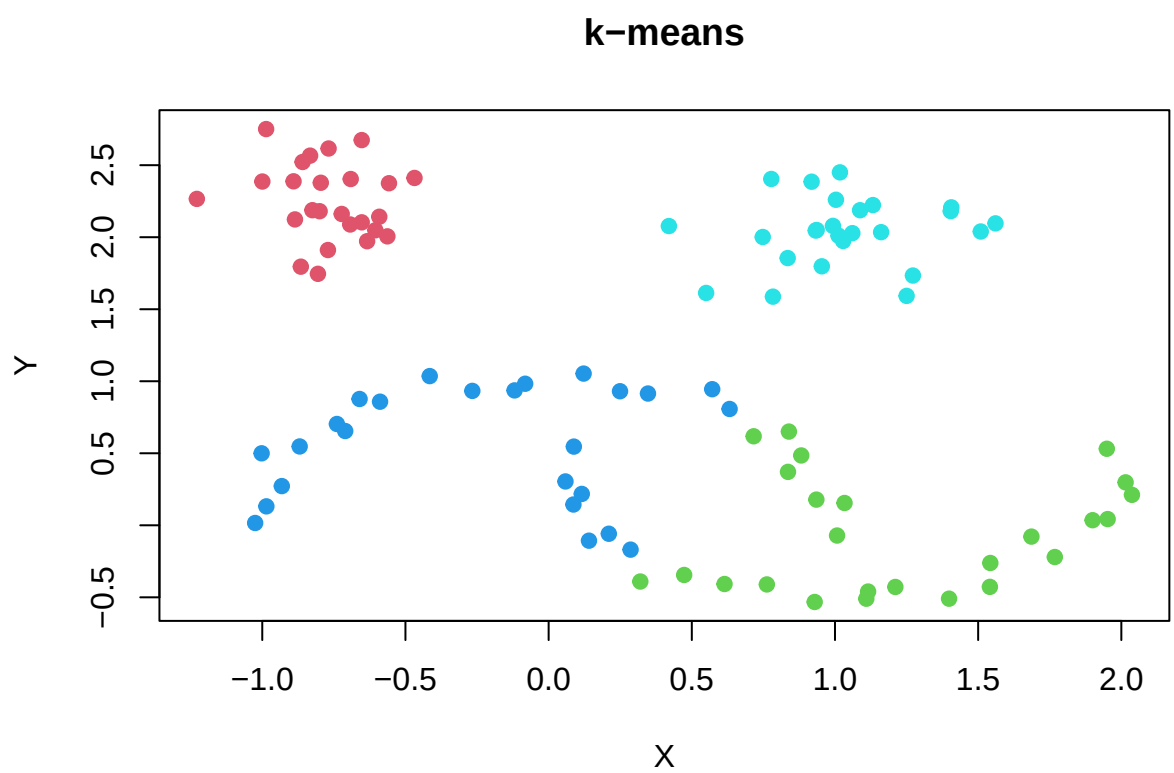

```r
# Let's look at:
# - single linkage with three clusters
# - complete linkage with two clusters
# - average linkage with two clusters

cluster.c = cutree(hclust.c, k=2) # euclidean-complete:
cluster.s = cutree(hclust.s, k=3) # euclidean-single
cluster.a = cutree(hclust.a, k=2) # euclidean-average

par(mfrow=c(1,3))
plot(moons, col=cluster.c + 1L, pch=19, main='Complete')
plot(moons, col=cluster.a + 1L, pch=19, main='Average')
plot(moons, col=cluster.s + 1L, pch=19, main='Single')
```




```r
# k-means
# Choose k -> elbow method
b = NULL
w = NULL
for(k in 1:10){
  k.means = kmeans(moons, k)
  w = c(w, sum(k.means$wit))
  b = c(b, k.means$bet)
}

matplot(1:10, w/(w+b), pch='', xlab='clusters', ylab='within/tot', main='Choice of k',
        ylim=c(0,1))
lines(1:10, w/(w+b), type='b', lwd=2)
```




```
# -> 4 clusters


k.means = kmeans(moons, centers = 4, nstart = 25)
plot(moons, col=k.means$cluster + 1L, pch=19, main='k-means')
```



```
#! Moral: DBScan is able to capture more complex shapes wrt other clasical methods

#! But of course there is no THE method, we have to study and practice a lot to learn
#! the insights, gain experience, ecc
```

8.7 Exam exercise

```
#_____________________________________________________________________________
##### Exercises

##### Problem 3 of July 1, 2009
#####-----------------------------
# The Veritatis Diagram is a mysterious work attributed to Galileo. Some
# semiologists believe that some pages of the book hide a coded message;
# they also believe that these pages are characterized by an abnormal
# numerosity of some letters of the alphabet. The veritatis.txt file
# lists, for 132 pages of the book, the absolute frequencies of the five
# vowels of the Latin alphabet.
# a) By using an agglomerative clustering algorithm (Manhattan distance,
#    average linkage), identify two clusters and report suspicious pages;
# b) assuming that, for each of the two clusters, the five absolute
#    frequencies are (approximately) normally distributed with the same
#    covariance matrix perform a test to prove the existence of a difference
#    in the mean value of the two distributions;
# c) using five Bonferroni intervals of global confidence 90%, comment the
#    results of the test at point (b).

vowels = read.table('Lab08/veritatis.txt', header=T)
head(vowels)
```

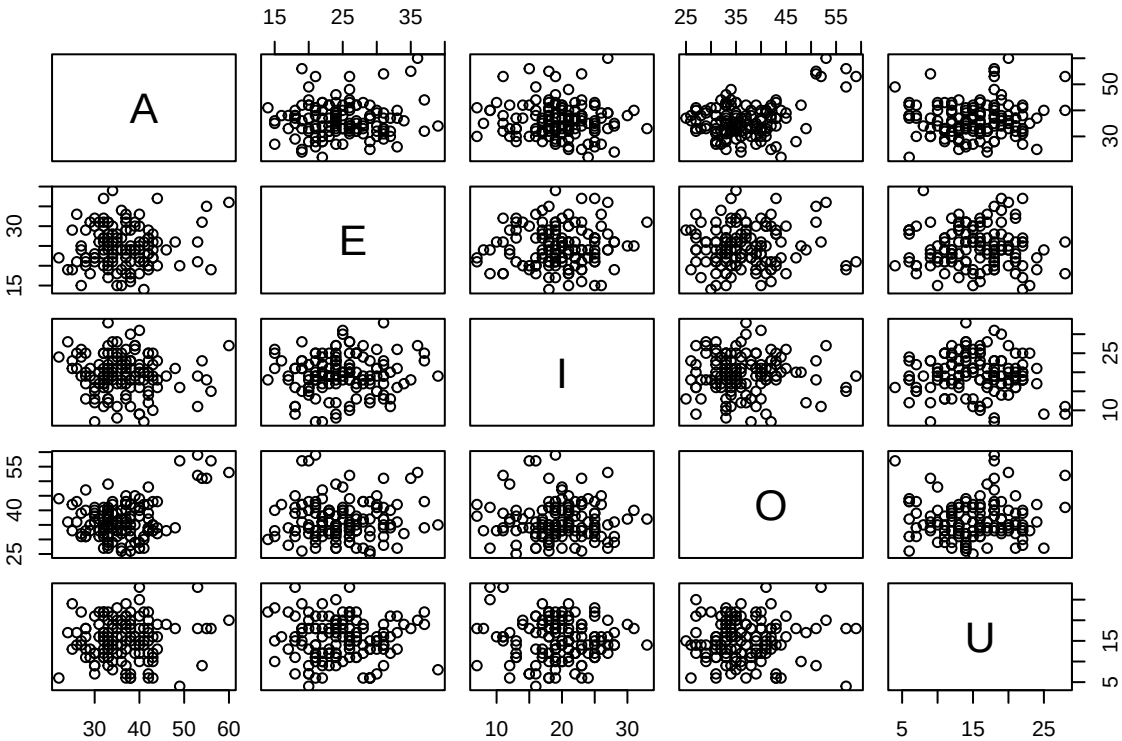
```
##    A  E  I  O  U
## 1 41 26 20 31 15
## 2 37 29 19 43  6
## 3 32 32 11 34 16
## 4 38 16 23 42 16
## 5 33 28 27 27 14
## 6 38 24 23 38 18
```

```
dim(vowels)
```

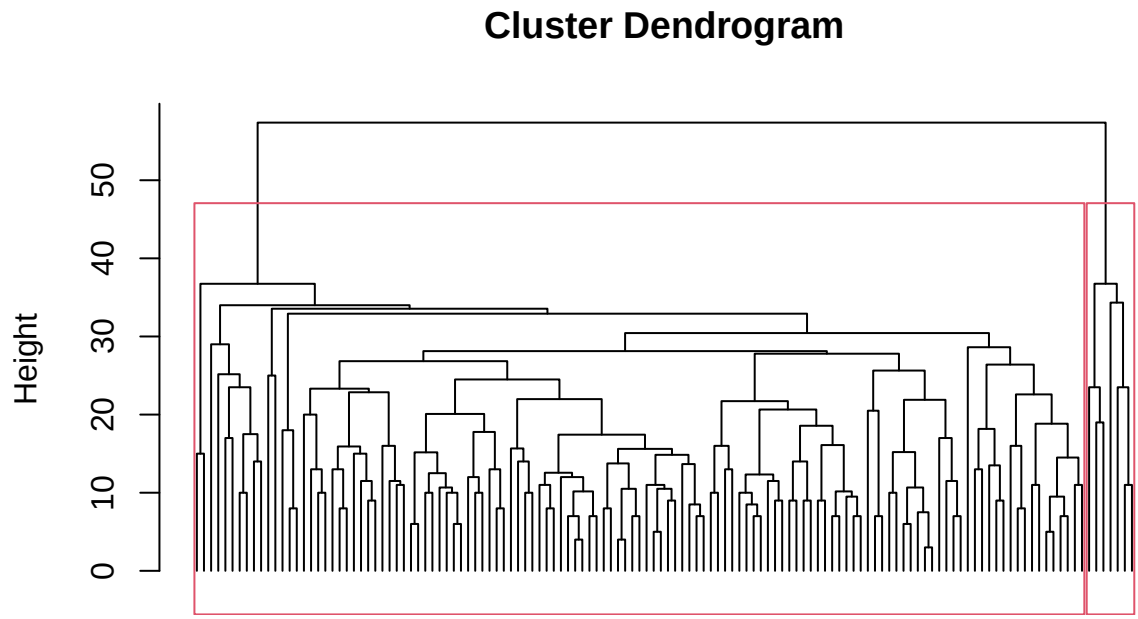
```
## [1] 132   5
```

8.7.1 Qa

```
### question a)
# a) By using an agglomerative clustering algorithm (Manhattan distance,
#    average linkage), identify two clusters and report suspicious pages;

plot(vowels)
```




```
HC = hclust(dist(vowels, method='manhattan'), method = 'average')

plot(HC, hang=-0.1, sub='', labels=F, xlab='')

# we cut the dendrogram at k=2 clusters
rect.hclust(HC, k=2)
```



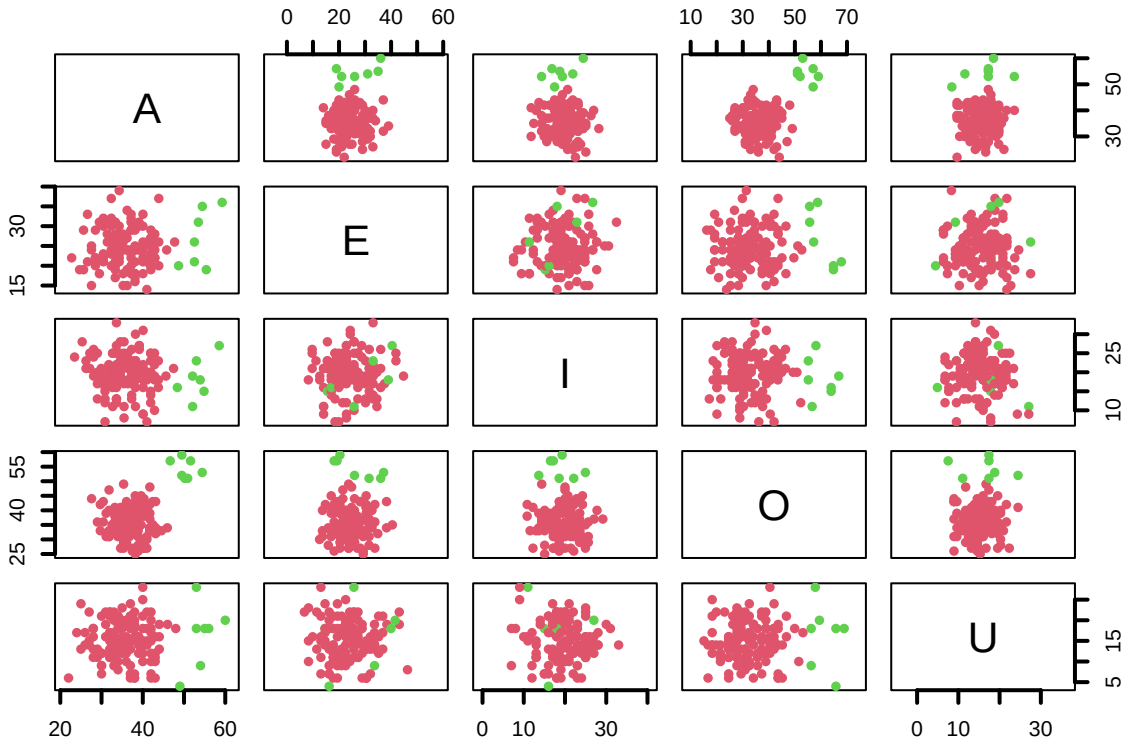

```
pag = cutree(HC, k=2)
table(pag)
```

```
## pag
##   1   2
## 125   7
```

```
which(pag==2)
```

```
##  33  54  79  93 102 105 117
##  33  54  79  93 102 105 117
```

```
plot(vowels, col=pag+1, asp=1, pch=16, lwd=2)
```

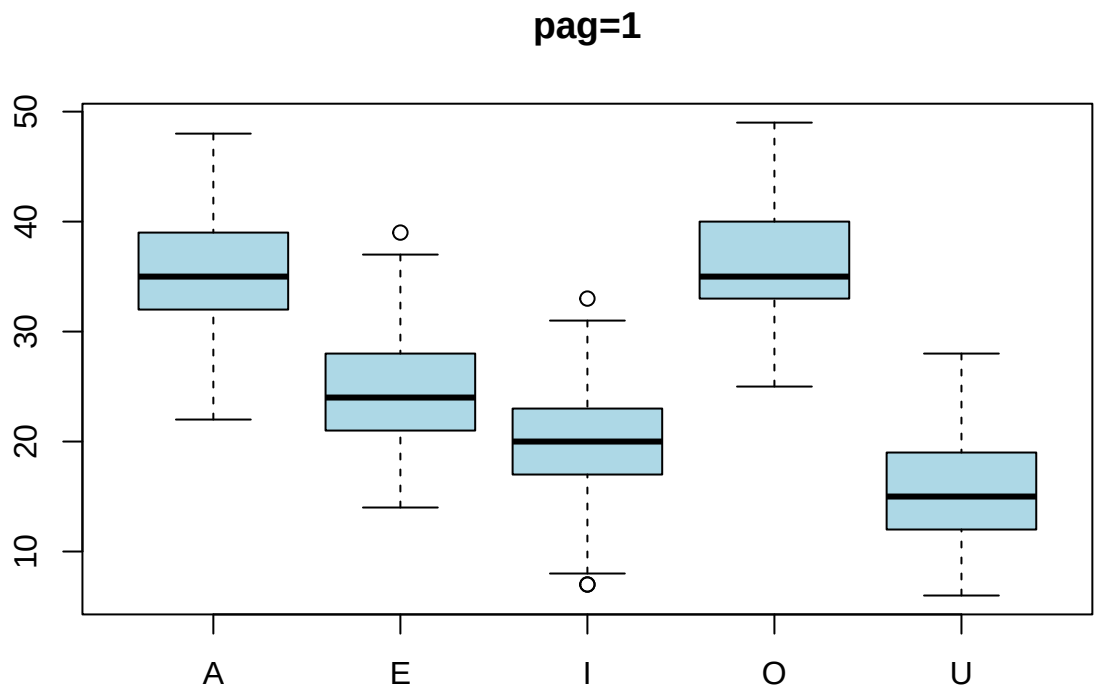


8.7.2 Qb

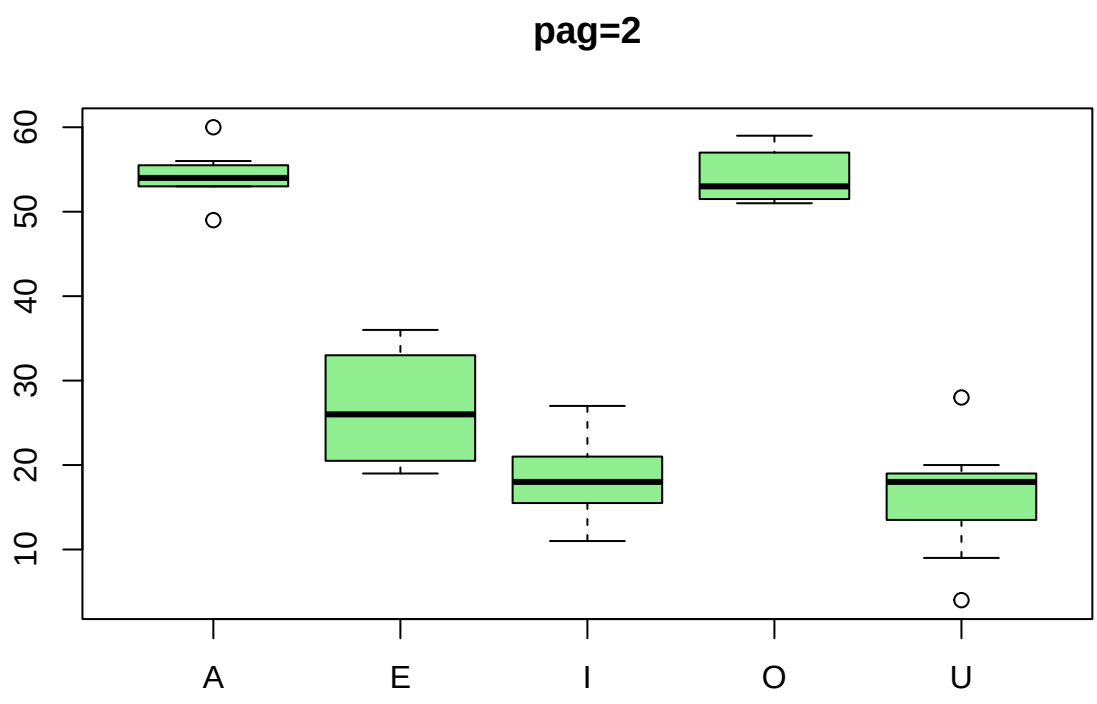
```
### question b)
# b) assuming that, for each of the two clusters, the five absolute
#    frequencies are (approximately) normally distributed with the same
#    covariance matrix perform a test to prove the existence of a difference
#    in the mean value of the two distributions;

p  = 5
n1 = table(pag)[1]
n2 = table(pag)[2]

# Verify gaussianity
mvn(vowels[pag=='1',])$multivariateNormality
```

```
##            Test       HZ    p value MVN
## 1 Henze-Zirkler 1.016064 0.02942827  NO
```

```
# mvn(vowels[pag=='2',])$multivariateNormality

boxplot(vowels[pag=='1',], col="lightblue", main = "pag=1")
```




```
boxplot(vowels[pag=='2',], col="lightgreen", main = "pag=2")
```



```r
# Test for independent Gaussian populations
t1.mean = sapply(vowels[pag=='1',],mean)
t2.mean = sapply(vowels[pag=='2',],mean)
t1.cov  =  cov(vowels[pag=='1',])
t2.cov  =  cov(vowels[pag=='2',])
Sp      = ((n1-1)*t1.cov + (n2-1)*t2.cov)/(n1+n2-2)

# Test: H0: mu.1-mu.2==0 vs H1: mu.1-mu.2!=0
delta.0 = c(0,0,0,0,0)
Spinv   = solve(Sp)
T2 = n1*n2/(n1+n2) * (t1.mean-t2.mean-delta.0) %*% Spinv %*% (t1.mean-t2.mean-delta.0)
P = 1 - pf(T2/(p*(n1+n2-2)/(n1+n2-1-p)), p, n1+n2-1-p)
P
```

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

8.7.3 Qc

```r
### question c)
# c) using five Bonferroni intervals of global confidence 90%, comment the
#    results of the test at point (b).

alpha = 0.1
IC = cbind(t2.mean-t1.mean - sqrt(diag(Sp)*(1/n1+1/n2)) * qt(1 - alpha/(p*2), n1+n2-2),
           t2.mean-t1.mean,
           t2.mean-t1.mean + sqrt(diag(Sp)*(1/n1+1/n2)) * qt(1 - alpha/(p*2), n1+n2-2))
IC
```

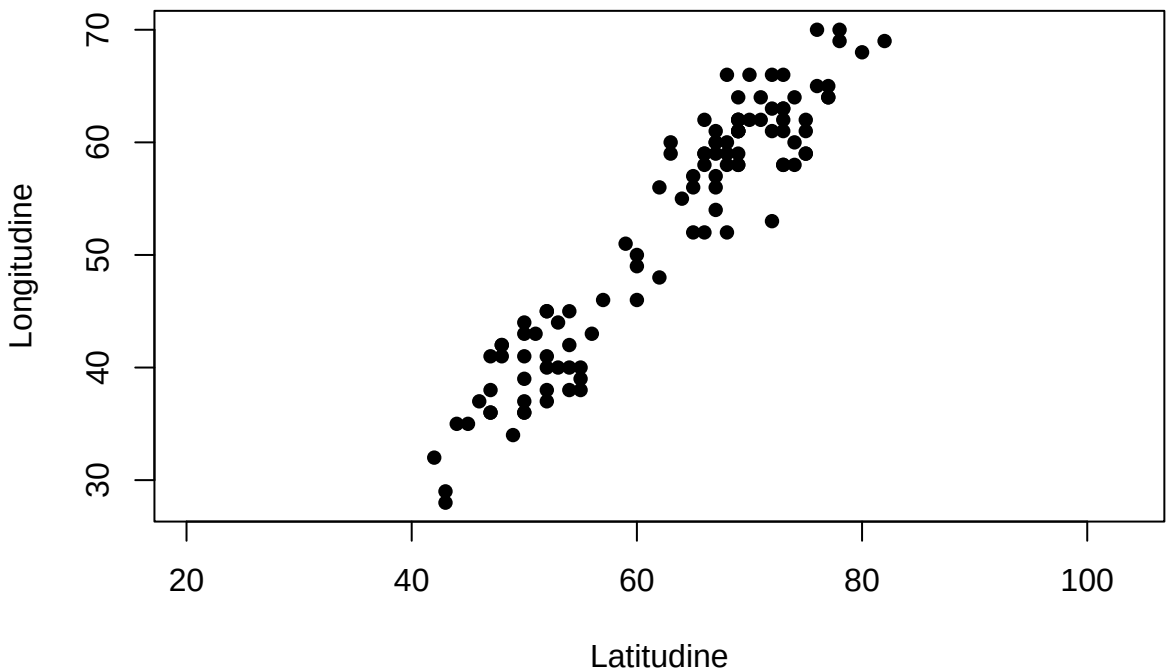
```
##        [,1]      [,2]      [,3]
## A 14.267514 18.949714 23.631915
## E -2.418594  2.201143  6.820880
## I -5.791398 -1.195429  3.400541
## O 13.936536 18.565714 23.194893
## U -3.277573  1.244571  5.766716
```

8.8 Exam exercise

```r
#_____________________________________________________________________________
##### Problem 2 of February 18, 2009
#####--------------------------------
# Friday, October 17, 2008, in Black Fortune skies a crash occurred between
# two artificial satellites. About a hundred debris were found on the
# ground (satellite.txt file). However, due to friction with the atmosphere it
# was not possible to define the origin of any debris. To clarify the
# accident, the US Air Force requests to estimate the relative position of
# the two points of impact on the ground.
# Assuming that the debris coming from the same satellite are scattered on
# the ground according to a normal law of unknown mean and covariance matrix:
# a) attribute the debris to the two satellites via a hierarchical clustering
#    algorithm that uses the Euclidean distance and the average linkage;
# b) report the numerosity of the two clusters and the value of the cophenetic
#    coefficient;
# c) introducing the appropriate hypotheses, identifying the points of impact
#    on the ground with the mean of the two normal distributions and assuming
#    correct the allocation of the debris to the two satellites, provide an elliptical
#    confidence region (global level 99%) for the relative position of the two
#    points of impact on the ground.

satellite = read.table('Lab08/satellite.txt', header=T)
head(satellite)
```

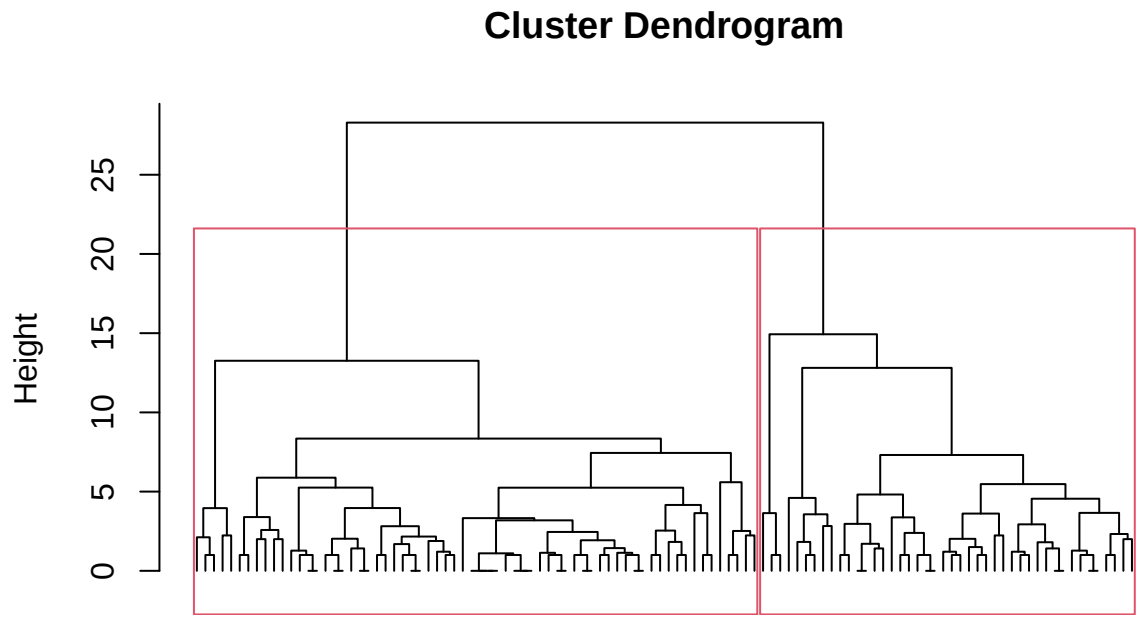
```
##   Latitudine Longitudine
## 1         55          38
## 2         43          29
## 3         56          43
## 4         72          63
## 5         50          44
## 6         75          62
```

```
plot(satellite, asp = 1, pch=16)
```

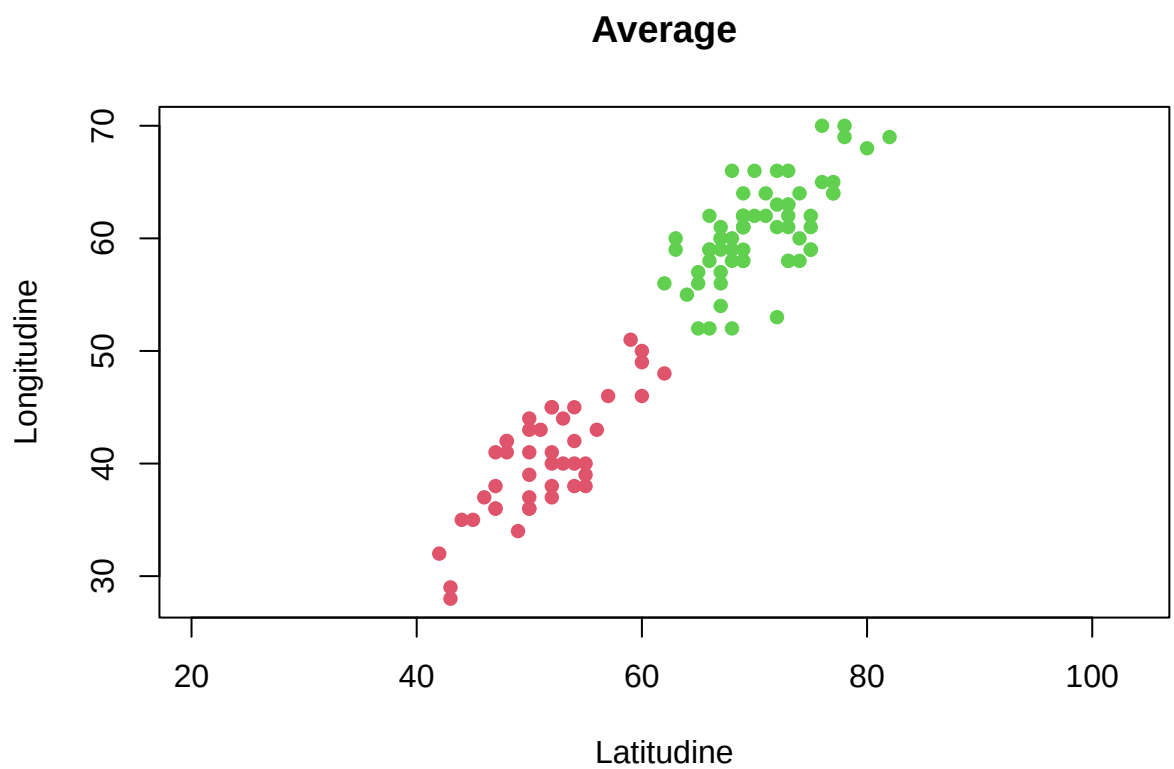


8.8.1 Qa

```
### question a)
# a) attribute the debris to the two satellites via a hierarchical clustering
#    algorithm that uses the Euclidean distance and the average linkage;

D.s = dist(satellite)
HCa = hclust(D.s, method = 'average')
plot(HCa, hang=-0.1, sub='', xlab='', labels=F)

# we know that there are 2 clusters, so we cut the dendrograms
# accordingly:
rect.hclust(HCa, k=2)
```




```
sata = cutree(HCa, k=2)
plot(satellite , col=sata+1, asp=1, pch=16, main='Average')
```



8.8.2 Qb

```r
### question b)
# b) report the numerosity of the two clusters and the value of the cophenetic
#    coefficient;

table(sata)
```

```
## sata
##  1  2
## 44 66
```

```r
coph.a = cophenetic(HCa)
coph.sat = cor(D.s, coph.a)
coph.sat
```

```
## [1] 0.8608342
```

8.8.3 Qc

```r
### question c)
# c) introducing the appropriate hypotheses, identifying the points of impact
#    on the ground with the mean of the two normal distributions and assuming
#    correct the allocation of the debris to the two satellites, provide an elliptical
#    confidence region (global level 99%) for the relative position of the two
#    points of impact on the ground.

p  = 2
n1 = table(sata)[1]
n2 = table(sata)[2]

# Assumptions:
# - Normality
# - Independent populations
# - Homogeneity of covariance structures

# Verificy Gaussian assumption
mvn(satellite[sata=='1',])$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.7265198 0.1151083 YES
```

```r
mvn(satellite[sata=='2',])$multivariateNormality
```

```
##            Test        HZ    p value MVN
## 1 Henze-Zirkler 0.8802319 0.06168594 YES
```

```r
t1.mean = sapply(satellite[sata=='1',],mean)
t2.mean = sapply(satellite[sata=='2',],mean)
t1.cov  =  cov(satellite[sata=='1',])
t2.cov  =  cov(satellite[sata=='2',])

# Homogeneity of covariances
t1.cov
```

```
##             Latitudine Longitudine
## Latitudine    23.33827    19.50740
## Longitudine   19.50740    25.80761
```

```r
t2.cov
```

```
##             Latitudine Longitudine
## Latitudine    19.29627    12.00909
## Longitudine   12.00909    17.40909
```

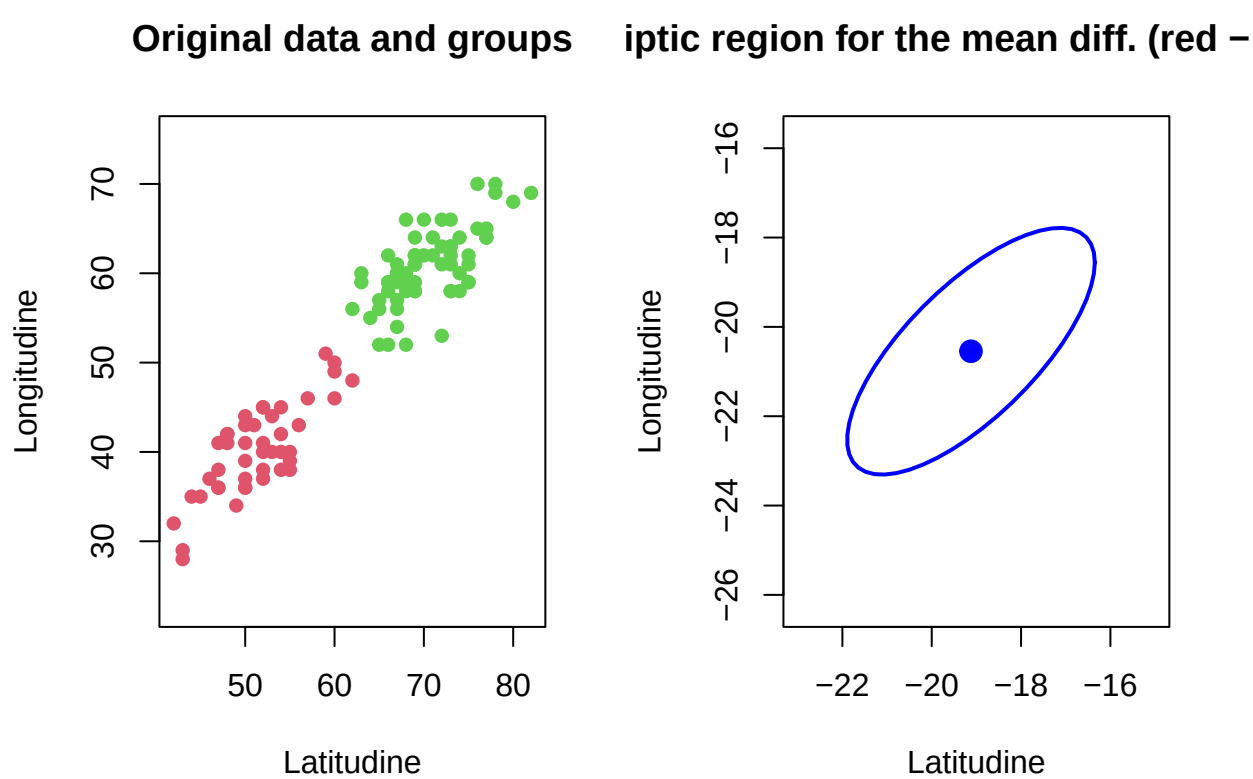
```r
Sp = ((n1-1)*t1.cov + (n2-1)*t2.cov)/(n1+n2-2)

# Elliptic confidence region at 99%
alpha = 0.01
cfr.fisher = (p*(n1+n2-2)/(n1+n2-1-p))*qf(1-alpha, p, n1+n2-1-p)

# Characterize the ellipse:
# Directions of the axes
eigen(Sp)$vector
```

```
##           [,1]      [,2]
## [1,] -0.708904  0.705305
## [2,] -0.705305 -0.708904
```

```r
# Radius
r = sqrt(cfr.fisher)

# Length of the semi-axes
r*sqrt(eigen(Sp)$values*(1/n1+1/n2))
```

```
## [1] 3.629574 1.464779
```

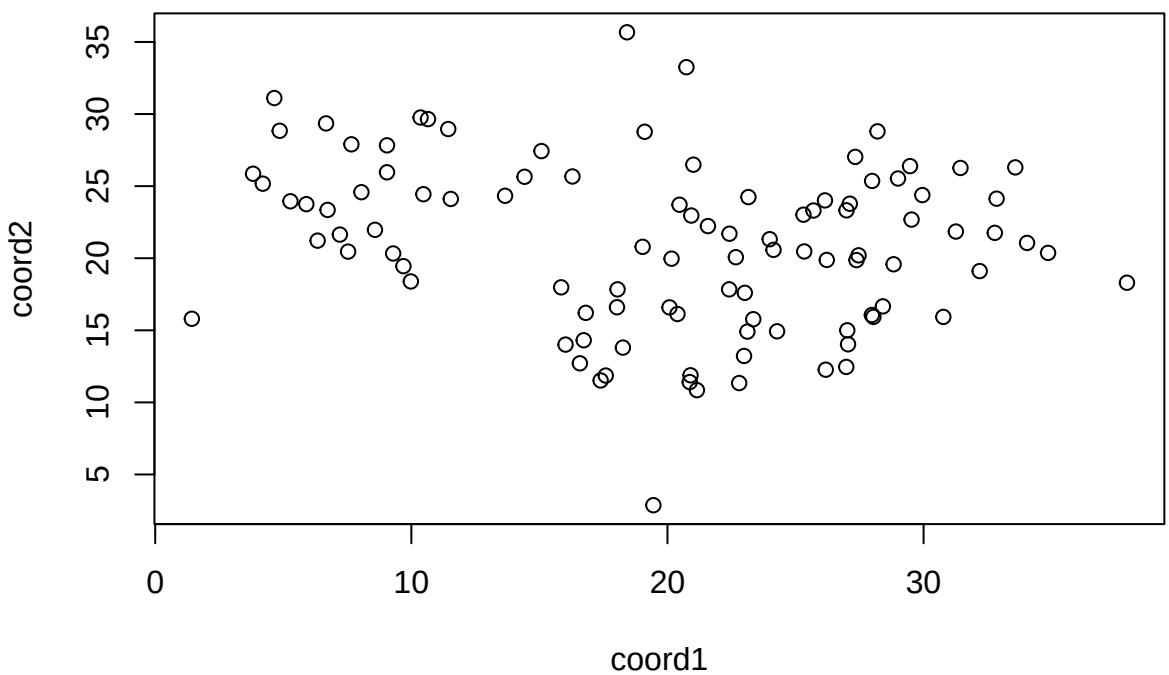
```r
par(mfrow=c(1,2))
plot(satellite , col=sata+1, asp=1, pch=16, main='Original data and groups')
plot(satellite, xlim=c(-23,-15), ylim=c(-25,-17), pch='', asp=1,
     main='Elliptic region for the mean diff. (red - green)')
# confidence region and sample mean in blue
ellipse(center=t1.mean-t2.mean, shape=Sp*(1/n1+1/n2), radius=sqrt(cfr.fisher),
        lwd=2, col='blue')
```



8.9 Exam exercise

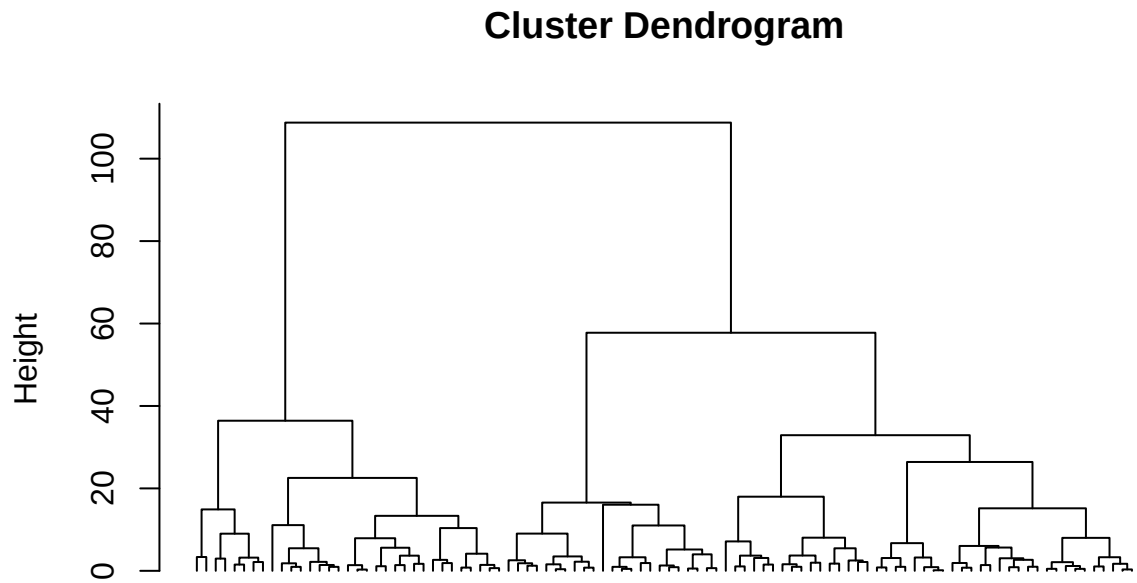
```r
#_____________________________________________________________________________
##### Problem 2 of February 28, 2013
#####--------------------------------
```

```r
# In view of the week of haute couture in Paris, the fashion house La Boutin
# has decided to create a unique-edition pair of shoes decorated with
# gemstones instead of crystals. During the processing of the pair of shoes,
# a bumbling craftsman caused the fall of the gemstones and of a
# box of crystals. The file preziosi.txt collects data related to the Cartesian
# coordinates of the found stones.
# a) Through a hierarchical clustering algorithm (Euclidean distance and
#    Ward linkage), identify the two clusters of stones.
# b) reporting the numerosity of the two groups identified at point a) and
#    the value of the cophenetic coefficient.
# c) Identify the gemstones with the smaller group. Having introduced
#    (and verified) the appropriate assumptions, estimate an elliptical region
#    containing 99% of gemstones (report the center, the length and the direction
#    of the principal axes of the ellipse).

stones = read.table('Lab08/preziosi.txt',header=TRUE)
plot(stones)
```



8.9.1 Qa

```r
# question a)
# a) Through a hierarchical clustering algorithm (Euclidean distance and
#    Ward linkage), identify the two clusters of stones.

p.e = dist(stones, method='euclidean')
p.ew = hclust(p.e, method='ward.D2')

plot(p.ew, hang=-0.1, sub='', xlab='', labels=F)
```




```
cl.ew = cutree(p.ew, k=2) # euclidean-ward
```

8.9.2 Qb

```
# question b)
# b) reporting the numerosity of the two groups identified at point a) and
#    the value of the cophenetic coefficient.

table(cl.ew)
```

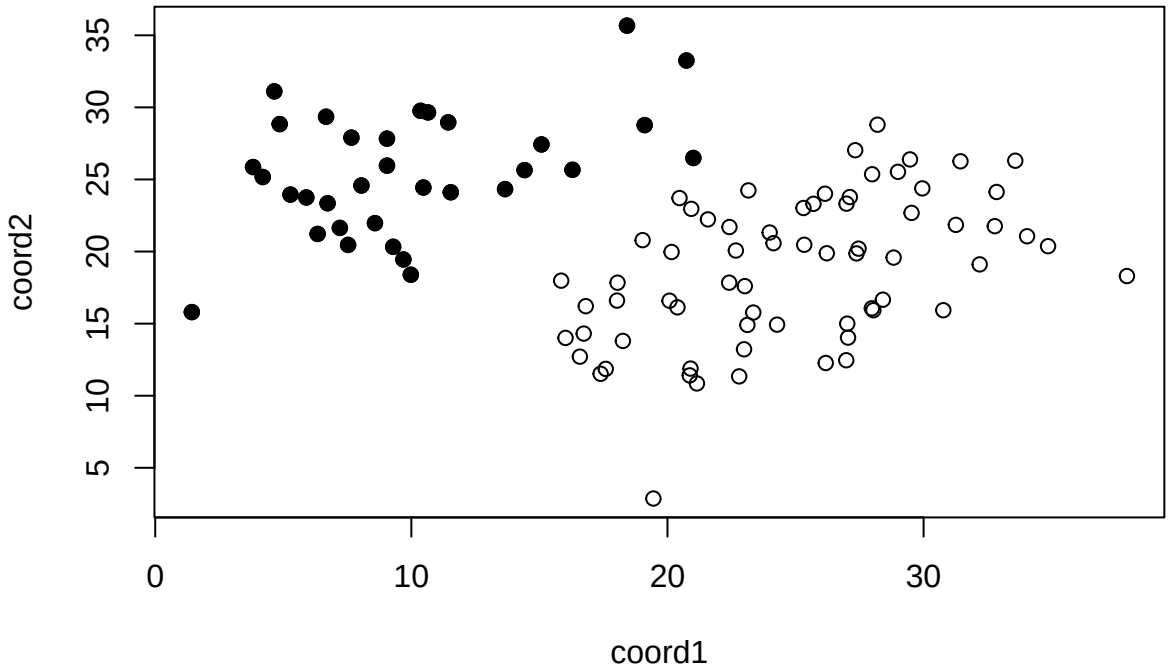
```
## cl.ew
##  1  2
## 67 33
```

```
coph.ew = cophenetic(p.ew)
ew = cor(p.e, coph.ew)
ew
```

```
## [1] 0.7424942
```

8.9.3 Qc

```
# question c)
# c) Identify the gemstones with the smaller group. Having introduced
#    (and verified) the appropriate assumptions, estimate an elliptical region
#    containing 99% of gemstones (report the center, the length and the direction
#    of the principal axes of the ellipse).

p.pr=stones[which(cl.ew==2),]
n = dim(p.pr)[1]
p =dim(p.pr)[2]

plot(stones)
points(p.pr, pch=19)
```




```r
# Verify gaussian assumptions
mvn(p.pr)$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.5164084 0.3240351 YES
```

```r
M = sapply(p.pr, mean)
S = cov(p.pr)
alpha = 0.01
cfr.chisq = qchisq(1-alpha,p)

# Characterize the ellipse:
# Axes directions:
eigen(S)$vectors
```

```
##            [,1]       [,2]
## [1,] -0.8108739  0.5852209
## [2,] -0.5852209 -0.8108739
```

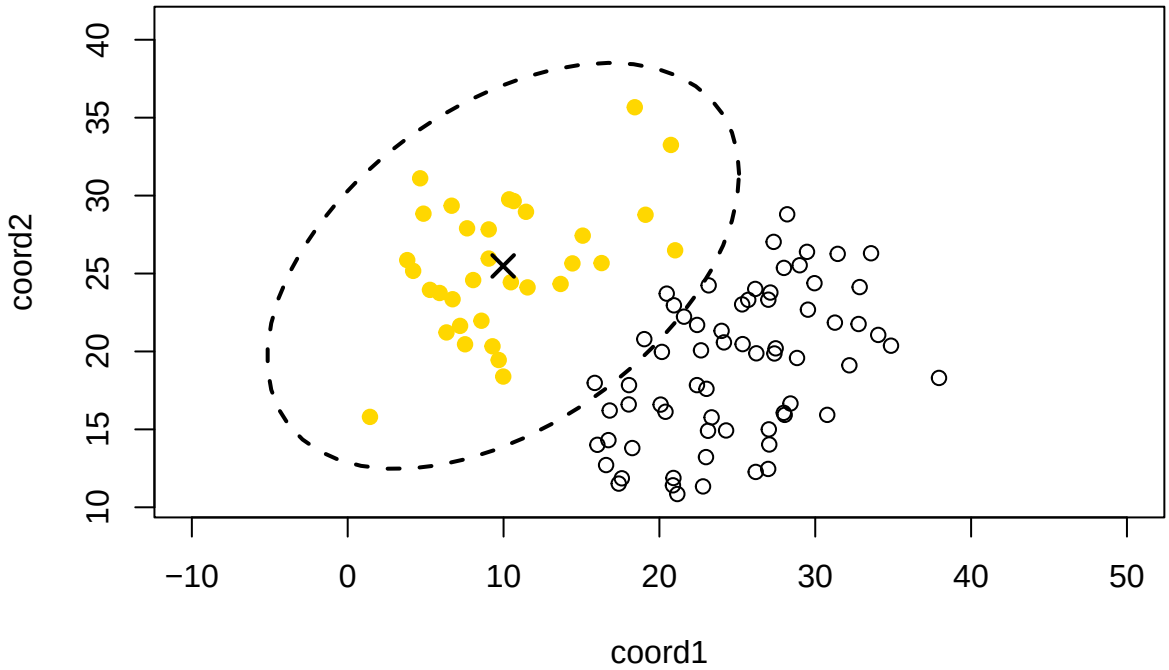
```r
# Center:
M
```

```
##    coord1    coord2
##  9.974545 25.487576
```

```r
# Radius of the ellipse:
r = sqrt(cfr.chisq)
# Length of the semi-axes:
r*sqrt(eigen(S)$values)
```

```
## [1] 17.13797 10.25600
```

```r
plot(p.pr, asp = 1, col='gold', pch=19, xlim=c(-10,50))
points(M[1], M[2], pch = 4, cex = 1.5, lwd = 2)

ellipse(center=M, shape=S, radius=sqrt(cfr.chisq), col = 'black', lty = 2, center.pch = 4)
points(stones[which(cl.ew==1),])
```



8.10 Exam exercise

```
#_____________________________________________________________________________
##### Problem 3 of February 28, 2007
#####-------------------------------

# The dataset Pb3.txt reports Length (cm) Width (cm) of the chest of
# 50 sparrows. The biologist who has collected the measurements aims to
# demonstrate that the sparrows can be divided into two distinct groups
# in terms of length and width of the chest. Help him to prove his theory
# by implementing and commenting on the following analyses:
# (a) By means of an agglomerative hierarchical clustering algorithm that
#     uses the Manhattan distance and the Single linkage state if it is
#     reasonable to cluster the data into two groups.
# (b) Implement a test to prove the difference of the means of the two groups.
# (c) Identify and comment the four Bonferroni intervals with global confidence
#     90% (lower bound, central value, upper bound) for:
#     - The difference of the mean of the variable length.
#     - The difference of the mean of the variable width.
#     - The difference of the mean of the sum of the variables length and width.
#     - The difference of the mean of the difference of variable length and
#       width.
# (d) verify the assumptions necessary to the implementation of the test.

sparrows = read.table('Lab08/Pb3.txt')
head(sparrows)
```

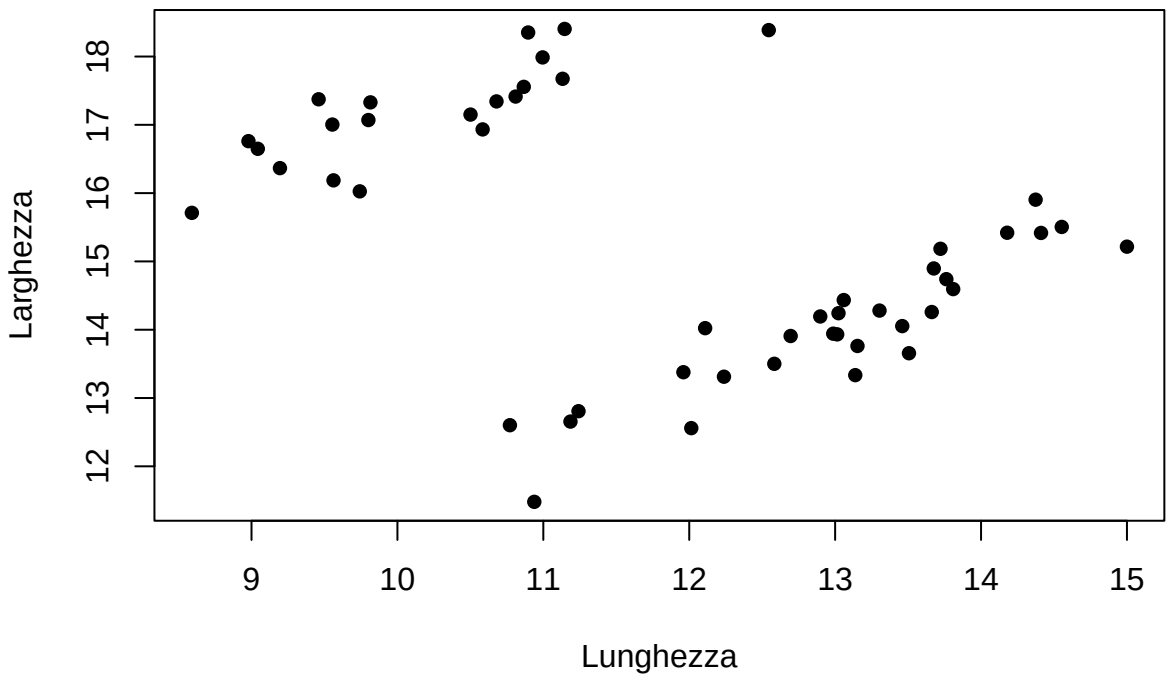
```
##   Lunghezza Larghezza
## 1     9.801    17.070
## 2     8.591    15.711
## 3    14.179    15.420
## 4    13.506    13.656
## 5    13.809    14.595
## 6    12.238    13.312
```

```
dim(sparrows)
```

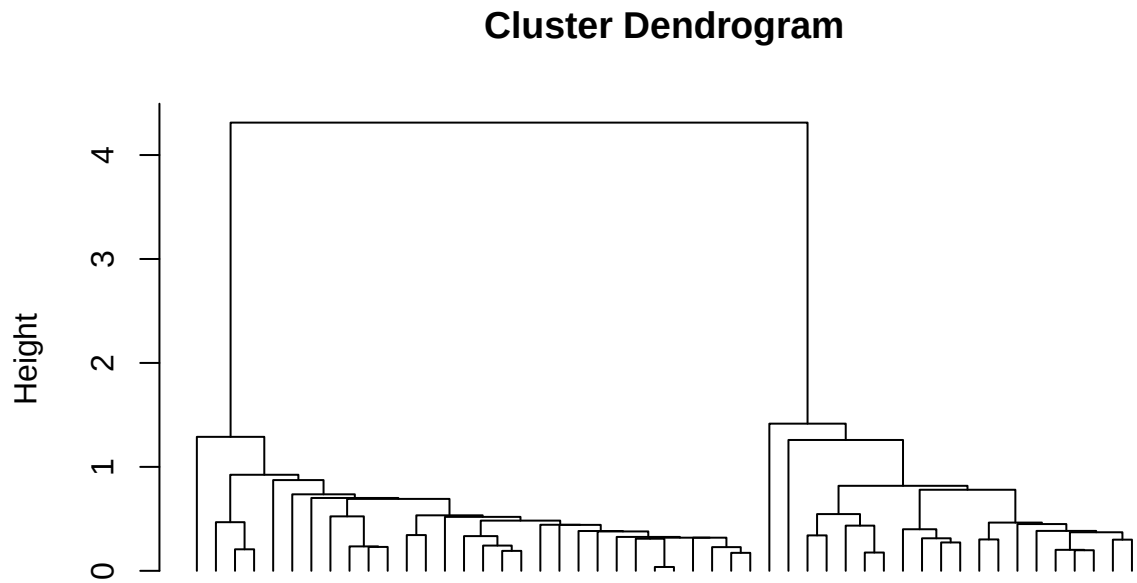
```
## [1] 50  2
```

8.10.1 Qa

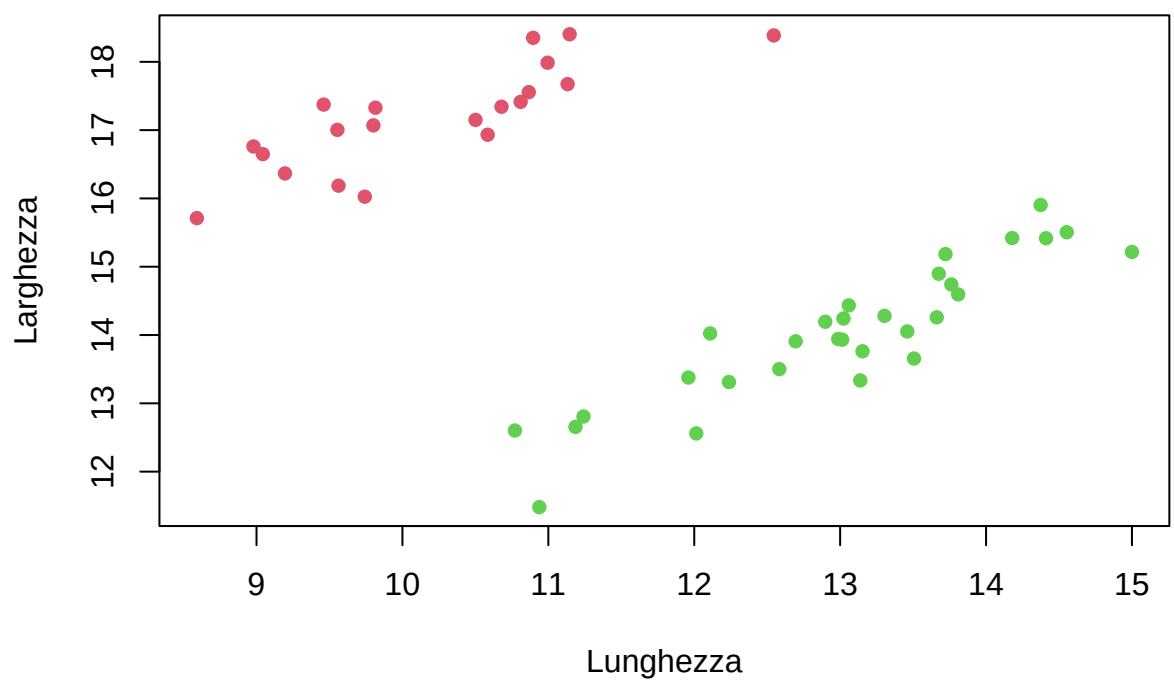
```
### question (a)
# (a) By means of an agglomerative hierarchical clustering algorithm that
#     uses the Manhattan distance and the Single linkage state if it is
#     reasonable to cluster the data into two groups.

plot(sparrows, pch=16)
```




```
gruppi = hclust(dist(sparrows, method='manhattan'), method='single')
plot(gruppi, hang=-0.1, sub='', xlab='', labels=F)
```




```r
# cut in two groups
cluster = cutree(gruppi, k=2)

plot(sparrows, pch=16, col = as.vector(cluster)+1)
```



```r
### question (b)
# (b) Implement a test to prove the difference of the means of the two groups.

g1 = sparrows[cluster==1,]
g2 = sparrows[cluster==2,]
g1
```

```
##    Lunghezza Larghezza
## 1      9.801    17.070
## 2      8.591    15.711
## 7     10.995    17.987
## 12    11.146    18.404
## 15    10.679    17.343
## 16    10.810    17.414
## 19    10.896    18.352
## 25    11.132    17.674
## 26     9.742    16.026
## 28     9.815    17.329
## 30     9.460    17.374
## 31    12.545    18.387
## 35     8.979    16.761
## 37    10.501    17.150
## 41     9.195    16.366
## 43     9.554    17.004
## 44    10.584    16.933
## 47     9.043    16.649
## 48    10.866    17.556
## 50     9.562    16.187
```

```r
g2
```

```
##    Lunghezza Larghezza
## 3     14.179    15.420
## 4     13.506    13.656
## 5     13.809    14.595
## 6     12.238    13.312
## 8     13.722    15.185
## 9     12.109    14.023
## 10    13.138    13.335
## 11    13.304    14.280
## 13    12.898    14.194
## 14    13.153    13.762
## 17    10.771    12.602
## 18    12.583    13.501
## 20    13.023    14.242
## 21    14.374    15.904
## 22    12.014    12.559
## 23    10.938    11.480
## 24    14.553    15.505
## 27    13.676    14.897
## 29    13.460    14.053
## 32    13.662    14.259
## 33    15.000    15.216
## 34    14.411    15.417
## 36    13.762    14.740
## 38    11.186    12.655
## 39    11.960    13.378
## 40    13.059    14.434
## 42    11.241    12.807
## 45    13.014    13.932
## 46    12.987    13.942
## 49    12.695    13.908
```

```r
# Test: H0: mu.1-mu.2==0 vs H1: mu.1-mu.2!=0
p  = 2
n1 = dim(g1)[1]
n2 = dim(g2)[1]
alpha = 0.10

mean1 = sapply(g1,mean)
mean2 = sapply(g2,mean)
cov1  =  cov(g1)
```

```r
cov2  =  cov(g2)
Sp      = ((n1-1)*cov1 + (n2-1)*cov2)/(n1+n2-2)

delta.0 = c(0,0)
Spinv   = solve(Sp)

T2 = n1*n2/(n1+n2) * (mean1-mean2-delta.0) %*% Spinv %*% (mean1-mean2-delta.0)

cfr.fisher = (p*(n1+n2-2)/(n1+n2-1-p))*qf(1-alpha,p,n1+n2-1-p)

pvalue = 1 - pf(T2/(p*(n1+n2-2)/(n1+n2-1-p)), p, n1+n2-1-p)
pvalue
```

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

8.10.2 Qc

```r
### question (c)
# (c) Identify and comment the four Bonferroni intervals with global confidence
#     90% (lower bound, central value, upper bound) for:
#     - The difference of the mean of the variable length.
#     - The difference of the mean of the variable width.
#     - The difference of the mean of the sum of the variables length and width.
#     - The difference of the mean of the difference of variable length and
#       width.

dm = (mean1-mean2)
A  = rbind(c(1,0), c(0,1), c(1,1), c(1,-1))
k  = dim(A)[1]

A.s2 = diag(A%*%Sp%*%t(A))
A.dm = A%*%(mean1-mean2)

Bonf = cbind(inf=A.dm - qt(1-(alpha/(2*k)), n1+n2-2) * sqrt( A.s2*(1/n1+1/n2) ),
             center=A.dm,
             sup=A.dm + qt(1-(alpha/(2*k)), n1+n2-2) * sqrt( A.s2*(1/n1+1/n2) ))
Bonf
```

```
##            [,1]       [,2]      [,3]
## [1,] -3.5160953 -2.8193667 -2.122638
## [2,]  2.5233414  3.1440833  3.764825
## [3,] -0.9524155  0.3247167  1.601849
## [4,] -6.2957694 -5.9634500 -5.631131
```

8.10.3 Qd

```r
### question (d)
# (d) verify the assumptions necessary to the implementation of the test.

mvn(g1)$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.2732724 0.7959886 YES
```

```r
mvn(g2)$multivariateNormality
```

```
##            Test        HZ   p value MVN
## 1 Henze-Zirkler 0.3570629 0.6684499 YES
```

```r
cov1
```

```
##           Lunghezza Larghezza
## Lunghezza 0.9462719 0.6157970
## Larghezza 0.6157970 0.5898402
```

```r
cov2
```

```
##           Lunghezza Larghezza
## Lunghezza  1.180818  1.006807
## Larghezza  1.006807  1.042966
```

8.11 Multidimensional scaling

```
#_____________________________________________________________________________
##### Multidimensional Scaling (principal coordinate analysis)
#####-------------------------------------------------------------

### Given the distances (dissimilarities) among n statistical units, look for
### the k-dimensional representation (k small) of the n statistical units
### such that the distances (dissimilarities) among the representations
### of the n units are as close as possible to the original distances
### (dissimilarities) among the n units.


### Example 1: European cities
###---------------------------

# help(eurodist)
eurodist
# road distances among European cities

# R function for multidimensional scaling: cmdscale
# help(cmdscale)

location = cmdscale(eurodist, k=2)
location

# I have to set asp=1 (equal scales on the two axes)
# to correctly represent Euclidean distances

plot(location[,1], location[,2], type='n', asp=1, axes=FALSE, main="MDS of European cities",
     xlab='',ylab='')
text(location[,1], location[,2], labels=colnames(as.matrix(eurodist)), cex = 0.75, pos = 3)

# change the sign to get the North in the upper part of the plot

plot(location[,1], -location[,2], type='n', asp=1, axes=FALSE, main="MDS of European cities",
     xlab='',ylab='')
text(location[,1], -location[,2], labels=colnames(as.matrix(eurodist)), cex = 0.75, pos = 3)

# compare the original matrix d_ij = d(x_i,x_j) and delta_ij = d(y_i,y_j)

plot(eurodist, dist(location))

# visualize the most different distances

par(cex = 0.75, mar = c(10,10,2,2))
image(1:21, 1:21, asp=1, abs(as.matrix(dist(location)) - as.matrix(eurodist)), axes = F,
      xlab = '', ylab ='')
axis(1, at = 1:21, labels = colnames(as.matrix(eurodist)), las = 2, cex = 0.75)
axis(2, at = 1:21, labels = colnames(as.matrix(eurodist)), las = 1, cex = 0.75)
box()

# Rome-Athens
as.matrix(eurodist)[19,1]
as.matrix(dist(location))[19,1]

# Cologne-Geneve
as.matrix(eurodist)[6,8]
as.matrix(dist(location))[6,8]


# Compute the "stress": the higher it is, the worse
# the matching between original distances and their
# geometrical representation through MDS
Stressk = NULL
for(k in 1:4)
{
  location.k = cmdscale(eurodist, k)
  Stress = (sum( (as.vector(eurodist) - as.vector(dist(location.k)))^2)  /
              sum( as.vector(location.k)^2))^(1/2)
  Stressk = c(Stressk, Stress)
}
```

```r
plot(1:4,Stressk,xlab='k',ylab='Stress',lwd=2)
# the stress increases for k>2 because of numerical problems
# (the representation of minimal stress is not always found)
```

9 Lab 09 - Linear models

```r
###---------------------###
### LAB 9 (11/05/2022)  ###
###---------------------###

### TOPICS:
### Linear models

library(MASS)
library(car)
library(rgl)

##### Linear models
#####---------------
```

9.1 Multiple linear regression, on dataset cars

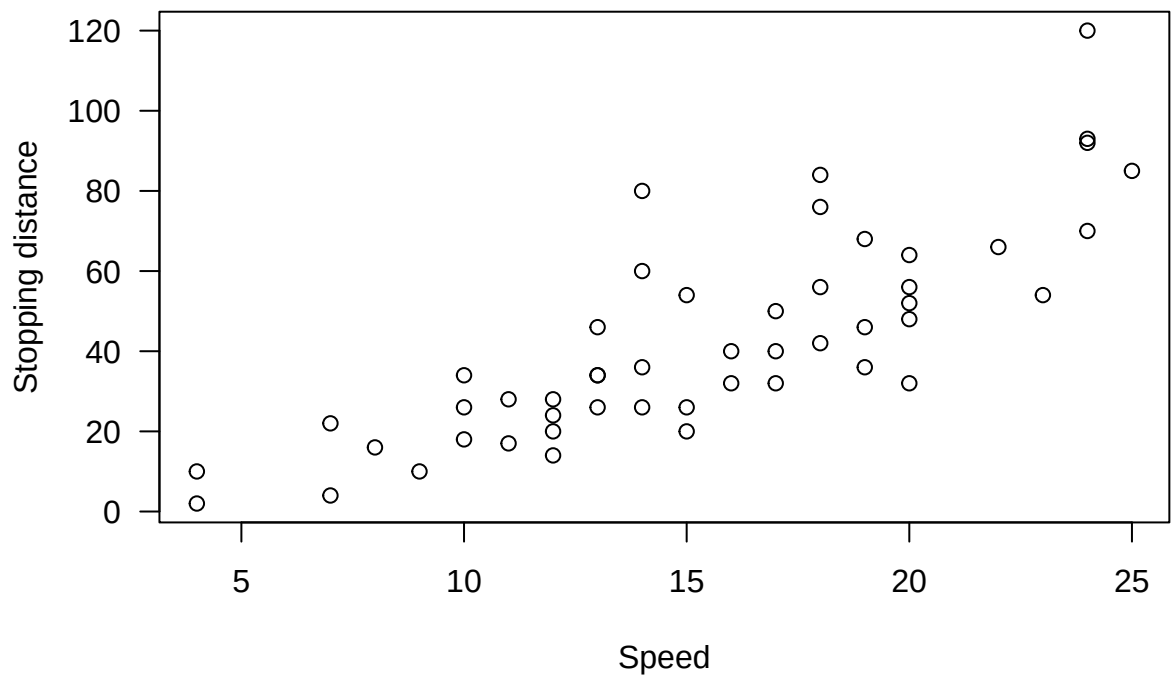
```r
### Example 1: Multiple linear regression
###----------------------------------------
### Dataset cars: distance taken to stop [ft] as a function of velocity [mph]
### for some cars in the 1920s

# help(cars)

head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```r
dim(cars)
```

```
## [1] 50  2
```

```r
plot(cars, xlab='Speed', ylab='Stopping distance', las=1)
```




```
#! The distance taken to stop can be explained (proportional to) speed^2
#! like from some physics formula. So we should introduce speed^2 to our model!
#! Really do transformations, dont use the original given data

n          = dim(cars)[[1]]
distance   = cars$dist
#! Design matrix Z: n*3
#! / 1   |     |     \
#! | 1 speed speed^2 |
#! \ 1   |     |     /
speed1     = cars$speed
speed2     = cars$speed^2

### Model:
### distance = beta_0 + beta_1 * speed + beta_2 * speed^2 + Eps
### (linear in the parameters!)

### Assumptions:
## 1) Parameter estimation: E(Eps) = 0  and  Var(Eps) = sigma^2
## 2) Inference:            Eps ~ N(0, sigma^2)

#! The gaussian assumption is in fact just needed when working on CIs,
#! ecc, so on the inference partt, not the building of the model
```

9.1.1 Estimate of the parameters

```
##### 1) Estimate of the parameters
##### Assumptions: E(Eps) = 0  and  Var(Eps) = sigma^2
#####-------------------------------------------------

# help(lm)
fm = lm(distance ~ speed1 + speed2)
```

```
summary(fm)
```

9.1.1.1 Lm summary explanation

```
##
## Call:
## lm(formula = distance ~ speed1 + speed2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.720  -9.184  -3.188   4.628  45.152
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47014   14.81716   0.167    0.868
## speed1       0.91329    2.03422   0.449    0.656
## speed2       0.09996    0.06597   1.515    0.136
##
## Residual standard error: 15.18 on 47 degrees of freedom
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12
```

```
# Note: colinearity
#
#! In the Coefficients section the std errors are already provided under the
#! assumption of gaussianity. And still under that there are the tests about the
#! significance of the coefficients beta_i, / H0: beta_i is useful
#!                                          \ H1: beta_i is useless
#! But are one-at-a-time effect, fixing just one, but having collinearity or
#! correlation does not make possible such a "fix"
#
#! Finally the F-statistic is for the test on the significance of all parameters
#! beta_i, for / H0: all variables are useless
#!             \ H1: at least one is useful, ie the model has sense
#! So we are glad to see a very low pvalue, we reject H0
#
#! R^2 tells how much percent of variability of Y we are able to explain with our model

fitted(fm)        # y hat
```

```
##         1         2         3         4         5         6         7         8
##  7.722637  7.722637 13.761157 13.761157 16.173834 18.786430 21.598944 21.598944
##         9        10        11        12        13        14        15        16
## 21.598944 24.611377 24.611377 27.823729 27.823729 27.823729 27.823729 31.235999
##        17        18        19        20        21        22        23        24
## 31.235999 31.235999 31.235999 34.848188 34.848188 34.848188 34.848188 38.660295
##        25        26        27        28        29        30        31        32
## 38.660295 38.660295 42.672321 42.672321 46.884266 46.884266 46.884266 51.296129
##        33        34        35        36        37        38        39        40
## 51.296129 51.296129 51.296129 55.907911 55.907911 55.907911 60.719611 60.719611
##        41        42        43        44        45        46        47        48
## 60.719611 60.719611 60.719611 70.942768 76.354224 81.965599 81.965599 81.965599
##        49        50
## 81.965599 87.776892
```

```
residuals(fm)     # eps hat
```

```
##           1           2           3           4           5           6
##  -5.7226371   2.2773629  -9.7611569   8.2388431  -0.1738340  -8.7864298
##           7           8           9          10          11          12
##  -3.5989441   4.4010559  12.4010559  -7.6113771   3.3886229 -13.8237287
##          13          14          15          16          17          18
##  -7.8237287  -3.8237287   0.1762713  -5.2359988   2.7640012   2.7640012
##          19          20          21          22          23          24
##  14.7640012  -8.8481876   1.1518124  25.1518124  45.1518124 -18.6602950
##          25          26          27          28          29          30
## -12.6602950  15.3397050 -10.6723209  -2.6723209 -14.8842655  -6.8842655
##          31          32          33          34          35          36
##   3.1157345  -9.2961287   4.7038713  24.7038713  32.7038713 -19.9079105
##          37          38          39          40          41          42
##  -9.9079105  12.0920895 -28.7196109 -12.7196109  -8.7196109  -4.7196109
##          43          44          45          46          47          48
##   3.2803891  -4.9427675 -22.3542237 -11.9655985  10.0344015  11.0344015
##          49          50
##  38.0344015  -2.7768919
```

```r
coefficients(fm)  # beta_i
```

```
## (Intercept)      speed1      speed2
##   2.4701378   0.9132876   0.0999593
```

```r
vcov(fm)          # cov(beta_i)
```

```
##             (Intercept)      speed1       speed2
## (Intercept) 219.5483705 -28.9523122  0.872858710
## speed1      -28.9523122   4.1380528 -0.131439753
## speed2        0.8728587  -0.1314398  0.004351805
```

```r
fm$rank # order of the model [r+1]
```

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

```r
fm$df   # degrees of freedom of the residuals [n-(r+1)]
```

```
## [1] 47
```

```r
#! 50-(2+1)

hatvalues(fm) # h_ii (or sometimes called "leverage")
```

```
##          1          2          3          4          5          6          7
## 0.28812937 0.28812937 0.09900328 0.09900328 0.06991940 0.05169124 0.04132501
##          8          9         10         11         12         13         14
## 0.04132501 0.04132501 0.03628040 0.03628040 0.03447059 0.03447059 0.03447059
##         15         16         17         18         19         20         21
## 0.03447059 0.03426225 0.03426225 0.03426225 0.03426225 0.03447551 0.03447551
##         22         23         24         25         26         27         28
## 0.03447551 0.03447551 0.03438403 0.03438403 0.03438403 0.03371490 0.03371490
##         29         30         31         32         33         34         35
## 0.03264873 0.03264873 0.03264873 0.03181959 0.03181959 0.03181959 0.03181959
##         36         37         38         39         40         41         42
## 0.03231507 0.03231507 0.03231507 0.03567621 0.03567621 0.03567621 0.03567621
##         43         44         45         46         47         48         49
## 0.03567621 0.05942708 0.08516635 0.12447031 0.12447031 0.12447031 0.12447031
##         50
## 0.18114746
```

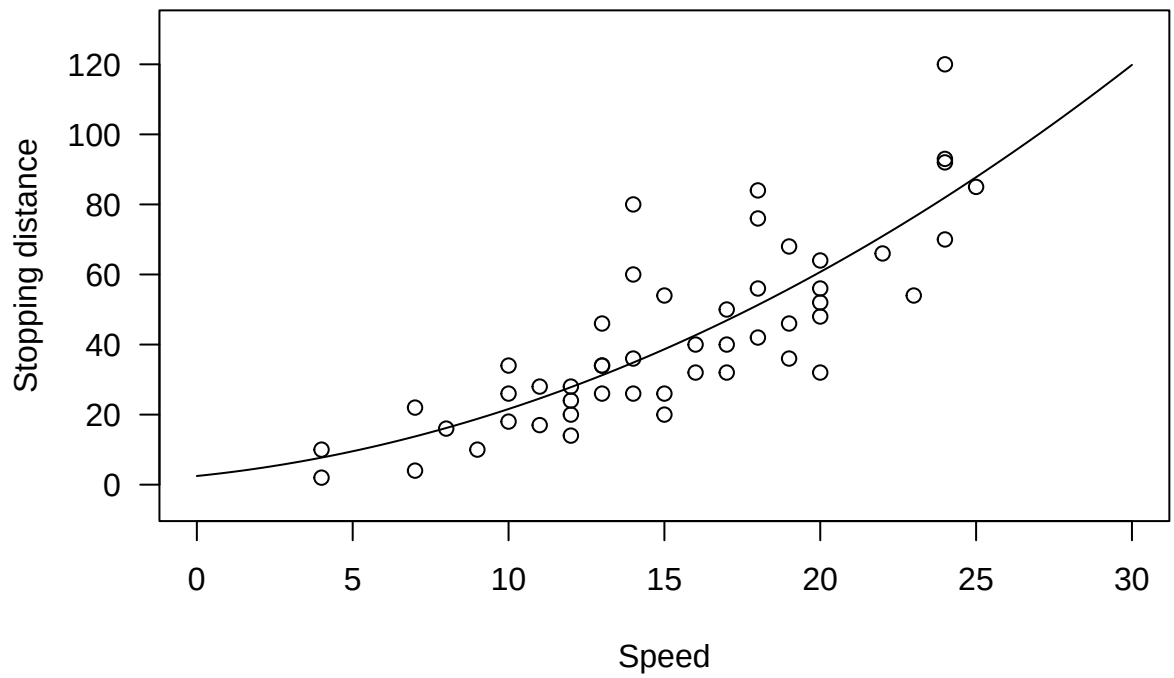
```r
# They quantify:
# 1) How far is the i-th observation from the other ones in the features space
#!   (like a Mahlanobis distance between unit i and the baricenter of the data cloud)
# 2) The influence of the i-th observation on the fit
#    (can be seen as the derivative dyhat_i / dy_i)

rstandard(fm)               # standardized residuals: eps_j / sqrt(s^2*(1-h_ii))
```

```
##           1           2           3           4           5           6
## -0.44692674  0.17785758 -0.67761087  0.57193320 -0.01187723 -0.59453611
##           7           8           9          10          11          12
## -0.24220331  0.29618418  0.83457167 -0.51089137  0.22745138 -0.92700723
##          13          14          15          16          17          18
## -0.52465245 -0.25641592  0.01182060 -0.35108365  0.18533152  0.18533152
##          19          20          21          22          23          24
##  0.98995429 -0.59335328  0.07723974  1.68666299  3.02784903 -1.25128708
##          25          26          27          28          29          30
## -0.84895032  1.02862118 -0.71539666 -0.17913343 -0.99718556 -0.46121793
##          31          32          33          34          35          36
##  0.20874160 -0.62253627  0.31500537  1.65435058  2.19008866 -1.33351949
##          37          38          39          40          41          42
## -0.66367546  0.80998139 -1.92711568 -0.85349909 -0.58509494 -0.31669079
##          43          44          45          46          47          48
##  0.22011751 -0.33582624 -1.54003007 -0.84263533  0.70663755  0.77705904
##          49          50
##  2.67843940 -0.20220729
```

```r
sum(residuals(fm)^2)/fm$df  # s^2 estimate of sigma^2
```

```
## [1] 230.3131
```

```r
plot(cars, xlab='Speed', ylab='Stopping distance', las=1, xlim=c(0,30), ylim=c(-5,130))
x = seq(0,30,by=0.1)
b = coef(fm)
#! We plot the obtained regression result, and here in the "original" space may
#! not look like a line if we did some non-linear (like speed^2) transformation
lines(x, b[1]+b[2]*x+b[3]*x^2)
```




```r
# points(cars$speed,fitted(fm))
```

9.1.2 Inference on the parameters

```r
##### Inference on the parameters
##### Assumption: Eps ~ N(0, sigma^2)
#####-----------------------------------------
### Test (Fisher):
# H0: (beta1, beta2) == (0, 0) vs H1: (beta1, beta2) != (0, 0)
linearHypothesis(fm, rbind(c(0,1,0), c(0,0,1)), c(0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## speed1 = 0
## speed2 = 0
##
## Model 1: restricted model
## Model 2: distance ~ speed1 + speed2
##
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
## 1     49 32539
## 2     47 10825  2     21714 47.141 5.852e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#! We got the same F-statistic of the original summary(fm)
summary(fm)
```

```
##
## Call:
## lm(formula = distance ~ speed1 + speed2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.720  -9.184  -3.188   4.628  45.152
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47014   14.81716   0.167    0.868
## speed1       0.91329    2.03422   0.449    0.656
## speed2       0.09996    0.06597   1.515    0.136
##
## Residual standard error: 15.18 on 47 degrees of freedom
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12
```

```r
p = 2  # number of tested coefficients
r = 2  # number of regressors
```

```r
# Confidence region: an ellipse
#! centered at the point estimate:
c(coefficients(fm)[2], coefficients(fm)[3])
```

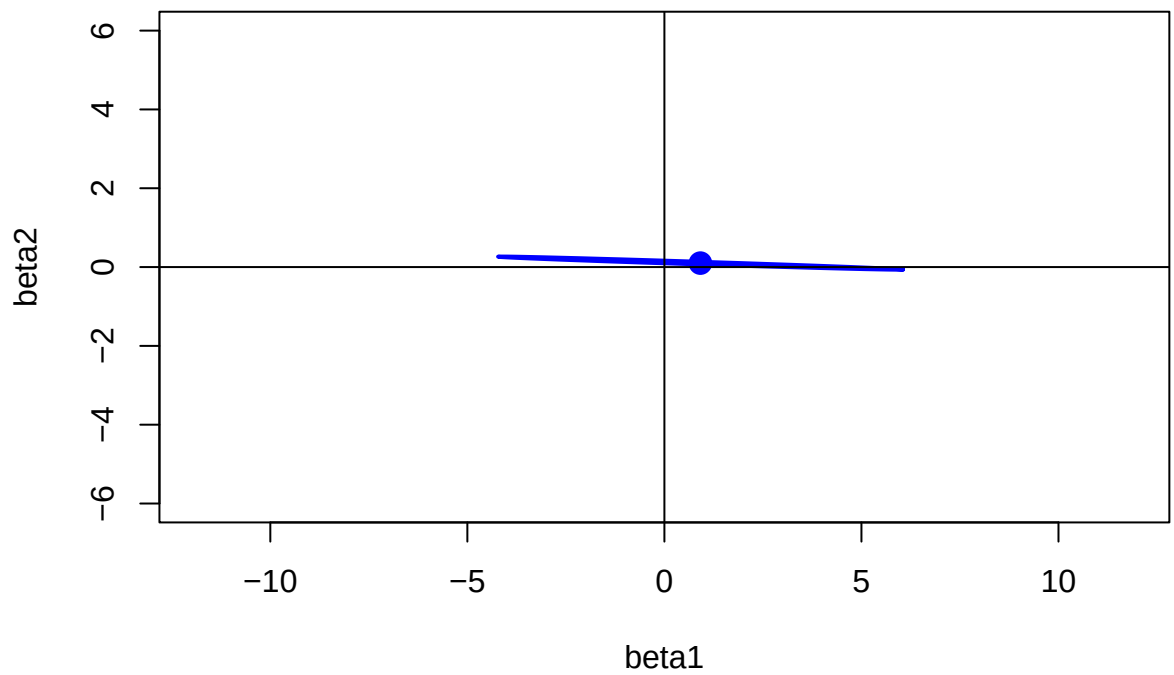
9.1.2.1 Confidence region

```
##    speed1    speed2
## 0.9132876 0.0999593
```

```r
#! those are beta1 and beta2
# with direction of the axes:
eigen(vcov(fm)[2:3,2:3])$vectors
```

```
##             [,1]        [,2]
## [1,] -0.99949587 -0.03174901
## [2,]  0.03174901 -0.99949587
```

```r
plot(coefficients(fm)[2], coefficients(fm)[3], xlim = c(-6,6), ylim = c(-6,6),
     asp=1, xlab='beta1', ylab='beta2')
ellipse(coefficients(fm)[2:3], vcov(fm)[2:3,2:3], sqrt(p*qf(1-0.05,p,n-(r+1))))
abline(v=0)
abline(h=0)
```




```
#! Note: colinearity! it causes that toghether the betas are not zero, but
#! singularly maybe yes
#
#! The zero point (0,0) is outside the confidence infedence
#! so we reject that beta1 and beta2 are, as a vector, equal to (0,0)
#
#! But if we project on the axis:
#! - for beta1 it will include zero, we wont exclude the fact that it may be zero
#! - for beta2 it will also include zero, so we wont exclude the fact that it may be zero
```

```
# Bonferroni intervals (level 95%)
Bf = rbind(
  beta1=c(coefficients(fm)[2]-sqrt(vcov(fm)[2,2])*qt(1-0.05/(2*p), n-(r+1)),
          coefficients(fm)[2]+sqrt(vcov(fm)[2,2])*qt(1-0.05/(2*p), n-(r+1))),
  beta2=c(coefficients(fm)[3]-sqrt(vcov(fm)[3,3])*qt(1-0.05/(2*p), n-(r+1)),
          coefficients(fm)[3]+sqrt(vcov(fm)[3,3])*qt(1-0.05/(2*p), n-(r+1)))
)
Bf
```

9.1.2.2 Bonferroni CIs

```
##            speed1   speed1
## beta1 -3.79692941 5.623505
## beta2 -0.05278943 0.252708
```

```
# or (only for intervals on beta)
confint(fm, level= 1-0.05/p)[2:3,] # Bonferroni correction!
```

```
##             1.25 %  98.75 %
## speed1 -3.79692941 5.623505
## speed2 -0.05278943 0.252708
```

```
# Note: confint() returns the confidence intervals one-at-a-time;
# to have a global level 95% we need to include a correction
```

9.1.3 Test on lin comb of the beta_i

```
### Test:
# H0: (beta0+beta2, beta1) == (0,0) vs H1: (beta0+beta2, beta1) != (0,0)
#! How to test different lin comb of the betas
C = rbind(c(1,0,1), c(0,1,0))
linearHypothesis(fm, C, c(0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## (Intercept)  + speed2 = 0
## speed1 = 0
##
## Model 1: restricted model
## Model 2: distance ~ speed1 + speed2
##
##   Res.Df   RSS Df Sum of Sq      F  Pr(>F)
## 1     49 11973
## 2     47 10825  2    1148.4 2.4932 0.09352 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Homework
# Build the associated confidence region
```

9.1.4 CIs and PIs on a new obs

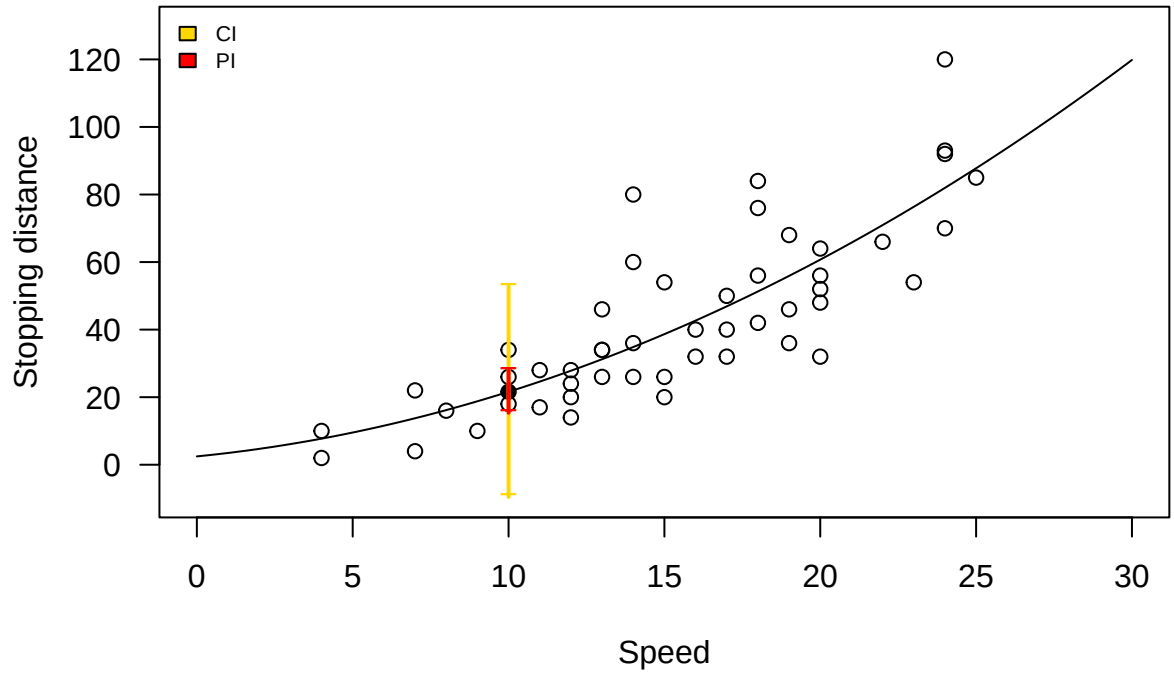
```
##### Confidence intervals for the mean & prediction intervals (new obs)
##### Assumption: Eps ~ N(0, sigma^2)
#####--------------------------------
# Command predict()

Z0.new = data.frame(speed1=10, speed2=10^2)
#! so a new unit at speed 10

# Conf. int. for the mean
Conf = predict(fm, Z0.new, interval='confidence', level=1-0.05)
Conf
```

```
##        fit      lwr      upr
## 1 21.59894 15.39257 27.80532
```

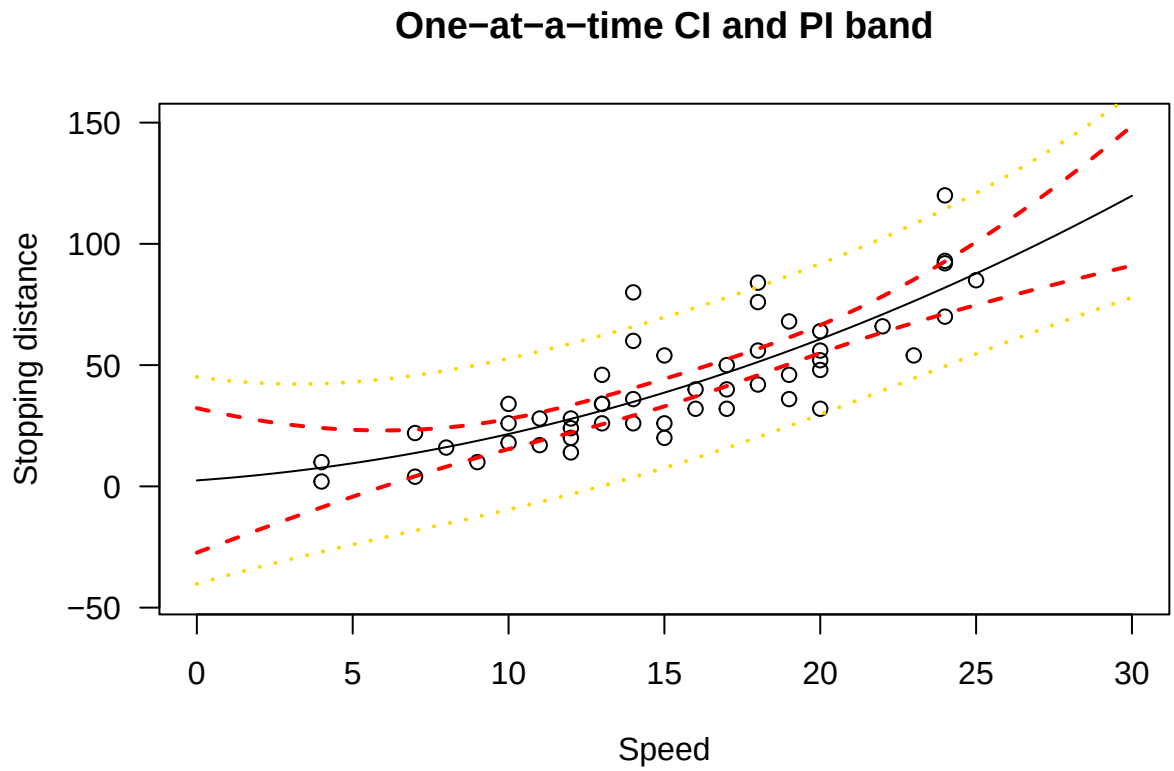
```
# Pred. int. for a new obs
Pred = predict(fm, Z0.new, interval='prediction', level=1-0.05)
Pred
```

```
##        fit       lwr      upr
## 1 21.59894 -9.555818 52.75371
```

```
plot(cars, xlab='Speed', ylab='Stopping distance', las=1, xlim=c(0,30), ylim=c(-10,130))
x = seq(0,30,by=0.1)
b = coef(fm)
lines(x, b[1]+b[2]*x+b[3]*x^2)
points(10,Conf[1], pch=19)
segments(10,Pred[2],10,Pred[3],col='gold', lwd=2)
segments(10,Conf[2],10,Conf[3],col='red', lwd=2)
points(10,Conf[2], pch='-', col='red', lwd=2)
points(10,Conf[3], pch='-', col='red', lwd=2)
points(10,Pred[2], pch='-', col='gold', lwd=2)
points(10,Pred[3], pch='-', col='gold', lwd=2)
legend("topleft",c("CI","PI"),fill=c("gold","red"), bty='n', cex = 0.7)
```

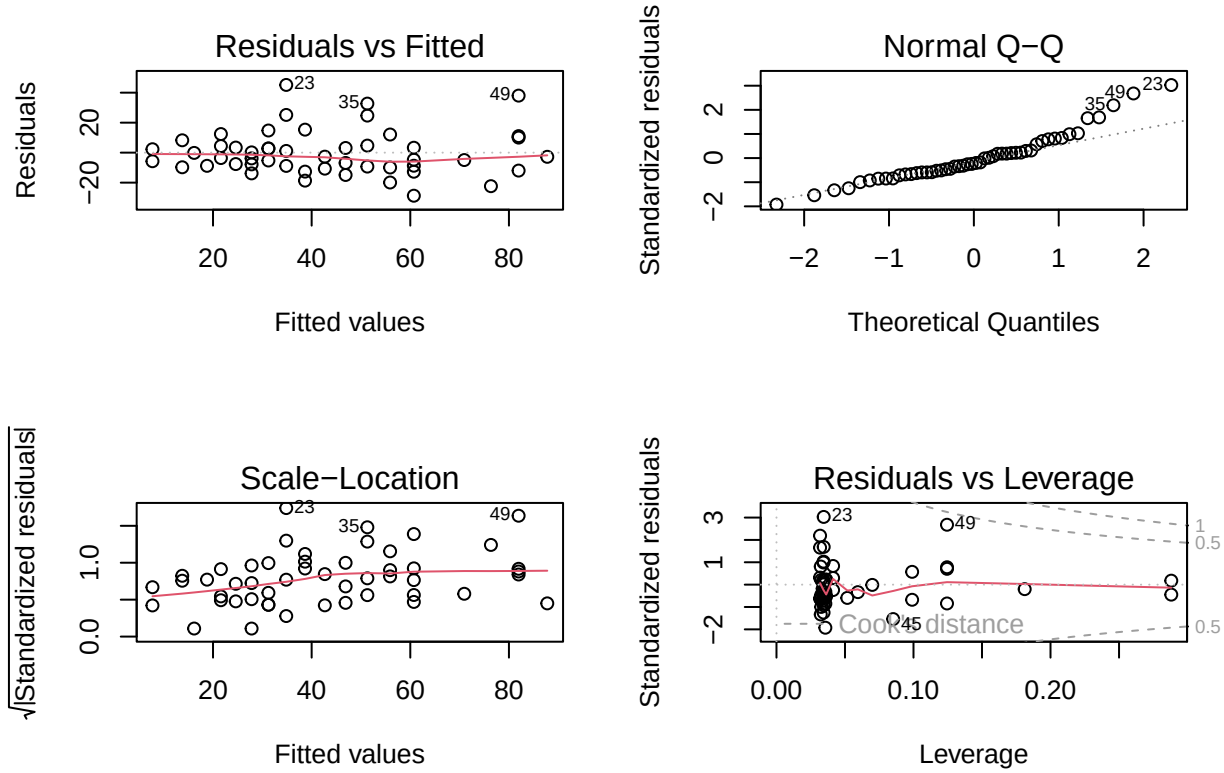



```
# We can repeat these for values of speed between 0 and 30
# (point-wise intervals!)
Z0   = data.frame(cbind(speed1=seq(0, 30, length=100),
                        speed2=seq(0, 30, length=100)^2))
Conf = predict(fm, Z0, interval='confidence')
Pred = predict(fm, Z0, interval='prediction')

plot(cars, xlab='Speed', ylab='Stopping distance', las=1, xlim=c(0,30), ylim=c(-45,150),
     main="One-at-a-time CI and PI band")
lines(Z0[,1], Conf[,'fit'])
lines(Z0[,1], Conf[,'lwr'], lty=2, col='red', lwd=2)
lines(Z0[,1], Conf[,'upr'], lty=2, col='red', lwd=2)
lines(Z0[,1], Pred[,'lwr'], lty=3, col='gold', lwd=2)
lines(Z0[,1], Pred[,'upr'], lty=3, col='gold', lwd=2)
```



9.1.5 Verify assumptions

```
##### Verify assumptions
#####--------------------

par(mfrow=c(2,2))
plot(fm)
```




```
#! NW: we should observe no specific pattern. Otherwise it means we have some relationship
#! between our residuals and fitted values (when they should be independent, geom orthogonal).
#! And vertically it should seem normally distributed. Red line is like a moving average
#
#! NE: qq-plot, we can understand if we may have outliers, like high residuals, but outliers
#! in terms of residuals, not space-cloude-units. Leverages instead tell that
#
#! SW: not really useful, is like the NW but standardized
#
#! SE: it allows us to see which obs have an high leverage, which obs influence a lot
#! our mode. If we have units outside 1-Cook-distance-region we may remove them. Ie it
#! summarizes problems of influence and eteroschedasticity

shapiro.test(residuals(fm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fm)
## W = 0.93419, p-value = 0.007988
```

```
#! we reject normality (pvalue low, reject H0) as shown in qq-plot

### What happens if we change unit of measure?
### Exercise: Compare the results obtained from:
distance.m  = cars$dist*0.3          # feet -> m
speed1.kmh  = cars$speed*1.6         # miles/hour -> km per hour
speed2.kmh2 = cars$speed^2 * 1.6^2
### Compare:
### - Coefficient estimates and corresponding covariance matrices
### - confidence intervals on the coefficients
### - fitted values
### - residuals
### - results of the tests
###   H0: (beta1, beta2) == (0,0) vs H1: (beta1, beta2) != (0,0)
###   H0: (beta0+beta2, beta1) == (0,0) vs H1: (beta0+beta2, beta1) != (0,0)
```

9.2 Importance of diagnostic of residuals

```r
### Example 2: Anscombe
###--------------------------------
#! Example about importance of diagnostic of residuals

### This example shows some cases for which examining data and
### residuals is crucial (don't look at R^2 only!)

anscombe
```

```
##    x1 x2 x3 x4    y1   y2    y3    y4
## 1  10 10 10  8  8.04 9.14  7.46  6.58
## 2   8  8  8  8  6.95 8.14  6.77  5.76
## 3  13 13 13  8  7.58 8.74 12.74  7.71
## 4   9  9  9  8  8.81 8.77  7.11  8.84
## 5  11 11 11  8  8.33 9.26  7.81  8.47
## 6  14 14 14  8  9.96 8.10  8.84  7.04
## 7   6  6  6  8  7.24 6.13  6.08  5.25
## 8   4  4  4 19  4.26 3.10  5.39 12.50
## 9  12 12 12  8 10.84 9.13  8.15  5.56
## 10  7  7  7  8  4.82 7.26  6.42  7.91
## 11  5  5  5  8  5.68 4.74  5.73  6.89
```

```r
# Four x-y datasets which have the same traditional statistical
# properties (mean, variance, correlation, regression line, etc.),
# yet are quite different.

attach(anscombe)

# dataset 1
lm1 = lm(y1~ x1)
summary(lm1)
```

```
##
## Call:
## lm(formula = y1 ~ x1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.92127 -0.45577 -0.04136  0.70941  1.83882
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0001     1.1247   2.667  0.02573 *
## x1            0.5001     0.1179   4.241  0.00217 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.237 on 9 degrees of freedom
## Multiple R-squared:  0.6665, Adjusted R-squared:  0.6295
## F-statistic: 17.99 on 1 and 9 DF,  p-value: 0.00217
```

```r
# dataset 2
lm2 = lm(y2~ x2)
summary(lm2)
```

```
##
## Call:
## lm(formula = y2 ~ x2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9009 -0.7609  0.1291  0.9491  1.2691
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.001      1.125   2.667  0.02576 *
## x2             0.500      0.118   4.239  0.00218 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1.237 on 9 degrees of freedom
## Multiple R-squared:  0.6662, Adjusted R-squared:  0.6292
## F-statistic: 17.97 on 1 and 9 DF,  p-value: 0.002179
```

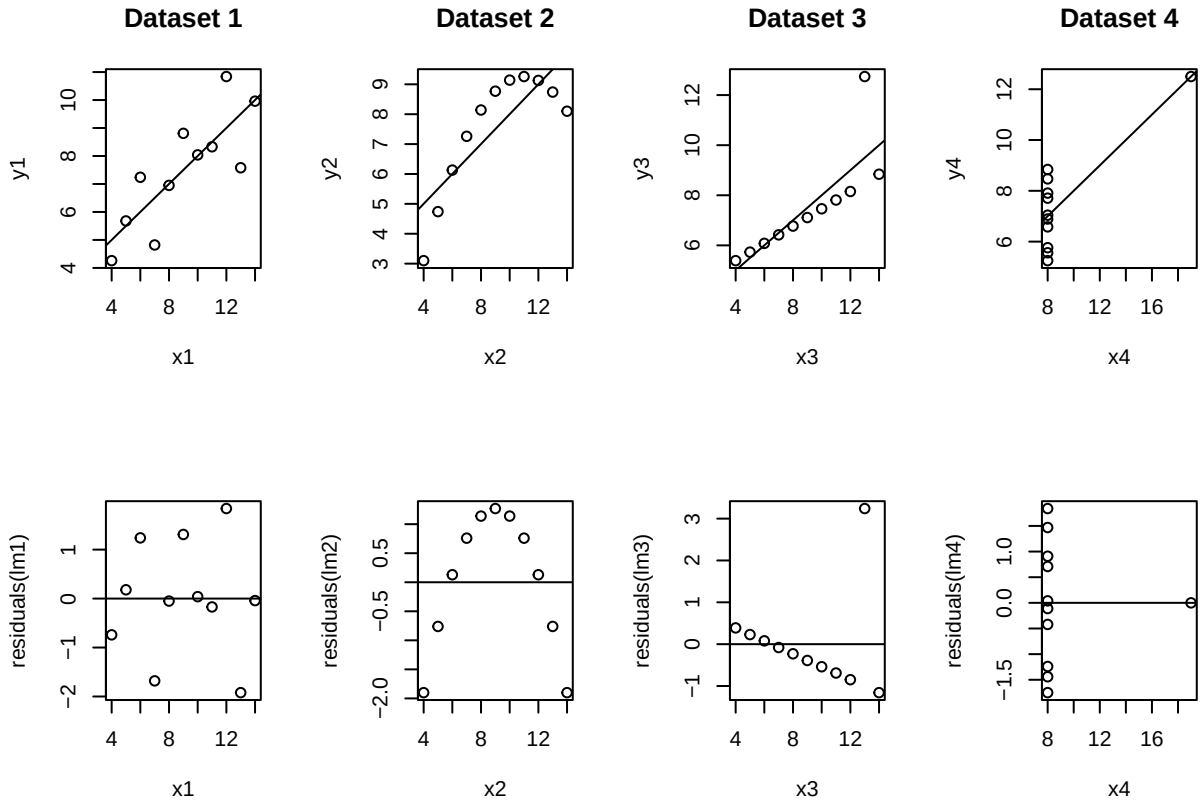
```r
# dataset 3
lm3 = lm(y3~ x3)
summary(lm3)
```

```
##
## Call:
## lm(formula = y3 ~ x3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1586 -0.6146 -0.2303  0.1540  3.2411
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0025     1.1245   2.670  0.02562 *
## x3            0.4997     0.1179   4.239  0.00218 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.236 on 9 degrees of freedom
## Multiple R-squared:  0.6663, Adjusted R-squared:  0.6292
## F-statistic: 17.97 on 1 and 9 DF,  p-value: 0.002176
```

```r
# dataset 4
lm4 = lm(y4~ x4)
summary(lm4)
```

```
##
## Call:
## lm(formula = y4 ~ x4)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.751 -0.831  0.000  0.809  1.839
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0017     1.1239   2.671  0.02559 *
## x4            0.4999     0.1178   4.243  0.00216 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.236 on 9 degrees of freedom
## Multiple R-squared:  0.6667, Adjusted R-squared:  0.6297
## F-statistic:    18 on 1 and 9 DF,  p-value: 0.002165
```

```r
# same R^2, same coefficient estimate, same residual std error
#! Can we consider that we have equally good fits in all cases? NO!

par(mfcol=c(2,4))

plot(x1,y1, main='Dataset 1')
abline(lm1)
plot(x1,residuals(lm1))
abline(h=0)

plot(x2,y2, main='Dataset 2')
abline(lm2)
plot(x2,residuals(lm2))
abline(h=0)

plot(x3,y3, main='Dataset 3')
abline(lm3)
plot(x3,residuals(lm3))
abline(h=0)

plot(x4,y4, main='Dataset 4')
abline(lm4)
plot(x4,residuals(lm4))
abline(h=0)
```




```
#! df1: all seems actually good
#! df2: we see a pattern, the parabola in residuals plot. So we should add a
#!      quadratic regressor (x2^2)
#! df3: we have an outlier, a leverage point, foundable from Cook distance (the
#!      leverage may not be that high, is still "close" to the baricenter) and
#!      from the residual plot also, we see that straight line
#! df4: now we have a leverage, a point very far away from the cloud of points.
#!      The best model there could just be the intercept, the mean

#! Moral: do diagnostic of residuals! dont stop at the summary

detach(anscombe)
```

9.3 Tuning the model, on dataset brain wheight

```
##### Example 3: brain_weight
#####-------------------------------------------------------

data = read.table('Lab09/brain_weight.txt', header=T)

head(data)
```

```
##                            body brain
## Lesser short-tailed shrew 0.005  0.14
## Little brown bat          0.010  0.25
## Big brown bat             0.023  0.30
## Mouse                     0.023  0.40
## Musk shrew                0.048  0.33
## Star-nosed mole           0.060  1.00
```

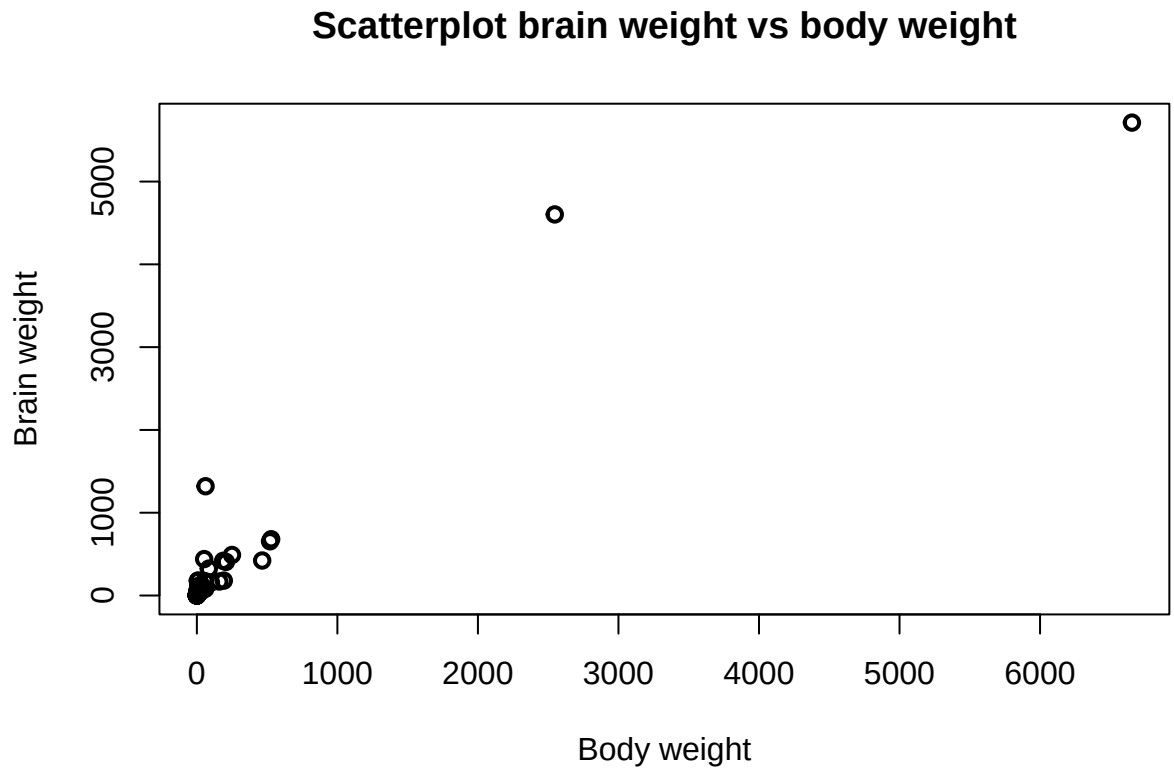
```
dim(data)
```

```
## [1] 62  2
```

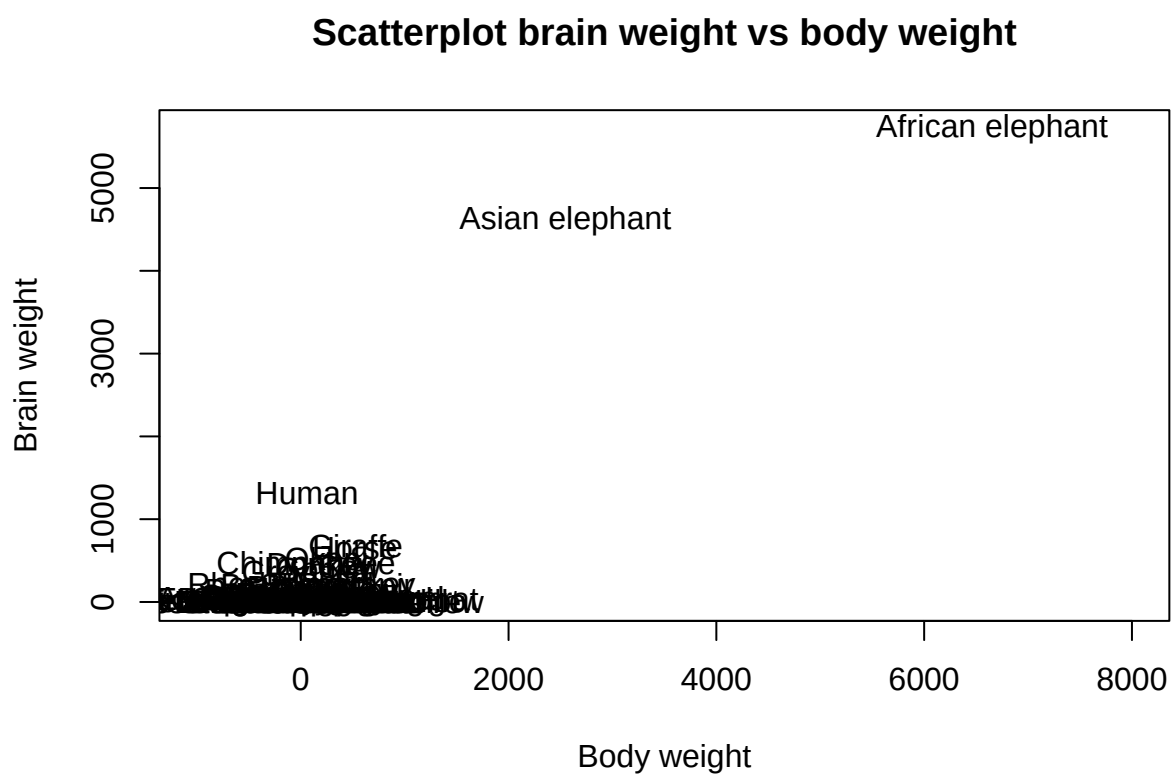
```
dimnames(data)
```

```
## [[1]]
##  [1] "Lesser short-tailed shrew" "Little brown bat"
##  [3] "Big brown bat"             "Mouse"
##  [5] "Musk shrew"                "Star-nosed mole"
##  [7] "E. American mole"          "Ground squirrel"
##  [9] "Tree shrew"                "Golden hamster"
## [11] "Mole"                      "Galago"
## [13] "Rat"                       "Chinchilla"
## [15] "Owl monkey"                "Desert hedgehog"
## [17] "Rock hyrax-a"              "European hedgehog"
## [19] "Tenrec"                    "Artic ground squirrel"
## [21] "African giant pouched rat" "Guinea pig"
## [23] "Mountain beaver"           "Slow loris"
## [25] "Genet"                     "Phalanger"
## [27] "N.A. opossum"              "Tree hyrax"
## [29] "Rabbit"                    "Echidna"
## [31] "Cat"                       "Artic fox"
## [33] "Water opossum"             "Nine-banded armadillo"
## [35] "Rock hyrax-b"              "Yellow-bellied marmot"
## [37] "Verbet"                    "Red fox"
## [39] "Raccoon"                   "Rhesus monkey"
## [41] "Potar monkey"              "Baboon"
## [43] "Roe deer"                  "Goat"
## [45] "Kangaroo"                  "Grey wolf"
## [47] "Chimpanzee"                "Sheep"
## [49] "Giant armadillo"           "Human"
## [51] "Grey seal"                 "Jaguar"
## [53] "Brazilian tapir"           "Donkey"
## [55] "Pig"                       "Gorilla"
## [57] "Okapi"                     "Cow"
## [59] "Horse"                     "Giraffe"
## [61] "Asian elephant"            "African elephant"
##
## [[2]]
## [1] "body"  "brain"
```

```
attach(data)
X = body
Y = brain
detach(data)


plot(X, Y, main='Scatterplot brain weight vs body weight', lwd=2,
     xlab='Body weight', ylab='Brain weight')
```




```
#! We look that we have problems about higher values

plot(X, Y, main='Scatterplot brain weight vs body weight', lwd=2,
     xlab='Body weight', ylab='Brain weight',col='white',xlim=c(-1000,8000))
text(X, Y,dimnames(data)[[1]],cex=1)
```

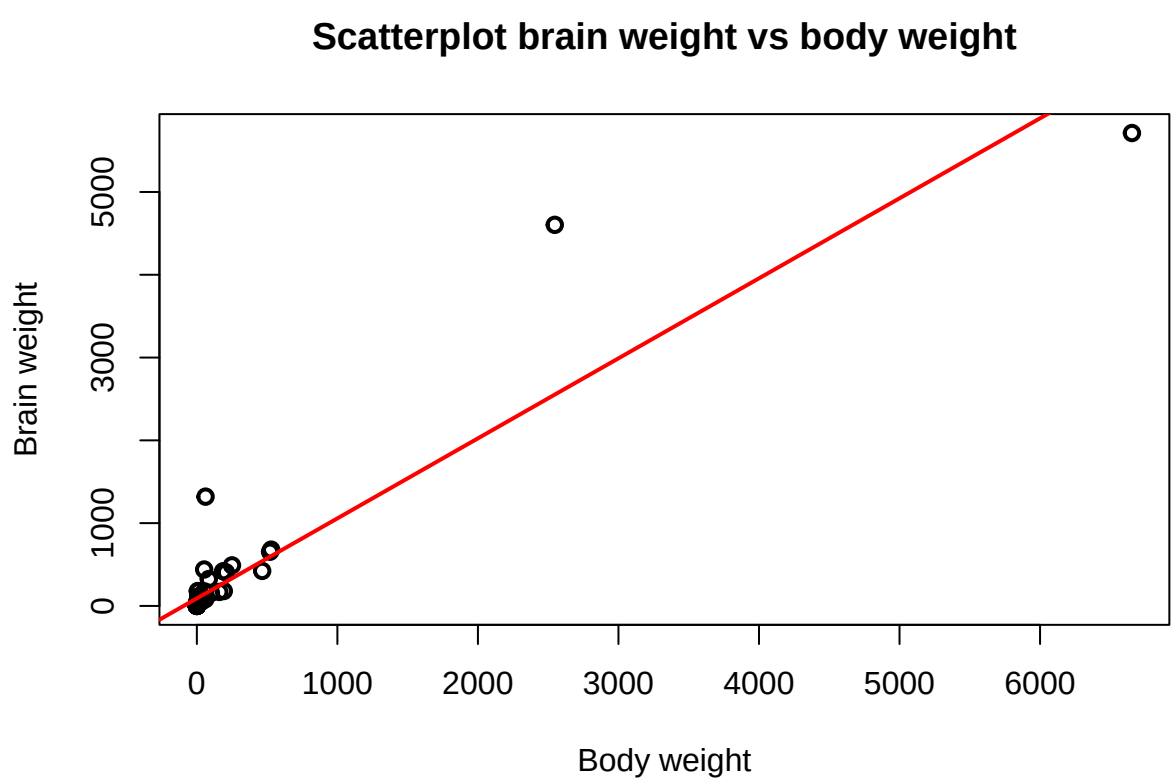


9.3.1 Standard default model

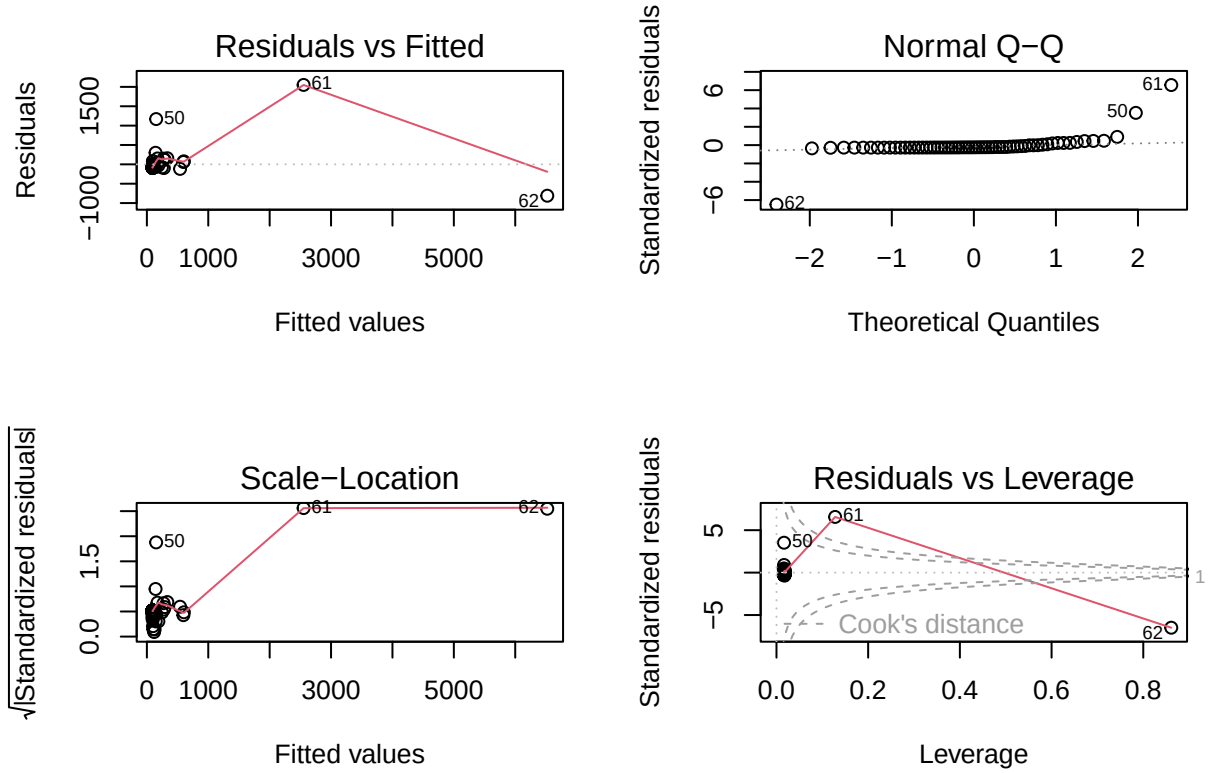
```
#! First attempt without doing nothing: we expect we will have problems
#! as the gaussianity assumption (needed later) will fail
result = lm(Y ~ X)
summary(result)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -810.07  -88.52  -79.64  -13.02 2050.33
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 91.00440   43.55258    2.09   0.0409 *
## X            0.96650    0.04766   20.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 334.7 on 60 degrees of freedom
## Multiple R-squared:  0.8727, Adjusted R-squared:  0.8705
## F-statistic: 411.2 on 1 and 60 DF,  p-value: < 2.2e-16
```

```
coef = result$coef
plot(X, Y, main='Scatterplot brain weight vs body weight', lwd=2,
     xlab='Body weight', ylab='Brain weight')
abline(coef[1],coef[2], lwd=2,col='red')
```




```
# diagnostics of the residuals
par(mfrow=c(2,2))
plot(result)
```

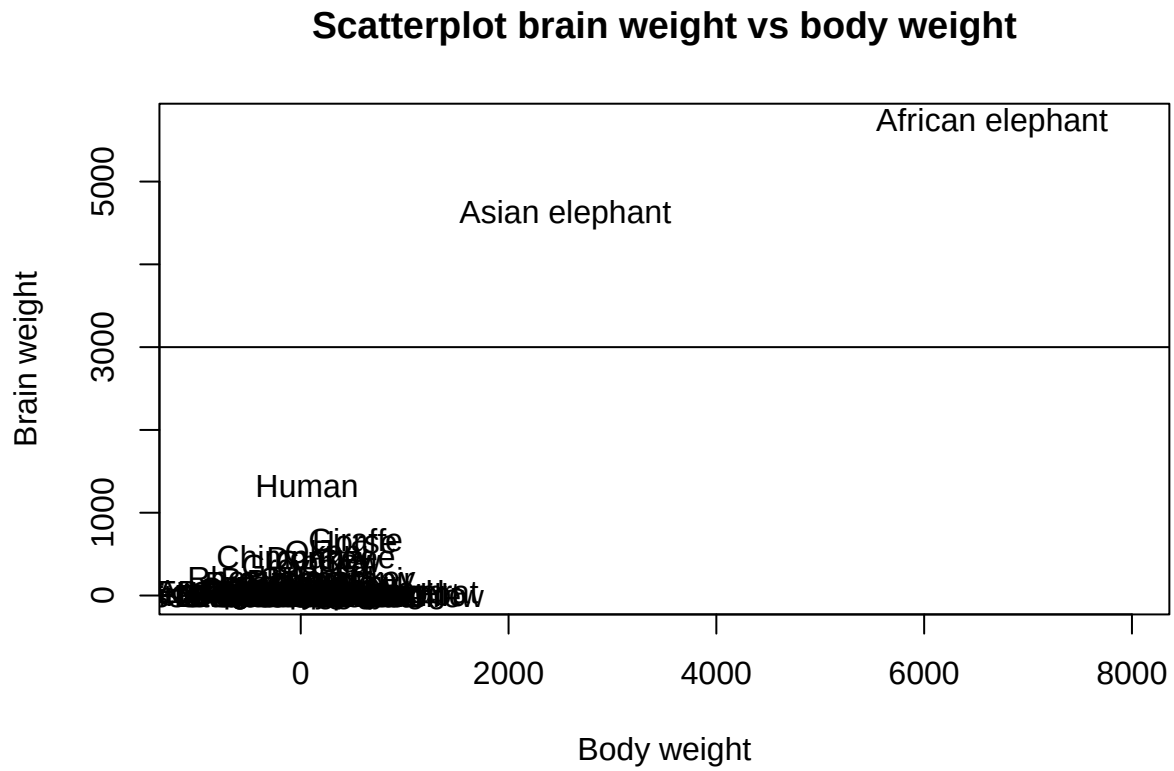



```
#! The most important picture here is the one of Cook distance, we can
#! see that there are two points really bad positioned

shapiro.test(residuals(result))
```

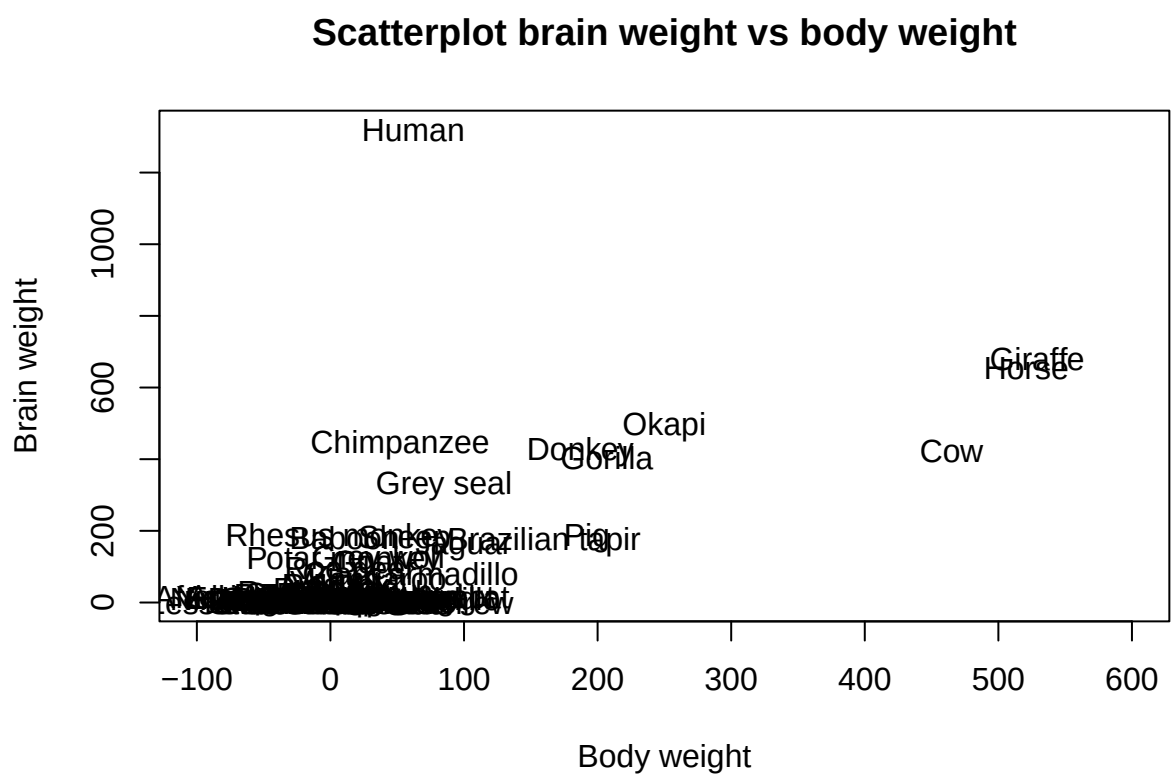
```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(result)
## W = 0.41112, p-value = 2.316e-14
```

9.3.2 Remove outliers

```
# exclude outliers?
plot(X, Y, main='Scatterplot brain weight vs body weight', lwd=2,
     xlab='Body weight', ylab='Brain weight',col='white',xlim=c(-1000,8000))
text(X, Y,dimnames(data)[[1]],cex=1)
abline(h=3000)
```




```
plot(X[which(Y<3000)], Y[which(Y<3000)],
     main='Scatterplot brain weight vs body weight', lwd=2,
     xlab='Body weight', ylab='Brain weight',col='white',xlim=c(-100,600))
text(X[which(Y<3000)], Y[which(Y<3000)],dimnames(data)[[1]],cex=1)
```



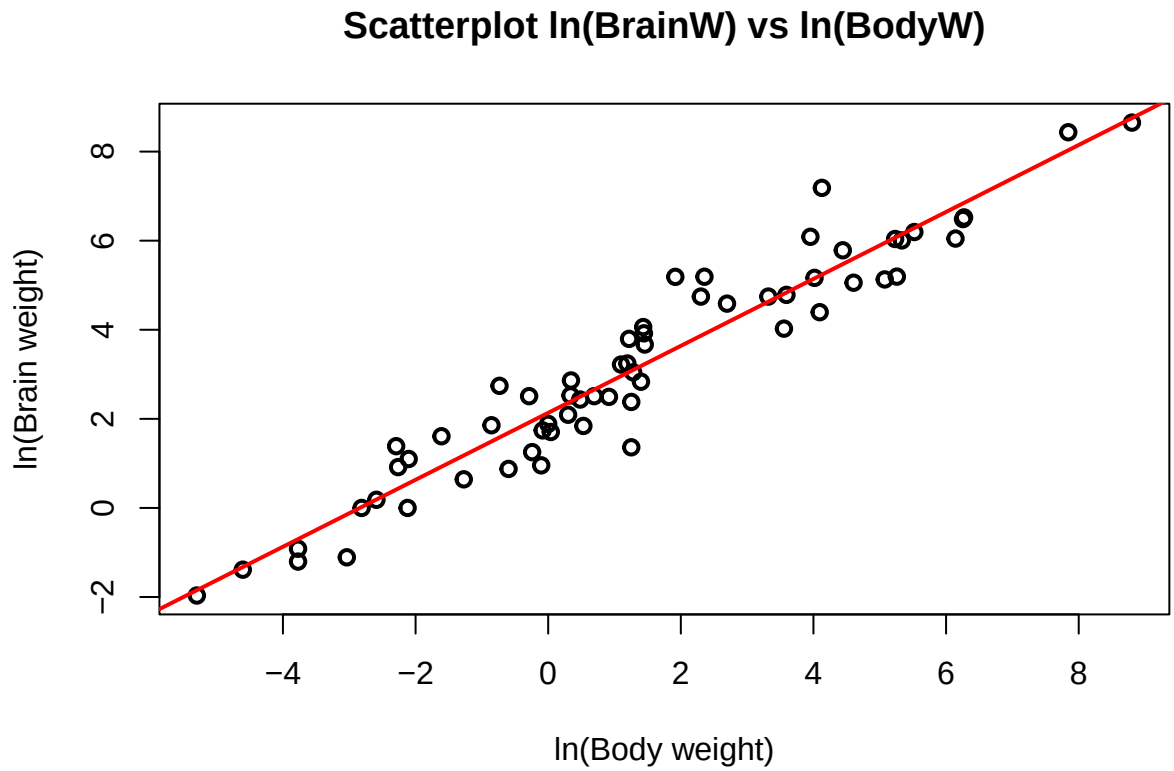
```r
#! there are still lots of points concentrated in some space
```

9.3.3 Transform data to ln

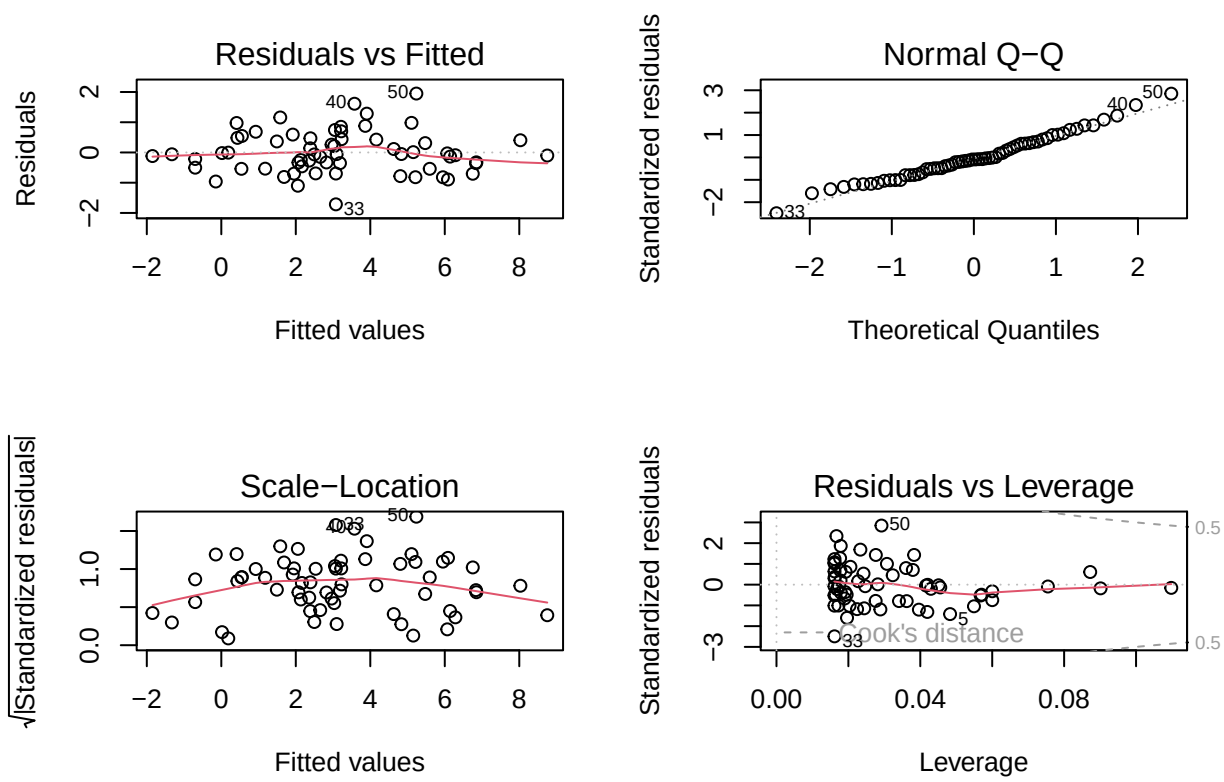
```r
# Then, logarithmic transformation of the data!
#! as we saw kind of an exp distribution
#!   - lots of mass for small values
#!   - fewer points on high values
log.X = log(X)
log.Y = log(Y)
#! we had problems on both variables, so we transform both

plot(log.X, log.Y, main='Scatterplot ln(BrainW) vs ln(BodyW)', lwd=2,
     xlab='ln(Body weight)', ylab='ln(Brain weight)')

result.log = lm(log.Y ~ log.X)
#! linear model on a log transformation means that we have a multiplicative
#! effect on the covariates, instead of an additive effect
#!    lnY = b0 + b1*lnX + b2*lnW
#!      Y = exp(b0) * X^b1 * W^b2
summary(result.log)
```

```
##
## Call:
## lm(formula = log.Y ~ log.X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.71550 -0.49228 -0.06162  0.43597  1.94829
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.13479    0.09604   22.23   <2e-16 ***
## log.X        0.75169    0.02846   26.41   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6943 on 60 degrees of freedom
## Multiple R-squared:  0.9208, Adjusted R-squared:  0.9195
## F-statistic: 697.4 on 1 and 60 DF,  p-value: < 2.2e-16
```

```r
coef.log = result.log$coef
abline(coef.log[1],coef.log[2], lwd=2,col='red')
```




```
# diagnostics of the residuals

par(mfrow=c(2,2))
plot(result.log)
```

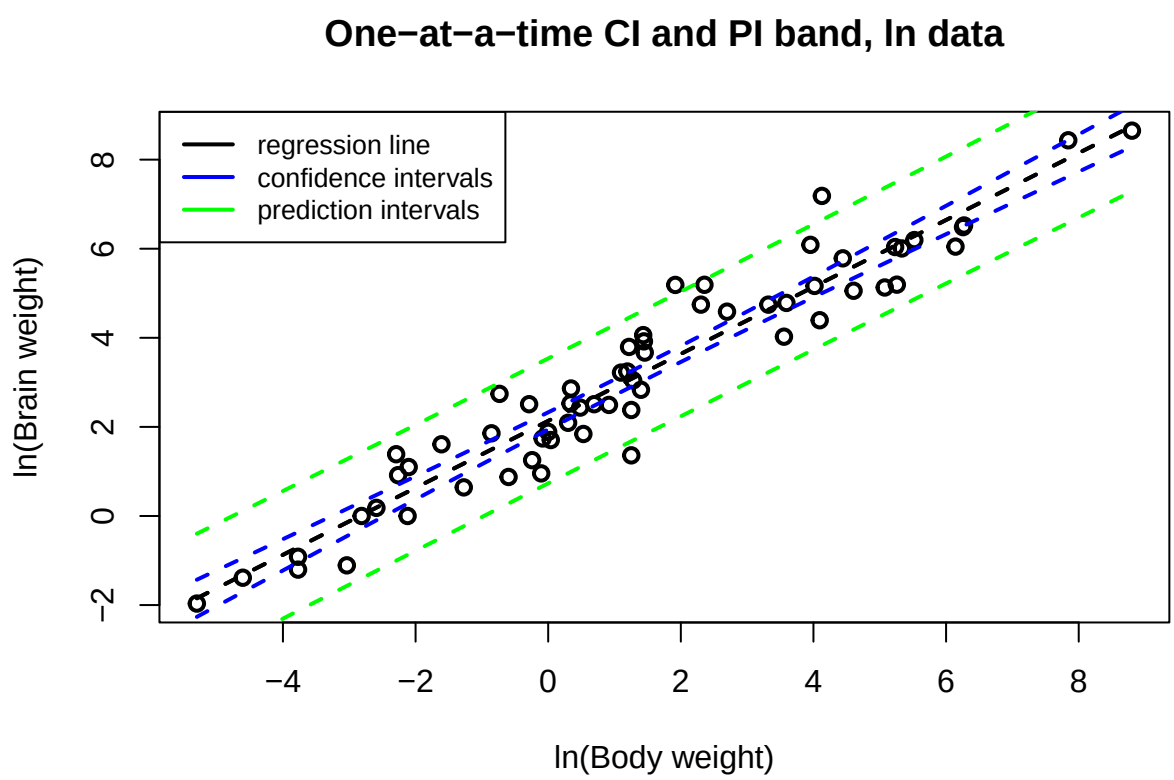


```r
shapiro.test(residuals(result.log))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(result.log)
## W = 0.98268, p-value = 0.5293
```

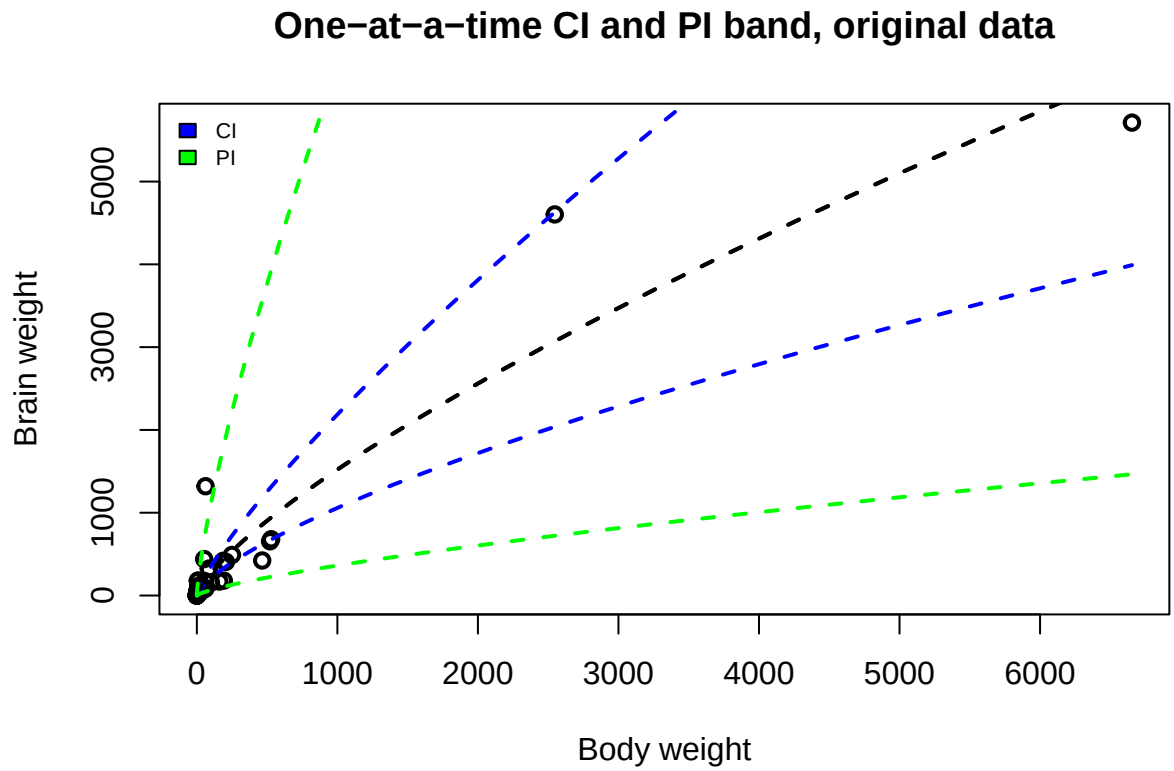
9.3.4 CIs and PIs, ln data

```r
# confidence intervals and prediction intervals
plot(log.X, log.Y, main='One-at-a-time CI and PI band, ln data', lwd=2,
     xlab='ln(Body weight)', ylab='ln(Brain weight)')

X.new.log = data.frame(log.X = seq(min(log.X), max(log.X), len=100))

IC.log =predict(result.log ,X.new.log,interval="confidence",level=0.95)
matplot(X.new.log,IC.log,add=T,type='l',col=c('black','blue','blue'),lwd=2,lty=2)

IP.log =predict(result.log ,X.new.log,interval="prediction",level=0.95)
matplot(X.new.log,IP.log,add=T,type='l',col=c('black','green','green'),lwd=2,lty=2)

legend('topleft',
       legend=c('regression line','confidence intervals','prediction intervals'),
       col=c('black','blue','green'), lwd=2, cex=0.85)
```



9.3.5 CIs and PIs, original data

```r
# plot on the original data
plot(X, Y, main='One-at-a-time CI and PI band, original data', lwd=2,
     xlab='Body weight', ylab='Brain weight')
IC = exp(IC.log)
IP = exp(IP.log)
X.new = exp(X.new.log)
matplot(X.new,IC,add=T,type='l',col=c('black','blue','blue'),lwd=2,lty=2)
matplot(X.new,IP,add=T,type='l',col=c('black','green','green'),lwd=2,lty=2)
legend("topleft",c("CI","PI"),fill=c("blue","green"), bty='n', cex = 0.7)
```



9.3.6 Model with clusters, on dataset earthquake

```
### Example 4: Earthquakes
###--------------------------------
Q = cbind(quakes[,1:2], depth=-quakes[,3]/100)

d = dist(Q)
clusterw = cutree(hclust(d, method='ward.D2'), 2)

open3d()
```

```
## wgl
##  14
```

```
par3d(windowRect=c(680,40,1350,720))
points3d(x=Q$lat, y=Q$long, z=Q$depth, size=4, col=clusterw+1, aspect = T)
box3d()
axes3d()
# close3d()
```

9.3.7 Standard default model

```
### Model 1 (all together), forgetting about the cluster structure
###-----------------------
# Model:
# depth = beta0 + beta1*lat + beta2*long + eps
fit  = lm(depth ~ lat + long, data=Q)
summary(fit)
```

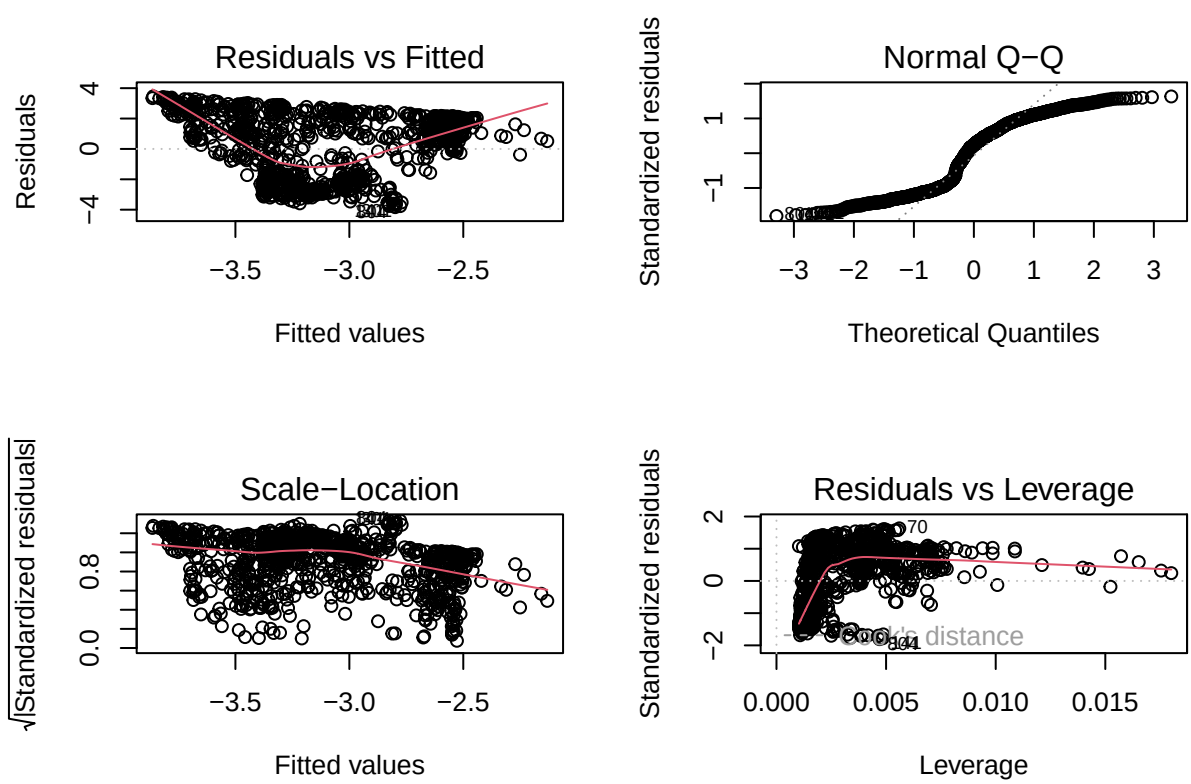
```
##
## Call:
## lm(formula = depth ~ lat + long, data = Q)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8314 -2.2432  0.5165  1.9615  3.4606
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.47977    2.04760   3.653 0.000273 ***
## lat         -0.04136    0.01436  -2.880 0.004068 **
## long        -0.06379    0.01190  -5.360 1.04e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.126 on 997 degrees of freedom
## Multiple R-squared:  0.02894,    Adjusted R-squared:  0.02699
## F-statistic: 14.86 on 2 and 997 DF,  p-value: 4.387e-07
```

```
#! Strange, very low R^2, but all covariates seems to be significant

open3d()
```

```
## wgl
##  15
```

```
#! Regression line will now be a plane, as regression region is of dim=n-1
#! Blue points are fitted values, but are not projections (as they are in pca),
#! like in the regression line the residuals are not the projection distance

par3d(windowRect=c(680,40,1350,720))
points3d(x=Q$lat, y=Q$long, z=Q$depth, size=4, col=clusterw+1, aspect = T)
box3d()
axes3d()
points3d(x=Q$lat, y=Q$long, z=fitted(fit), size=4, col = 'blue')
surface3d(range(Q$lat), range(Q$long),
          matrix(predict(fit, expand.grid(lat=range(Q$lat), long=range(Q$long))),2,2),
          alpha = 0.5)
# close3d()


par(mfrow=c(2,2))
plot(fit)
```



```r
#! we have a pattern, not normal qq-plot, again a linear pattern, ecc

shapiro.test(rstandard(fit))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  rstandard(fit)
## W = 0.90507, p-value < 2.2e-16
```

```r
#! From this diagnostic we see that our model is still good, not to directly throw
#! away, as we have significant values for parameters. So we just need to fix it
```

9.3.8 Model with a dummy variable

```r
### Model 2 (dummy variable), we want to use the cluster structure we found
###--------------------------
dummy = clusterw - 1
# 0 = red
# 1 = green
#! dummy variables should always be binary

Qd = cbind(Q, dummy)
head(Qd)
```

```
##      lat   long depth dummy
## 1 -20.42 181.62 -5.62     0
## 2 -20.62 181.03 -6.50     0
## 3 -26.00 184.10 -0.42     0
## 4 -17.97 181.66 -6.26     0
## 5 -20.42 181.96 -6.49     0
## 6 -19.68 184.31 -1.95     0
```

```r
# Model:
# depth = beta0       + beta1*lat       + beta2*long        +
#       + beta3*dummy + beta4*dummy*lat + beta5*dummy*long  + eps
#
#! so singular, group dummy, interaction; effects
#! like lat and long have baseline effects, but then one depending on the
#! dummy group in which we are.
#! It is better than separating data and then do two models. As this unique one
#! is better at computing the residuals.
#
# ie
# depth = B0[g] + B1[g]*lat + B2[g]*long + eps
# with B0[g]=beta0       if the unit is in group st dummy=0 (red)
#      B0[g]=beta0+beta3 if the unit is in group st dummy=1 (green)
#      B1[g]=beta1       if the unit is in group st dummy=0 (red)
#      B1[g]=beta1+beta4 if the unit is in group st dummy=1 (green)
#      B2[g]=beta2       if the unit is in group st dummy=0 (red)
#      B2[g]=beta2+beta5 if the unit is in group st dummy=1 (green)

fitd = lm(depth ~ lat + long + dummy + lat:dummy + long:dummy, data=Qd)
summary(fitd)
```

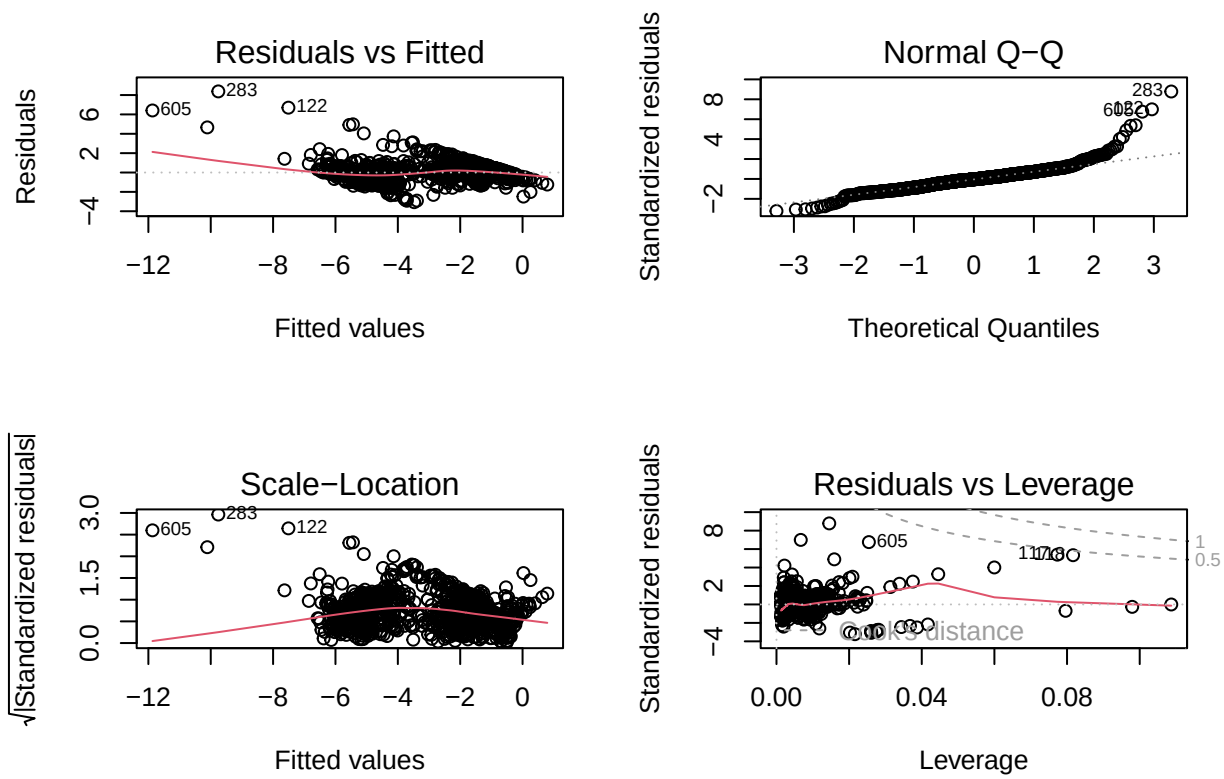
```
##
## Call:
## lm(formula = depth ~ lat + long + dummy + lat:dummy + long:dummy,
##     data = Qd)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0644 -0.5425 -0.0247  0.4532  8.3771
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.896e+02  3.351e+00  -56.59   <2e-16 ***
## lat         -3.304e-01  8.412e-03  -39.28   <2e-16 ***
## long         9.813e-01  1.788e-02   54.88   <2e-16 ***
## dummy        3.061e+02  9.119e+00   33.57   <2e-16 ***
```

```
## lat:dummy   -3.924e-02  2.929e-02   -1.34    0.181
## long:dummy  -1.718e+00  5.532e-02  -31.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.961 on 994 degrees of freedom
## Multiple R-squared:  0.8022, Adjusted R-squared:  0.8012
## F-statistic: 806.2 on 5 and 994 DF,  p-value: < 2.2e-16
```

```
#! R^2 improved
#! Specifying that interaction allows our model to fit two planes

# fitd = lm(depth ~ lat + long + dummy, data=Qd)
#! in this case we would have get two parallel planes

# Fitted model:
open3d()
```

```
## wgl
##  16
```

```
par3d(windowRect=c(680,40,1350,720))
points3d(x=Q$lat, y=Q$long, z=Q$depth, size=4, col=clusterw+1, aspect = T)
points3d(x=Qd$lat, y=Qd$long, z=fitted(fitd), size=4, col = 'blue')

surface3d(range(Q$lat), range(Q$long),
          matrix(predict(fitd, expand.grid(lat=range(Q$lat), long=range(Q$long),
                                           dummy=c(1))),2,2),
          alpha = 0.5, col='green')
surface3d(range(Q$lat), range(Q$long),
          matrix(predict(fitd, expand.grid(lat=range(Q$lat), long=range(Q$long),
                                           dummy=c(0))),2,2),
          alpha = 0.5, col='red')
box3d()
axes3d()
# close3d()

# Residuals:
par(mfrow=c(2,2))
plot(fitd)
```



```r
#! It seems we still have some problems (NW) but they dont seem to be influential (SE)

shapiro.test(rstandard(fitd))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  rstandard(fitd)
## W = 0.8803, p-value < 2.2e-16
```

9.3.9 Test for the difference in the two models

```r
# test: are the two planes needed?
#! / H0: simple model at the beginning
#! \ H1: new model we just did
A = rbind(c(0,0,0,1,0,0), c(0,0,0,0,1,0), c(0,0,0,0,0,1))
#! as with A*beta we filter beta3, beta4, beta5, the one with dummy
b = c(0,0,0)
linearHypothesis(fitd, A, b)
```

```
## Linear hypothesis test
##
## Hypothesis:
## dummy = 0
## lat:dummy = 0
## long:dummy = 0
##
## Model 1: restricted model
## Model 2: depth ~ lat + long + dummy + lat:dummy + long:dummy
##
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    997 4506.6
## 2    994  918.1  3    3588.5 1295.1 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.3.10 Model reduction

```r
# Reduce the model:
summary(fitd)
```

```
##
## Call:
## lm(formula = depth ~ lat + long + dummy + lat:dummy + long:dummy,
##     data = Qd)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0644 -0.5425 -0.0247  0.4532  8.3771
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.896e+02  3.351e+00  -56.59   <2e-16 ***
## lat         -3.304e-01  8.412e-03  -39.28   <2e-16 ***
## long         9.813e-01  1.788e-02   54.88   <2e-16 ***
## dummy        3.061e+02  9.119e+00   33.57   <2e-16 ***
## lat:dummy   -3.924e-02  2.929e-02   -1.34    0.181
## long:dummy  -1.718e+00  5.532e-02  -31.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.961 on 994 degrees of freedom
## Multiple R-squared:  0.8022, Adjusted R-squared:  0.8012
## F-statistic: 806.2 on 5 and 994 DF,  p-value: < 2.2e-16
```

```r
### Model 3 (reduced model)
###-------------------------
# Model:
# depth = beta0 +beta1*lat +beta2*long +beta3*dummy +beta4*dummy*long +eps
# ie,
```

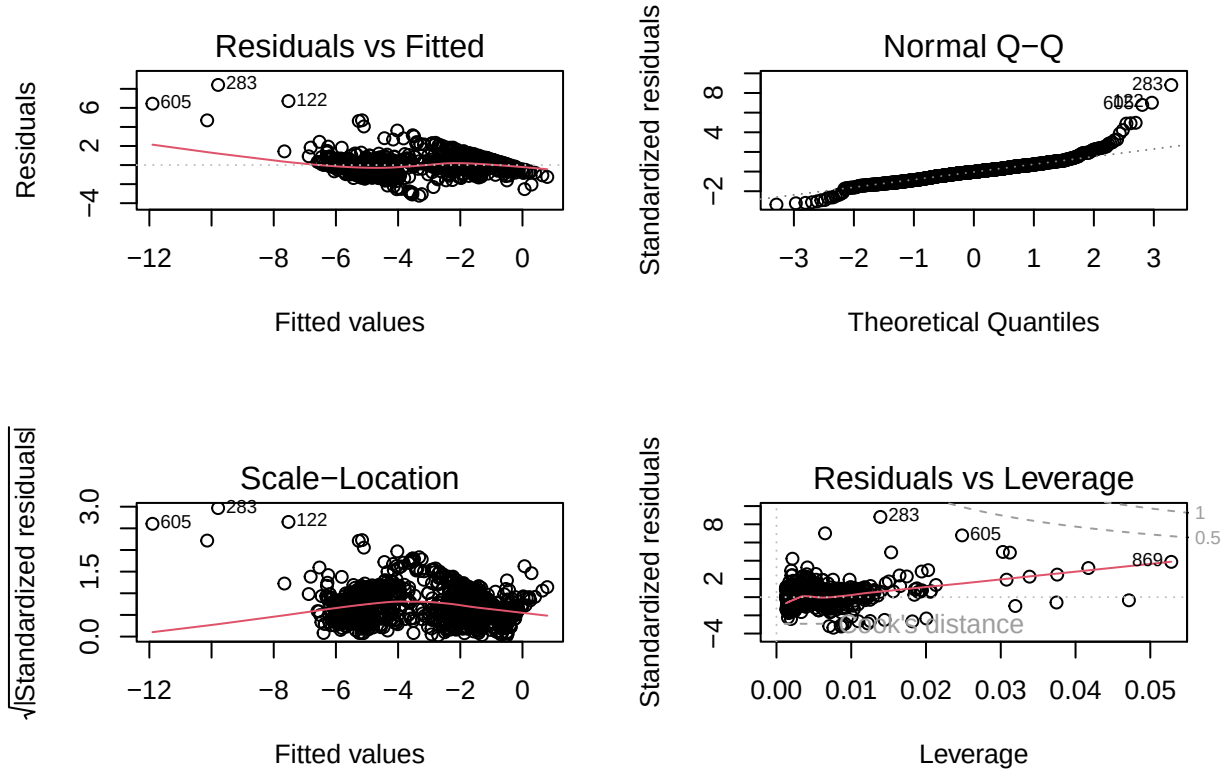
```r
# depth = B0[g] + B1*lat + B2[g]*long
# with B0[g]=beta0       if the unit is in group s.t. dummy=0 (red)
#      B0[g]=beta0+beta3 if the unit is in group s.t. dummy=1 (green)
#      B1=beta1
#      B2[g]=beta2       if the unit is in group s.t. dummy=0 (red)
#      B2[g]=beta2+beta5 if the unit is in group s.t. dummy=1 (green)

fitD = lm(depth ~ lat + long + dummy + long:dummy, data=Qd)
summary(fitD)
```

```
##
## Call:
## lm(formula = depth ~ lat + long + dummy + long:dummy, data = Qd)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2167 -0.5394 -0.0330  0.4690  8.4094
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.903e+02  3.316e+00  -57.39   <2e-16 ***
## lat         -3.336e-01  8.061e-03  -41.39   <2e-16 ***
## long         9.845e-01  1.772e-02   55.55   <2e-16 ***
## dummy        2.989e+02  7.378e+00   40.52   <2e-16 ***
## long:dummy  -1.672e+00  4.291e-02  -38.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9614 on 995 degrees of freedom
## Multiple R-squared:  0.8018, Adjusted R-squared:  0.801
## F-statistic:  1006 on 4 and 995 DF,  p-value: < 2.2e-16
```

```r
# Fitted model
open3d()
```

```
## wgl
##   1
```

```r
par3d(windowRect=c(680,40,1350,720))
points3d(x=Q$lat, y=Q$long, z=Q$depth, size=4, col=clusterw+1, aspect = T)
axes3d()
points3d(x=Qd$lat, y=Qd$long, z=fitted(fitD), size=4, col = 'blue')
surface3d(range(Q$lat), range(Q$long),
          matrix(predict(fitD, expand.grid(lat = range(Q$lat), long = range(Q$long),
                                           dummy=c(1))),2,2),
          alpha = 0.5, col='green')
surface3d(range(Q$lat), range(Q$long),
          matrix(predict(fitD, expand.grid(lat = range(Q$lat), long = range(Q$long),
                                           dummy=c(0))),2,2),
          alpha = 0.5, col='red')

# close3d()

# Residuals:
par(mfrow=c(2,2))
plot(fitD)
```




```
#! Not much better, but always choose the model with less parameters

shapiro.test(rstandard(fitD))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  rstandard(fitD)
## W = 0.88297, p-value < 2.2e-16
```

```
### Homework: Fit a linear model with quadratic regressors
### (lat, long, lat^2, long^2, lat:long)
#! lat:long interaction may be interesting as we are dealing with spacial data
### Perform appropriate statistical tests to answer the following questions:
### Q1: are the quadratic terms needed?
### Q2: is the latitude needed?
### Q3: is the longitude needed?
```

9.4 Bias/Variance trade-off, on simulated data

```
##### Example 5: simulated data (Bias-variance trade-off)
#####-----------------------------------

# generation of training set and test set
set.seed(1)

# true model: regressors x, x^2, x^3; coefficients (1,1,1,-1)
f = function(x){1+x+x^2-x^3}
sigma = 0.25
x = seq(-1, 1.5, length = 21)
y = f(x) + rnorm(21, sd = sigma)
y.new = f(x) + rnorm(21, sd = sigma)

# build design matrix, by cbinding x^0 x^1 x^2 ... x^21
data = NULL
for(p in 0:20)
  data = cbind(data, x^p)
```

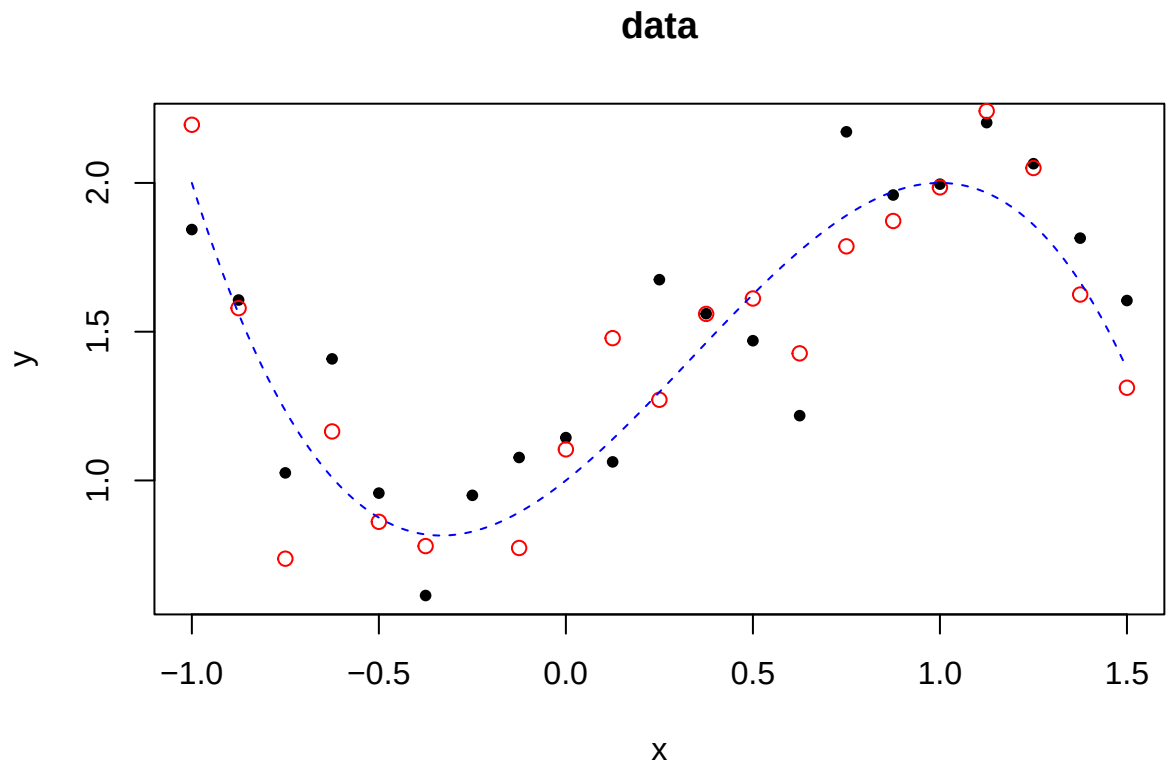
```r
colnames(data) = c(paste('x', 0:20, sep=''))
data = data.frame(data)
head(data)
```

```
##   x0     x1       x2          x3         x4           x5          x6
## 1  1 -1.000 1.000000 -1.00000000 1.00000000 -1.000000000 1.000000000
## 2  1 -0.875 0.765625 -0.66992188 0.58618164 -0.512908936 0.448795319
## 3  1 -0.750 0.562500 -0.42187500 0.31640625 -0.237304688 0.177978516
## 4  1 -0.625 0.390625 -0.24414062 0.15258789 -0.095367432 0.059604645
## 5  1 -0.500 0.250000 -0.12500000 0.06250000 -0.031250000 0.015625000
## 6  1 -0.375 0.140625 -0.05273438 0.01977539 -0.007415771 0.002780914
##             x7           x8            x9          x10           x11
## 1 -1.000000000 1.0000000000 -1.0000000000 1.000000e+00 -1.000000e+00
## 2 -0.392695904 0.3436089158 -0.3006578013 2.630756e-01 -2.301911e-01
## 3 -0.133483887 0.1001129150 -0.0750846863 5.631351e-02 -4.223514e-02
## 4 -0.037252903 0.0232830644 -0.0145519152 9.094947e-03 -5.684342e-03
## 5 -0.007812500 0.0039062500 -0.0019531250 9.765625e-04 -4.882812e-04
## 6 -0.001042843 0.0003910661 -0.0001466498 5.499367e-05 -2.062263e-05
##            x12           x13          x14           x15          x16
## 1 1.000000e+00 -1.000000e+00 1.000000e+00 -1.000000e+00 1.000000e+00
## 2 2.014172e-01 -1.762401e-01 1.542101e-01 -1.349338e-01 1.180671e-01
## 3 3.167635e-02 -2.375726e-02 1.781795e-02 -1.336346e-02 1.002260e-02
## 4 3.552714e-03 -2.220446e-03 1.387779e-03 -8.673617e-04 5.421011e-04
## 5 2.441406e-04 -1.220703e-04 6.103516e-05 -3.051758e-05 1.525879e-05
## 6 7.733484e-06 -2.900057e-06 1.087521e-06 -4.078205e-07 1.529327e-07
##             x17          x18           x19          x20
## 1 -1.000000e+00 1.000000e+00 -1.000000e+00 1.000000e+00
## 2 -1.033087e-01 9.039511e-02 -7.909572e-02 6.920876e-02
## 3 -7.516947e-03 5.637710e-03 -4.228283e-03 3.171212e-03
## 4 -3.388132e-04 2.117582e-04 -1.323489e-04 8.271806e-05
## 5 -7.629395e-06 3.814697e-06 -1.907349e-06 9.536743e-07
## 6 -5.734975e-08 2.150616e-08 -8.064809e-09 3.024303e-09
```

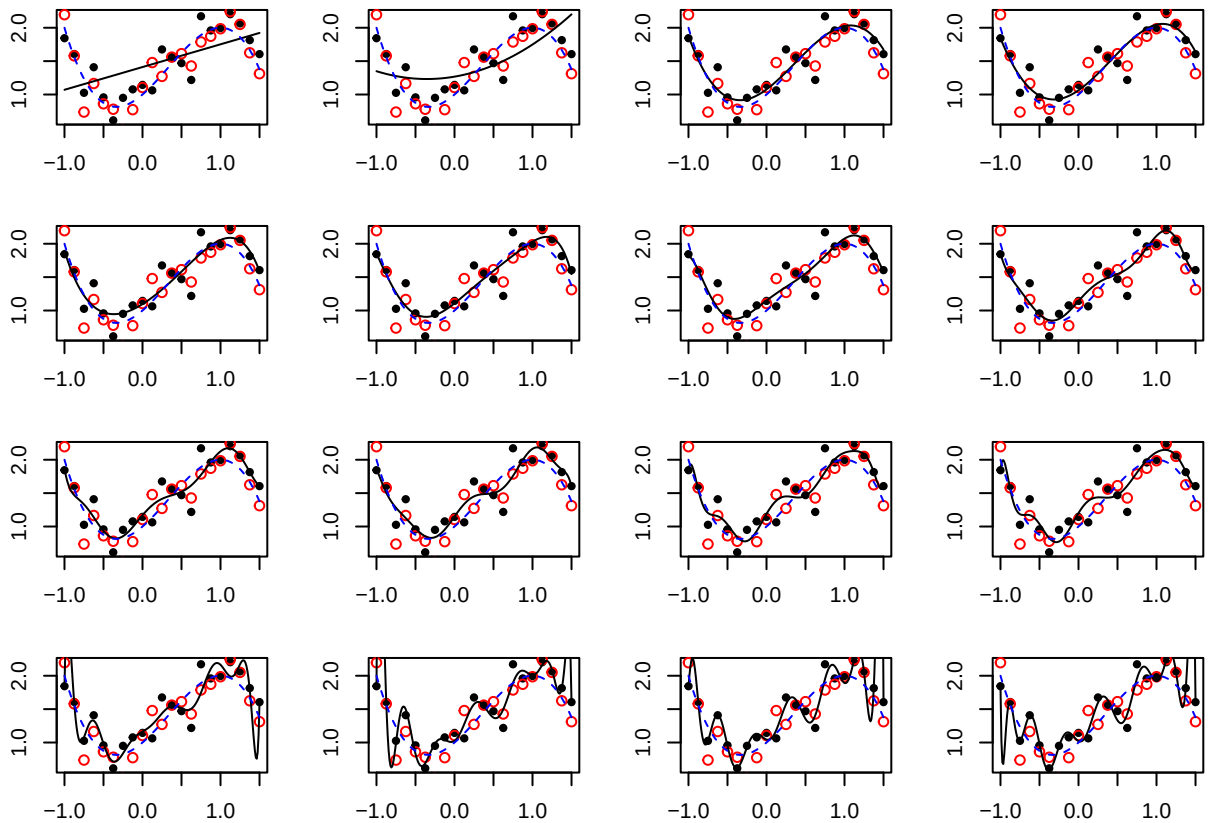
```r
dim(data)
```

```
## [1] 21 21
```

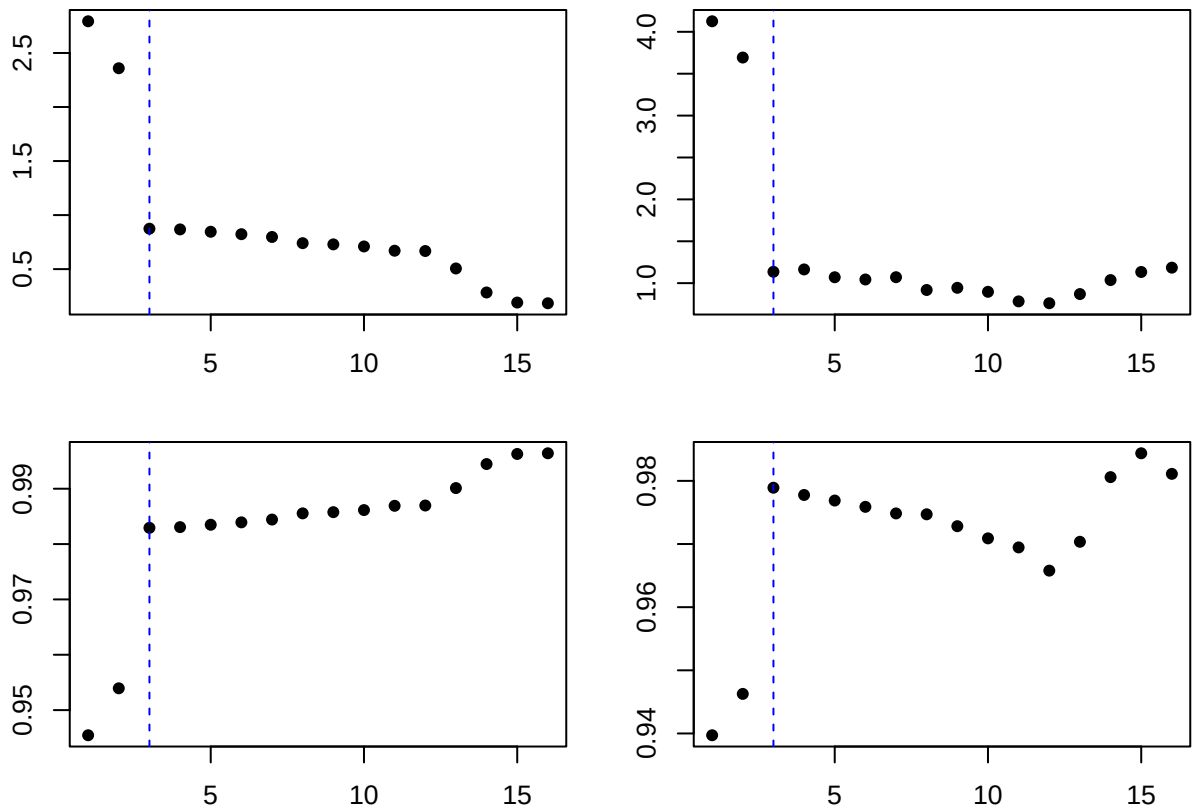
```r
#! And we want to try incremental cols of Z to see if/how we can get better
# grid to plot
data.plot = NULL
x.plot = seq(-1, 1.5, length = 210)
for(p in 0:20)
  data.plot = cbind(data.plot, x.plot^p)
colnames(data.plot) = c(paste('x', 0:20, sep=''))
data.plot = data.frame(data.plot)

# plot of the training set, test set and "true" mean curve
plot(x, y, pch=20, main="data")                # training set
points(x, y.new, col='red')                    # test set
lines(x.plot, f(x.plot), col='blue', lty=2)    # true mean
```



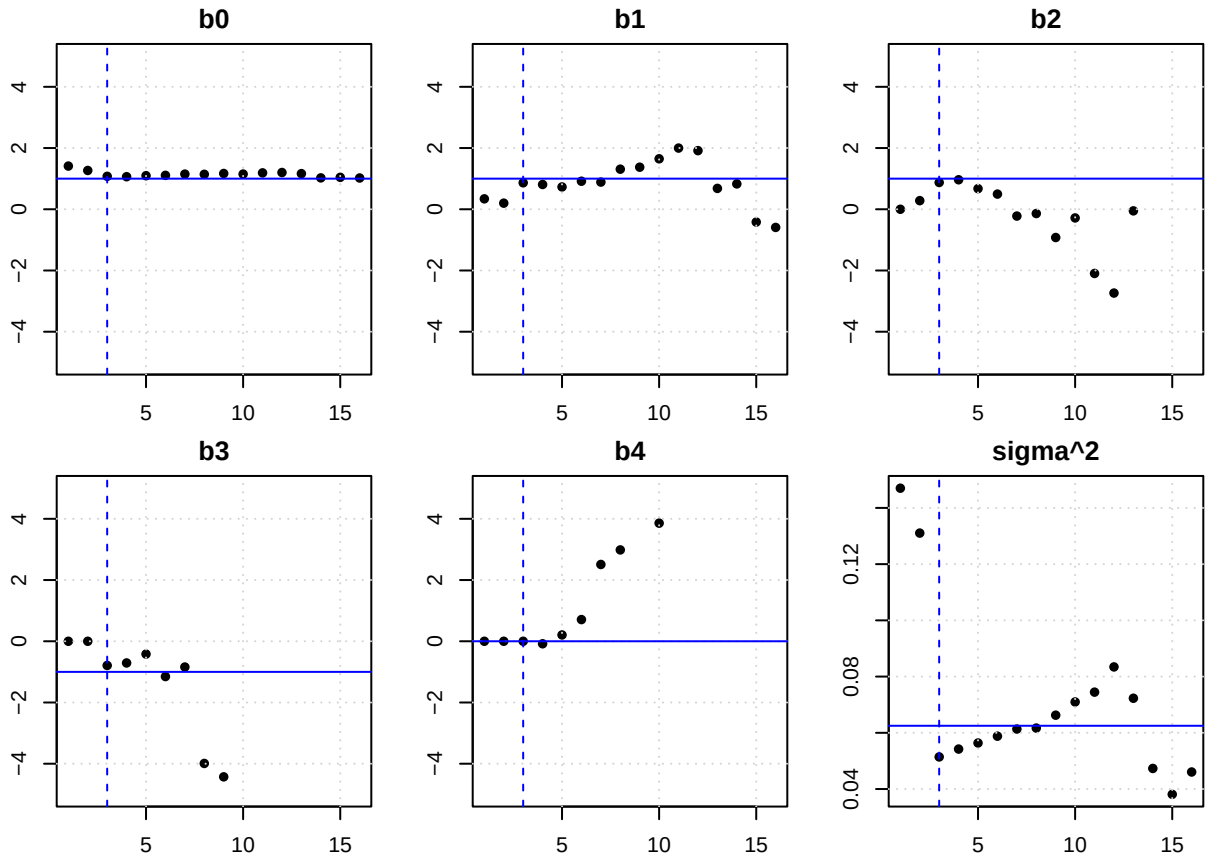

```r
# regression with polynomials of increasing order
SSres = SSres.new = s2 <- b <- R2 <- R2.adj <- NULL
n = 21

par(mfrow=c(4,4), mar=rep(2,4))
for(p in 1:16)
{
  fit = lm(y ~ 0 + . , data[,1:(p+1)])
  plot(x, y, pch=20,xlab="",ylab="",)
  points(x, y.new, col='red')
  lines(x.plot, predict(fit, data.plot))
  lines(x.plot, f(x.plot), col='blue', lty=2)

  SSres = c(SSres, sum((y - fitted(fit))^2))
  SSres.new = c(SSres.new, sum((y.new - fitted(fit))^2))
  s2 = c(s2, sum((y - fitted(fit))^2)/(n - (p+1)))
  R2 = c(R2, summary(fit)$r.squared)
  R2.adj = c(R2.adj, summary(fit)$adj.r.squared)
  bp = rep(0,17)
  bp[1:(p+1)] = coefficients(fit)
  b = cbind(b, bp)
}
```




```
#! As we go up we are overfitting the data, interpolates the points
#! From (high bias, low variance) to (low bias, high variance)
#! variance about what will happen if we change the black dots

# compare some indices

par(mfrow=c(2,2))
plot(1:16, SSres, pch=16)
abline(v=3, col='blue', lty=2)
plot(1:16, SSres.new, pch=16)
abline(v=3, col='blue', lty=2)
plot(1:16, R2, pch=16)
abline(v=3, col='blue', lty=2)
plot(1:16, R2.adj, pch=16)
abline(v=3, col='blue', lty=2)
```




```
# compare parameter estimates
# true model: regressors x, x^2, x^2; coefficients (1,1,1,-1)
b.true = c(1,1,1,-1, rep(0,8))


par(mfrow=c(2,3))
for(i in 1:5)
{
  plot(1:16, b[i,], ylim=c(-5, 5), main=paste('b',i-1,sep=''), pch=16)
  grid()
  abline(v=3, col='blue', lty=2)
  abline(h=b.true[i], col='blue')
}
plot(1:16, s2, main='sigma^2', pch=16)
grid()
abline(v=3, col='blue', lty=2)
abline(h=sigma^2, col='blue')
```




```
#! The higher order plots have small sigma^2, but the values for the parameters
#! beta are completely over scale, as they try to interpolate
```

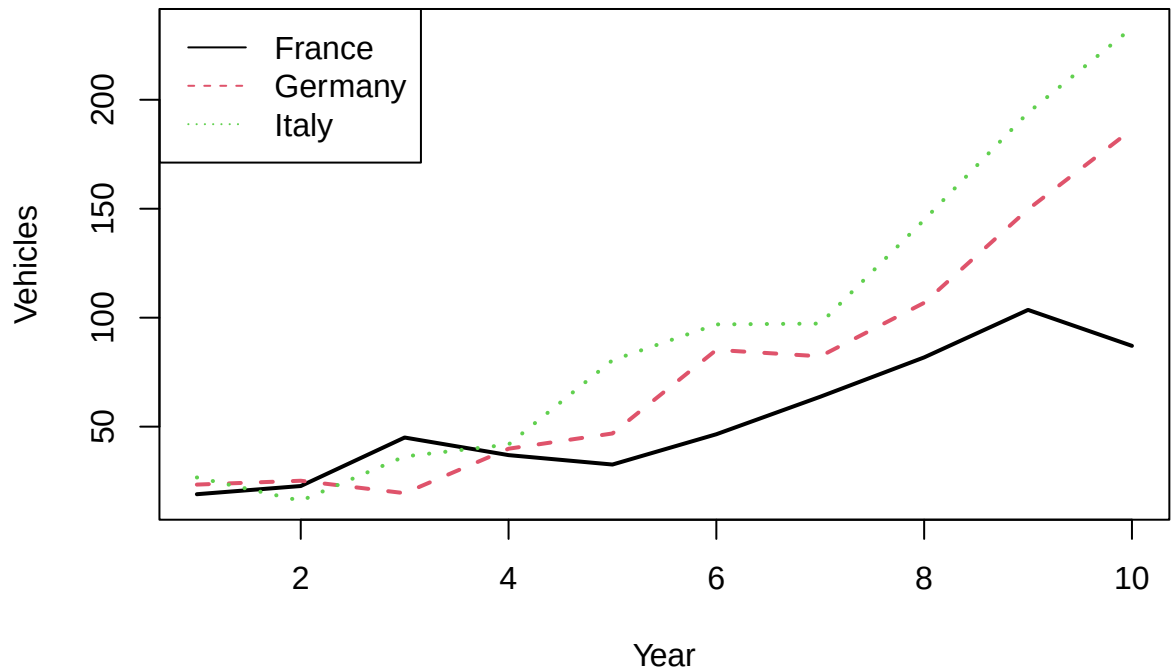
9.5 Exam exercise

```
##### Exercises on linear models
#####----------------------------

##### Problem 4 of 6/2/2007
#####-------------------------
# The file Pb4.txt reports the number Y (expressed in thousands of units)
# of vehicles registered annually in three countries of the European Union
# (France, Germany and Italy) during a reference period of 10 years.
# Recent economic models describe the behavior of this variable according
# to the model:
# Y | (X = x, G = g) = beta0.g + beta1.g * x^2 + eps
# with eps ~ N (0, sigma^2), x = 1, 2, ... , 10 (year) and
# g = France, Germany, Italy (EU country).
# (a) With the least squares method, estimate the 7 parameters of the model.
# (b) Using appropriate statistical tests, state if you deem necessary to
#     include into the model:
#     1. the variable x^2;
#     2. the variable G;
#     3. the effect of the variable G on the coefficient that multiplies
#        the regressor x^2;
#     4. the effect of the variable G on the intercept.
# (c) Once identified the "best model", build three prediction intervals
#     for the number of vehicles registered in the three countries
#     during the eleventh year, so that the three new observations
#     will fall simultaneously within the respective ranges with 95%
#     of probability.

pb4  = read.table('Lab09/Pb4.txt')
pb4
```

```
##    Francia Germania Italia
## 1    18.99    23.40  26.68
```

```
## 2    22.75    25.17  15.97
## 3    45.00    19.46  36.31
## 4    36.93    39.86  41.80
## 5    32.61    46.86  80.48
## 6    46.51    85.22  96.91
## 7    63.71    82.27  97.29
## 8    81.78   106.83 144.87
## 9   103.57   149.71 194.06
## 10   87.08   186.36 232.97
```

```
matplot(pb4, type='l',lwd=2, xlab='Year', ylab='Vehicles')
legend("topleft",c("France", "Germany", "Italy"),lty=1:3,col=1:3)
```



9.5.1 Qa

```
### question (a)
# (a) With the least squares method, estimate the 7 parameters of the model.

# We first build the design matrix and the vector of the responses
Year = rep(1:10,3)
Year
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5
## [26]  6  7  8  9 10
```

```
Reg = c(pb4[,1], pb4[,2], pb4[,3])
Reg
```

```
##  [1]  18.99  22.75  45.00  36.93  32.61  46.51  63.71  81.78 103.57  87.08
## [11]  23.40  25.17  19.46  39.86  46.86  85.22  82.27 106.83 149.71 186.36
## [21]  26.68  15.97  36.31  41.80  80.48  96.91  97.29 144.87 194.06 232.97
```

```
# Model: Reg = beta0.g + beta1.g*Year^2 + eps
# with g=France,Germany,Italy; E[eps]=0, Var(eps)=sigma^2

# We need to build appropriate dummy variables to account for the
```

```r
# levels of the categorical variable g=France,Germany,Italy (3 levels)
# g groups => g-1 dummy variables (3 groups => 2 dummy variables)

dFr = rep(c(1,0), c(10,20))      # dFr = 1 if France,  0 otherwise
dGer= rep(c(0,1,0), c(10,10,10)) # dGer= 1 if Germany, 0 otherwise
dFr
```

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

```r
dGer
```

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

```r
# Equivalent Model:
# Reg = b.0+b.1*dFr+b.2*dGer +b.3*Year^2+b.4*dFr*Year^2+b.5*dGer*Year^2 + eps

# Indeed:
# beta0.It=b.0;       beta1.It=b.3;
# beta0.Fr=b.0+b.1;   beta1.Fr=b.3+b.4;
# beta0.Ger=b.0+b.2;  beta1.Ger=b.3+b.5

dati = data.frame(Reg   = Reg,
                  Year2 = Year^2,
                  dFr   = rep(c(1,0), c(10,20)),      # dummy for France
                  dGer  = rep(c(0,1,0), c(10,10,10))) # dummy for Germany
dati
```

```
##       Reg Year2 dFr dGer
## 1   18.99     1   1    0
## 2   22.75     4   1    0
## 3   45.00     9   1    0
## 4   36.93    16   1    0
## 5   32.61    25   1    0
## 6   46.51    36   1    0
## 7   63.71    49   1    0
## 8   81.78    64   1    0
## 9  103.57    81   1    0
## 10  87.08   100   1    0
## 11  23.40     1   0    1
## 12  25.17     4   0    1
## 13  19.46     9   0    1
## 14  39.86    16   0    1
## 15  46.86    25   0    1
## 16  85.22    36   0    1
## 17  82.27    49   0    1
## 18 106.83    64   0    1
## 19 149.71    81   0    1
## 20 186.36   100   0    1
## 21  26.68     1   0    0
## 22  15.97     4   0    0
## 23  36.31     9   0    0
## 24  41.80    16   0    0
## 25  80.48    25   0    0
## 26  96.91    36   0    0
## 27  97.29    49   0    0
## 28 144.87    64   0    0
## 29 194.06    81   0    0
## 30 232.97   100   0    0
```

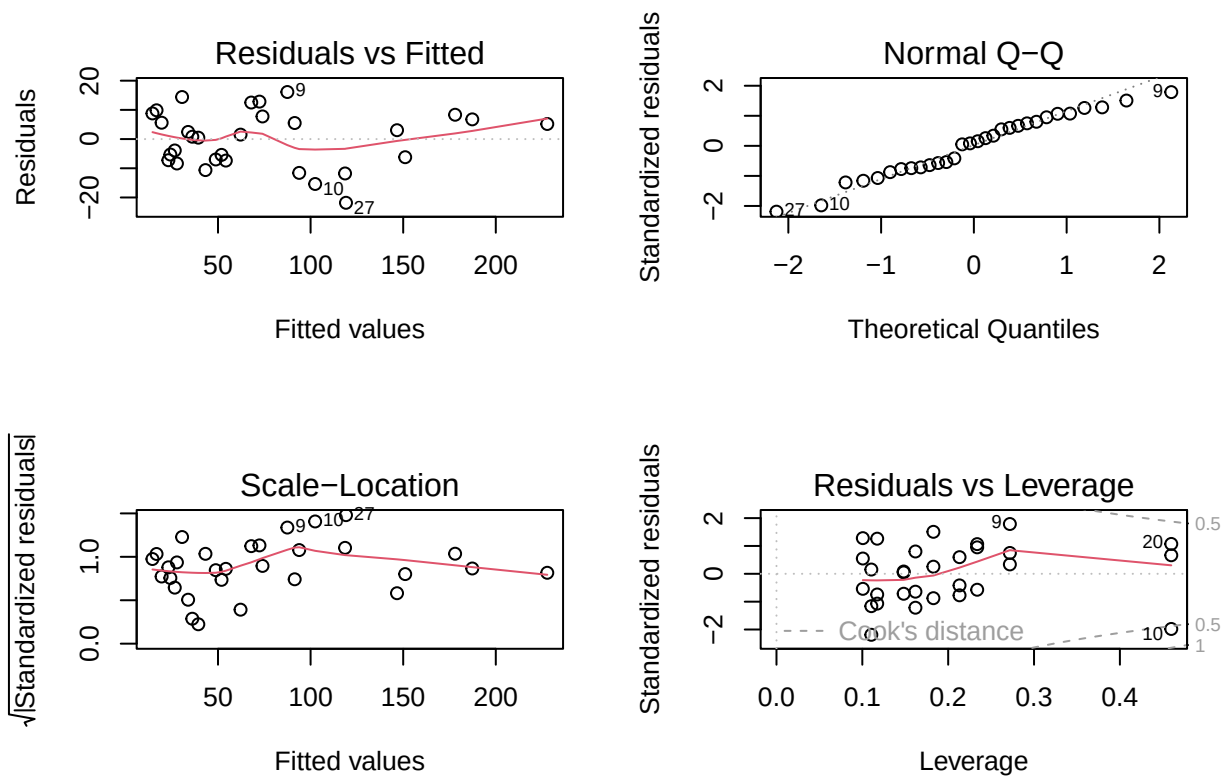
```r
fit = lm(Reg ~ dFr + dGer + Year2 + Year2:dFr + Year2:dGer, data=dati)

# Equivalent syntax:
# fit = lm(Reg ~ dFr + dGer + I(Year^2) + I(Year^2*dFr) + I(Year^2*dGer),
#          data=data.frame(Reg=Reg, Year=Year, dFr=rep(c(1,0), c(10,20)),
#                          dGer = rep(c(0,1,0), c(10,10,10))))

summary(fit)
```

```
##
## Call:
## lm(formula = Reg ~ dFr + dGer + Year2 + Year2:dFr + Year2:dGer,
##     data = dati)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.821  -7.176   1.165   7.502  16.117
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  14.6861     5.1874   2.831  0.00924 **
## dFr           8.8056     7.3360   1.200  0.24173
## dGer         -1.7229     7.3360  -0.235  0.81631
## Year2         2.1311     0.1031  20.678  < 2e-16 ***
## dFr:Year2    -1.3415     0.1458  -9.204 2.42e-09 ***
## dGer:Year2   -0.4804     0.1458  -3.296  0.00304 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.57 on 24 degrees of freedom
## Multiple R-squared:  0.9717, Adjusted R-squared:  0.9658
## F-statistic:   165 on 5 and 24 DF,  p-value: < 2.2e-16
```

9.5.2 Qb

```r
### question (b)
# (b) Using appropriate statistical tests, state if you deem necessary to
#     include into the model: ecc

par(mfrow=c(2,2))
plot(fit)
```




```r
shapiro.test(residuals(fit))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fit)
## W = 0.9713, p-value = 0.5753
```

```r
# 1. the variable x^2;
linearHypothesis(fit,
                 rbind(c(0,0,0,1,0,0),
                       c(0,0,0,0,1,0),
                       c(0,0,0,0,0,1)),
                 c(0,0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Year2 = 0
## dFr:Year2 = 0
## dGer:Year2 = 0
##
## Model 1: restricted model
## Model 2: Reg ~ dFr + dGer + Year2 + Year2:dFr + Year2:dGer
##
##   Res.Df   RSS Df Sum of Sq     F    Pr(>F)
## 1     27 85606
## 2     24  2679  3     82927 247.6 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# 2. the variable G;
linearHypothesis(fit,
                 rbind(c(0,1,0,0,0,0),
                       c(0,0,1,0,0,0),
                       c(0,0,0,0,1,0),
                       c(0,0,0,0,0,1)),
                 c(0,0,0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## dFr = 0
## dGer = 0
## dFr:Year2 = 0
## dGer:Year2 = 0
##
## Model 1: restricted model
## Model 2: Reg ~ dFr + dGer + Year2 + Year2:dFr + Year2:dGer
##
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1     28 21576.5
## 2     24  2679.4  4     18897 42.316 1.548e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# 3. the effect of the variable G onto the coefficient that multiplies
# the regressor x^2;
linearHypothesis(fit,
                 rbind(c(0,0,0,0,1,0),
                       c(0,0,0,0,0,1)),
                 c(0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## dFr:Year2 = 0
## dGer:Year2 = 0
##
## Model 1: restricted model
## Model 2: Reg ~ dFr + dGer + Year2 + Year2:dFr + Year2:dGer
##
##   Res.Df     RSS Df Sum of Sq     F    Pr(>F)
## 1     26 12390.2
## 2     24  2679.4  2    9710.8 43.49 1.046e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# 4. the effect of the variable G on the intercept.
linearHypothesis(fit,
                 rbind(c(0,1,0,0,0,0),
                       c(0,0,1,0,0,0)),
                 c(0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## dFr = 0
## dGer = 0
##
## Model 1: restricted model
## Model 2: Reg ~ dFr + dGer + Year2 + Year2:dFr + Year2:dGer
##
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     26 2944.1
## 2     24 2679.4  2    264.64 1.1852 0.3229
```

9.5.3 Qc

```r
### question (c)
fit2 = lm(Reg ~ Year2 + Year2:dFr + Year2:dGer, data=dati)
summary(fit2)
```

```
##
## Call:
## lm(formula = Reg ~ Year2 + Year2:dFr + Year2:dGer, data = dati)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.424  -8.235   1.961   7.074  19.965
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.04701    3.01618   5.652 6.07e-06 ***
## Year2        2.09523    0.08106  25.847  < 2e-16 ***
## Year2:dFr   -1.20765    0.09455 -12.773 1.04e-12 ***
## Year2:dGer  -0.50663    0.09455  -5.358 1.31e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.64 on 26 degrees of freedom
## Multiple R-squared:  0.9689, Adjusted R-squared:  0.9654
## F-statistic: 270.4 on 3 and 26 DF,  p-value: < 2.2e-16
```

```r
new_data = data.frame(Year2 = c(11,11,11)^2, dFr=c(1,0,0), dGer=c(0,1,0))
IP = predict(fit2, newdata=new_data, interval='prediction', level=1-0.05/3)
rownames(IP) = c('Fr','Ger','It')
IP
```

```
##          fit       lwr      upr
## Fr  124.4451  89.63246 159.2578
## Ger 209.2684 174.45569 244.0810
## It  270.5703 235.75764 305.3830
```

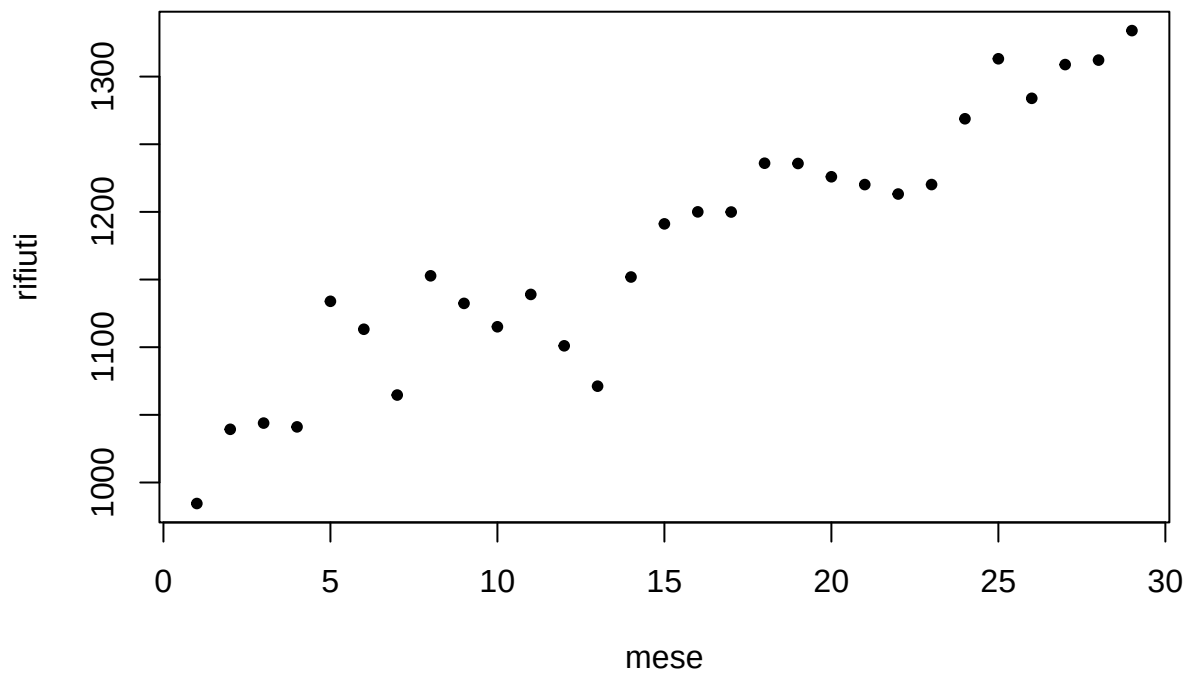
9.6 Exam exercise

```r
##### Problem 4 of 29/6/2011
#####-----------------------
# The file people.txt records the tons of waste collected monthly
# in the city of Santander since January 2009 (t = 1) until May 2011
# (t = 29). Assuming a model of the type:
#   Waste = A + B * t  + C * (1-cos(2pi / 12 * t)) + eps
# with eps ~ N(0, sigma^2) and identifying the contribution of the residents
# with the first two factors, and that of the tourists with the third
# addendum, answer the following questions.
# a) Estimate the parameters of the model.
# b) On the basis of the model (a), is there statistical evidence of an
#    increase attributable to residents?
```

```r
# c) On the basis of the model (a), is there statistical evidence of a
#    significant contribution by tourists?
# d) The University of Cantabria considered that the GROWTH attributable to
#    residents is quantifiable in an increase of 10 tons per month.
#    Can you deny this statement?
# e) Based on the test (b), (c) and (d) propose a possible reduced and/or
#    constrained model and estimate its parameters.
# f) On the basis of model (e), provide three pointwise forecasts for the
#    waste that will be collected in June 2011, for the waste that will be
#    collected in June 2011 due to residents and that which will be collected
#    in June 2011 due to the tourists.

people = read.table('Lab09/people.txt', header=T)
people
```

```
##    mese rifiuti
## 1     1  984.48
## 2     2 1039.32
## 3     3 1043.92
## 4     4 1041.11
## 5     5 1133.93
## 6     6 1113.26
## 7     7 1064.64
## 8     8 1152.77
## 9     9 1132.39
## 10   10 1115.10
## 11   11 1139.01
## 12   12 1101.04
## 13   13 1071.19
## 14   14 1151.87
## 15   15 1191.16
## 16   16 1200.03
## 17   17 1199.90
## 18   18 1235.97
## 19   19 1235.77
## 20   20 1225.97
## 21   21 1220.18
## 22   22 1213.21
## 23   23 1220.21
## 24   24 1268.82
## 25   25 1313.18
## 26   26 1283.94
## 27   27 1308.87
## 28   28 1312.18
## 29   29 1333.98
```

```r
plot(people,pch=20)
```

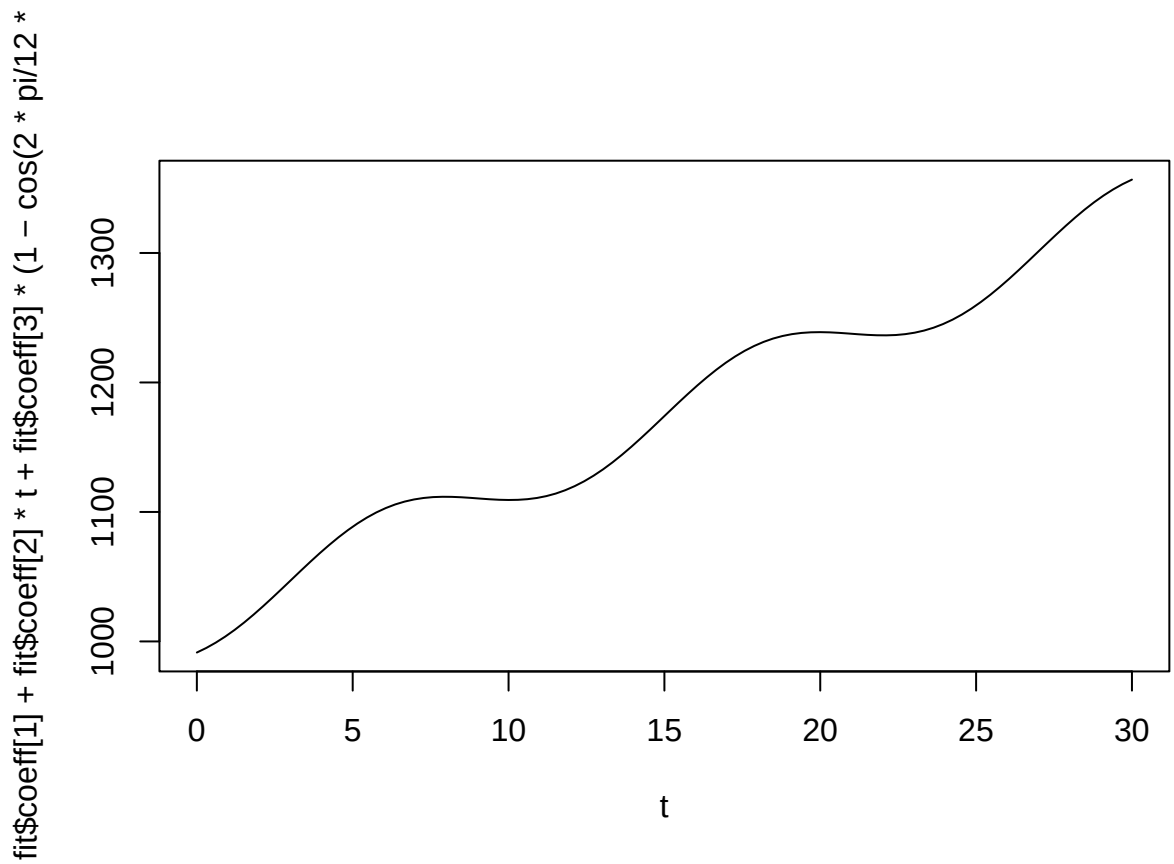


9.6.1 Qa

```
### question a)
# a) Estimate the parameters of the model.

attach(people)
fit = lm(rifiuti ~ mese + I(1 - cos(2*pi/12*mese)))
summary(fit)
```

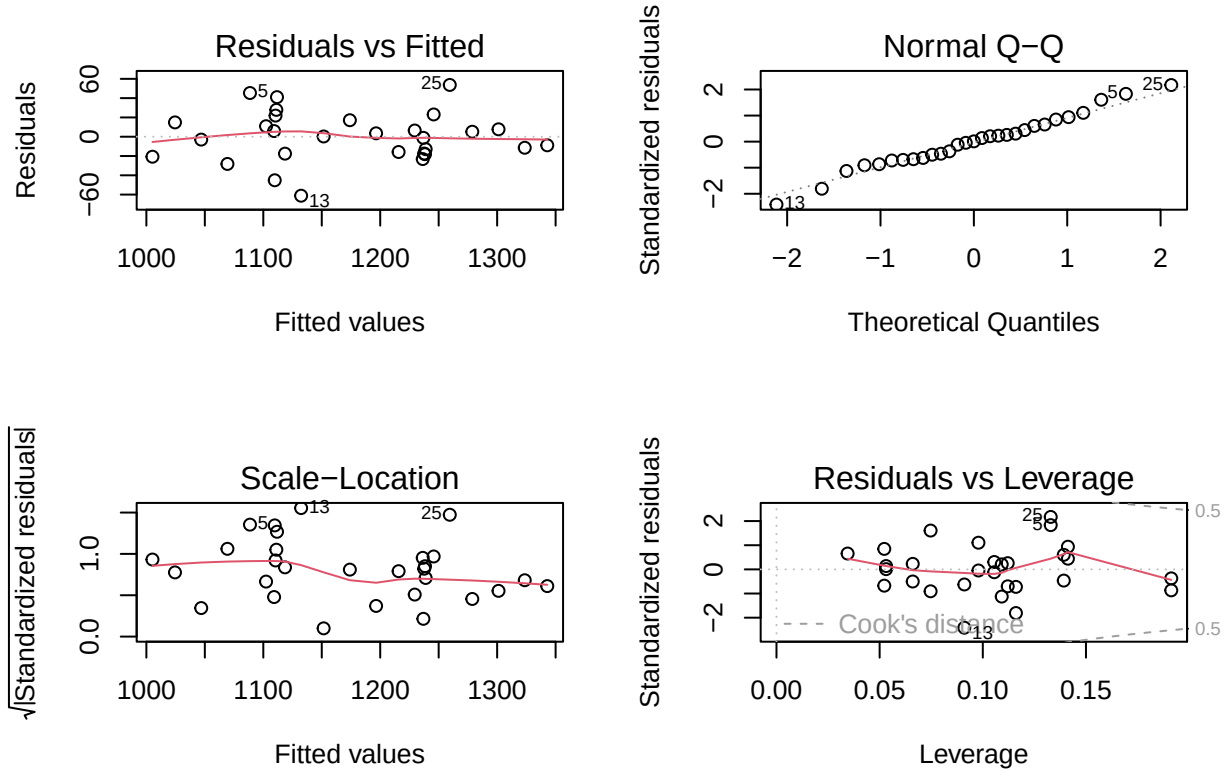
```
##
## Call:
## lm(formula = rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese)))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.162 -17.438   0.256  14.846  53.688
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   991.445     12.600  78.689  < 2e-16 ***
## mese                           10.595      0.590  17.957 3.57e-16 ***
## I(1 - cos(2 * pi/12 * mese))   23.678      7.105   3.333  0.00259 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.56 on 26 degrees of freedom
## Multiple R-squared:  0.9267, Adjusted R-squared:  0.9211
## F-statistic: 164.4 on 2 and 26 DF,  p-value: 1.753e-15
```

```
t = seq(from=0,to=30,length=100)
plot(t,fit$coeff[1]+fit$coeff[2]*t+fit$coeff[3]*(1-cos(2*pi/12*t)),type='l')
```



9.6.2 Qb

```
### question b)
# b) On the basis of the model (a), is there statistical evidence of an
#    increase attributable to residents?

par(mfrow=c(2,2))
plot(fit)
```




```
shapiro.test(residuals(fit))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fit)
## W = 0.98413, p-value = 0.9285
```

```
# Test: H0: beta_1==0 vs beta_1!=0
summary(fit)
```

```
##
## Call:
## lm(formula = rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese)))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.162 -17.438   0.256  14.846  53.688
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   991.445     12.600  78.689  < 2e-16 ***
## mese                           10.595      0.590  17.957 3.57e-16 ***
## I(1 - cos(2 * pi/12 * mese))   23.678      7.105   3.333  0.00259 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.56 on 26 degrees of freedom
## Multiple R-squared:  0.9267, Adjusted R-squared:  0.9211
## F-statistic: 164.4 on 2 and 26 DF,  p-value: 1.753e-15
```

```
## or
linearHypothesis(fit,rbind(c(0,1,0)),0)
```

```
## Linear hypothesis test
##
## Hypothesis:
```

```
## mese = 0
##
## Model 1: restricted model
## Model 2: rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese))
##
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1     27 245757
## 2     26  18337  1    227420 322.45 3.57e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.6.3 Qc

```r
### question c)
# c) On the basis of the model (a), is there statistical evidence of a
#    significant contribution by tourists?

# Test: H0: beta_2==0 vs beta_2!=0
summary(fit)
```

```
##
## Call:
## lm(formula = rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese)))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.162 -17.438   0.256  14.846  53.688
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   991.445     12.600  78.689  < 2e-16 ***
## mese                           10.595      0.590  17.957 3.57e-16 ***
## I(1 - cos(2 * pi/12 * mese))   23.678      7.105   3.333  0.00259 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.56 on 26 degrees of freedom
## Multiple R-squared:  0.9267, Adjusted R-squared:  0.9211
## F-statistic: 164.4 on 2 and 26 DF,  p-value: 1.753e-15
```

```r
## or
linearHypothesis(fit,rbind(c(0,0,1)),0)
```

```
## Linear hypothesis test
##
## Hypothesis:
## I(1 - cos(2 * pi/12 * mese)) = 0
##
## Model 1: restricted model
## Model 2: rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese))
##
##   Res.Df   RSS Df Sum of Sq      F   Pr(>F)
## 1     27 26171
## 2     26 18337  1    7833.7 11.107 0.002587 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.6.4 Qd

```r
### question d)
# d) The University of Cantabria considered that the GROWTH attributable to
#    residents is quantifiable in an increase of 10 tons per month.
#    Can you deny this statement?

linearHypothesis(fit,rbind(c(0,1,0)),10)
```

```
## Linear hypothesis test
##
## Hypothesis:
## mese = 10
```

```
##
## Model 1: restricted model
## Model 2: rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese))
##
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     27 19055
## 2     26 18337  1    717.21 1.0169 0.3225
```

```r
# or (from the summary)
summary(fit)
```

```
##
## Call:
## lm(formula = rifiuti ~ mese + I(1 - cos(2 * pi/12 * mese)))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.162 -17.438   0.256  14.846  53.688
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   991.445     12.600  78.689  < 2e-16 ***
## mese                           10.595      0.590  17.957 3.57e-16 ***
## I(1 - cos(2 * pi/12 * mese))   23.678      7.105   3.333  0.00259 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.56 on 26 degrees of freedom
## Multiple R-squared:  0.9267, Adjusted R-squared:  0.9211
## F-statistic: 164.4 on 2 and 26 DF,  p-value: 1.753e-15
```

```r
t = (coef(fit)[2]-10)/sqrt(diag(vcov(fit))[2])
t
```

```
##    mese
## 1.00842
```

```r
pval = 2*(1-pt(t,29-(2+1)))
pval
```

```
##      mese
## 0.3225463
```

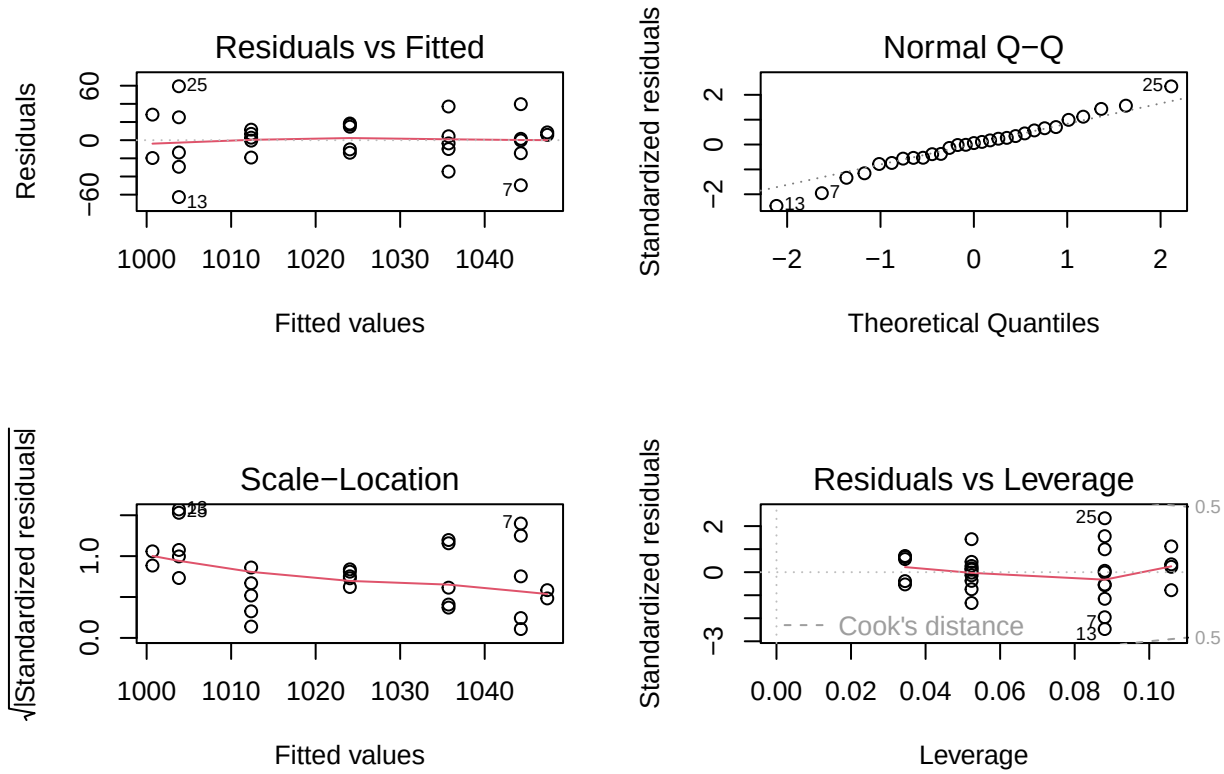
9.6.5 Qe

```r
### question e)
# e) Based on the test (b), (c) and (d) propose a possible reduced and/or
#    constrained model and estimate its parameters.

rifiuti.vinc = rifiuti - 10*mese

fit2 = lm(rifiuti.vinc ~ I(1 - cos(2*pi/12*mese)))
summary(fit2)
```

```
##
## Call:
## lm(formula = rifiuti.vinc ~ I(1 - cos(2 * pi/12 * mese)))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -62.629 -13.868   1.493  14.822  59.361
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  1000.690      8.645  115.75  < 2e-16 ***
## I(1 - cos(2 * pi/12 * mese))   23.358      7.100    3.29  0.00279 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.57 on 27 degrees of freedom
## Multiple R-squared:  0.2862, Adjusted R-squared:  0.2597
## F-statistic: 10.82 on 1 and 27 DF,  p-value: 0.002791
```

9.6.6 Qf

```
### question f)
# f) On the basis of model (e), provide three pointwise forecasts for the
#    waste that will be collected in June 2011, for the waste that will be
#    collected in June 2011 due to residents and that which will be collected
#    in June 2011 due to the tourists.

par(mfrow=c(2,2))
plot(fit2)
```




```
shapiro.test(residuals(fit2))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fit2)
## W = 0.98643, p-value = 0.9631
```

```
coefficients(fit2)
```

```
##                  (Intercept) I(1 - cos(2 * pi/12 * mese))
##                   1000.69006                     23.35821
```

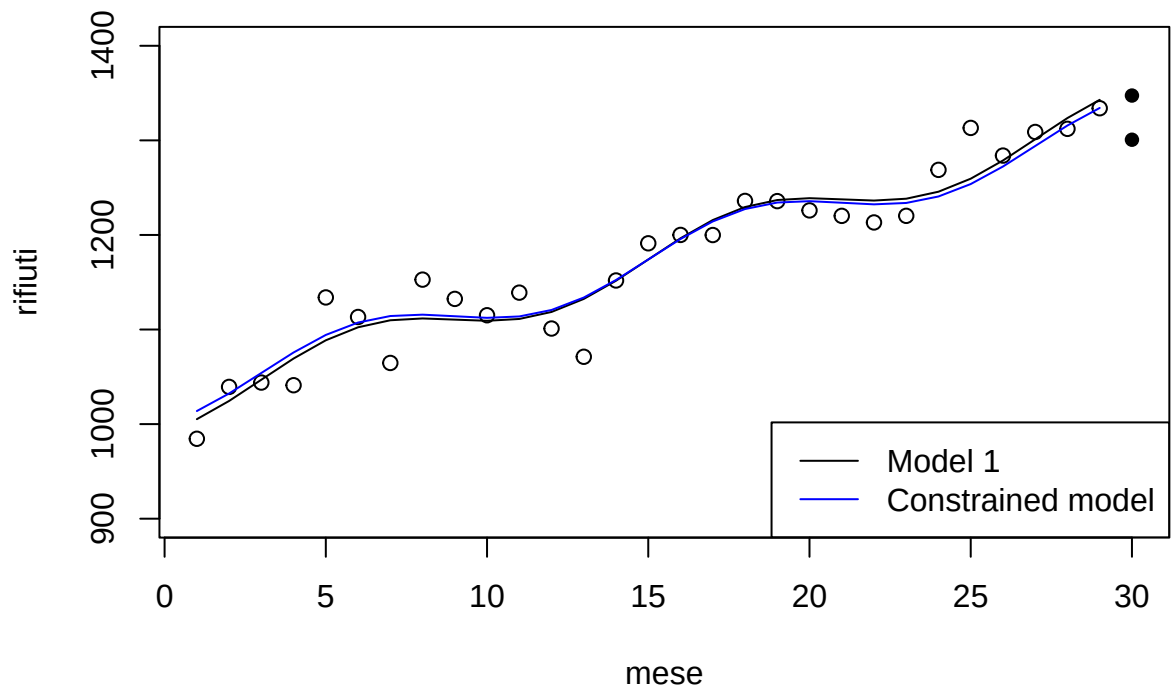
```
C = rbind(c(1,(1 - cos(2*pi/12*30))),   # total waste in June 2011 [mese=30]
          c(1,0),                       # waste due to residents in June 2011
          c(0,(1 - cos(2*pi/12*30))))   # waste due to tourists in June 2011
C
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    0
## [3,]    0    2
```

```r
pred = C %*% coefficients(fit2) + c(10*30, 10*30, 0)
# pred = C %*% beta.hat[fit.mod.constrained] + 10*mese[constrained part]
pred
```

```
##            [,1]
## [1,] 1347.40649
## [2,] 1300.69006
## [3,]   46.71643
```

```r
plot(people, xlim=c(1,30), ylim=c(900,1400))
lines(people$mese, fitted(fit))
lines(people$mese, fitted(fit2) + 10*people$mese, col='blue')
points(c(30,30,30), pred, pch=16)
legend('bottomright',c('Model 1', 'Constrained model'), lty=1, col=c('black','blue'))
```



9.7 Linear regression with categorical data

```r
#_____________________________________________________________________________
##### Multiple linear regression with qualitative predictors
#####-------------------------------------------------------

data = read.table('Lab09/work.txt', header=T)
head(data)
```

```
##   Employee Average_Score Years_Service    Sex     Race
## 1        1           7.6             5 Female Nonwhite
## 2        2           9.0            30 Female Nonwhite
## 3        3           8.0            12 Female Nonwhite
## 4        4           6.8             7 Female Nonwhite
## 5        5           7.4             7 Female Nonwhite
## 6        6           9.8            27 Female    White
```

```r
dim(data)
```

```
## [1] 20  5
```

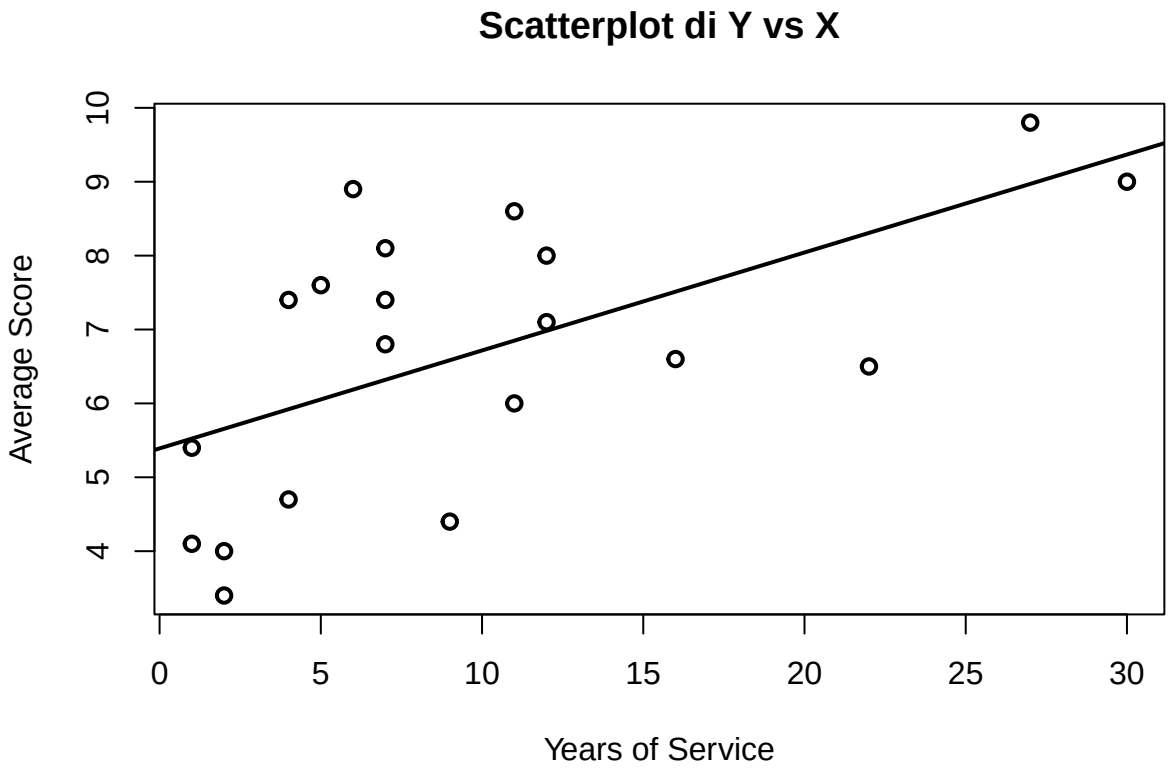
```r
n = dim(data)[1]

names(data)
```

```
## [1] "Employee"      "Average_Score" "Years_Service" "Sex"
## [5] "Race"
```

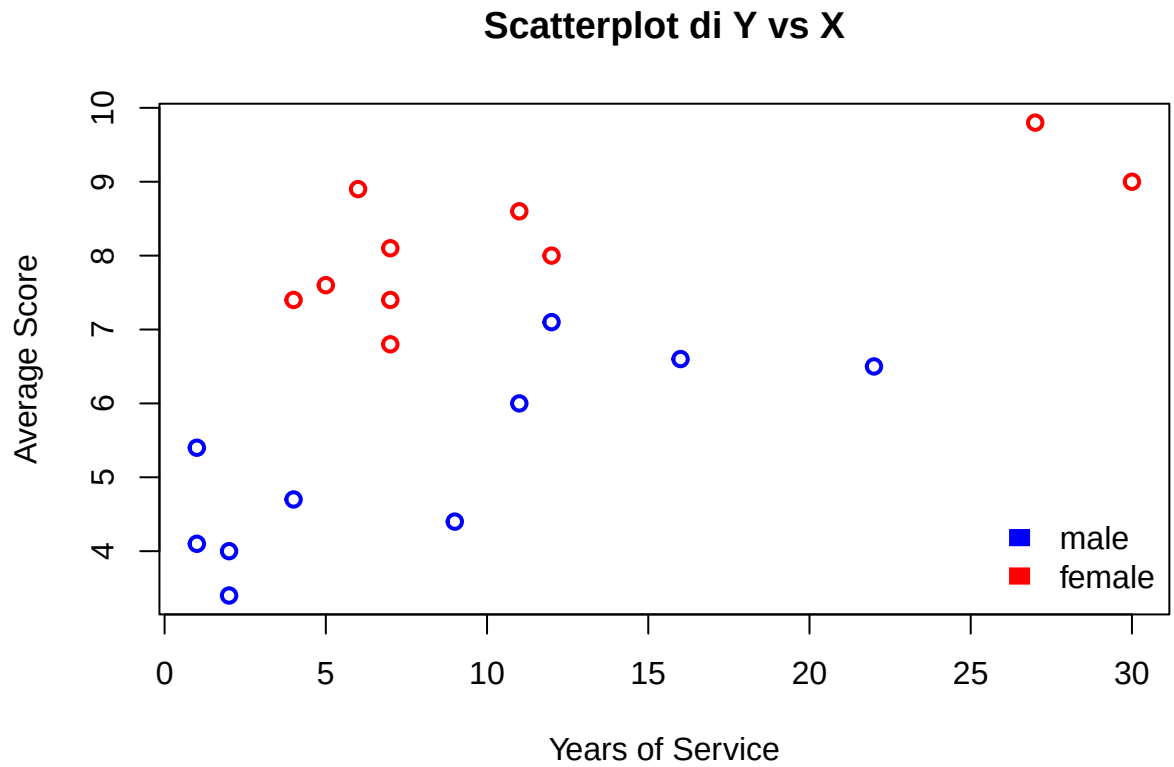
```r
attach(data)

Y = Average_Score
X = Years_Service
C1 = Sex
C2 = Race

detach(data)

plot(X, Y, main='Scatterplot di Y vs X', lwd=2,
     xlab='Years of Service', ylab='Average Score')

result = lm(Y ~ X)
summary(result)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.25596 -1.27167 -0.00252  1.18062  2.71376
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.39082    0.53911   9.999 8.94e-09 ***
## X            0.13257    0.04242   3.125  0.00585 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.535 on 18 degrees of freedom
## Multiple R-squared:  0.3517, Adjusted R-squared:  0.3157
## F-statistic: 9.766 on 1 and 18 DF,  p-value: 0.00585
```

```r
coef = result$coef
abline(coef[1],coef[2],lwd=2)
```




```
# differences between males and females:
col = rep('blue',n)
females = which(C1=='Female')
males = which(C1=='Male')
col[females] = 'red'

plot(X, Y, main='Scatterplot di Y vs X', lwd=2,
     xlab='Years of Service', ylab='Average Score', col = col)
legend("bottomright",c("male","female"),fill=c("blue","red"), bty="n",border="white")
```



9.7.1 With one qualitative predictor

```
### Multiple linear regression with one qualitative predictor

C1.new = rep(0,n)
C1.new[males] = 1

result1 = lm(Y ~  X + C1.new + X:C1.new)
summary(result1)
```

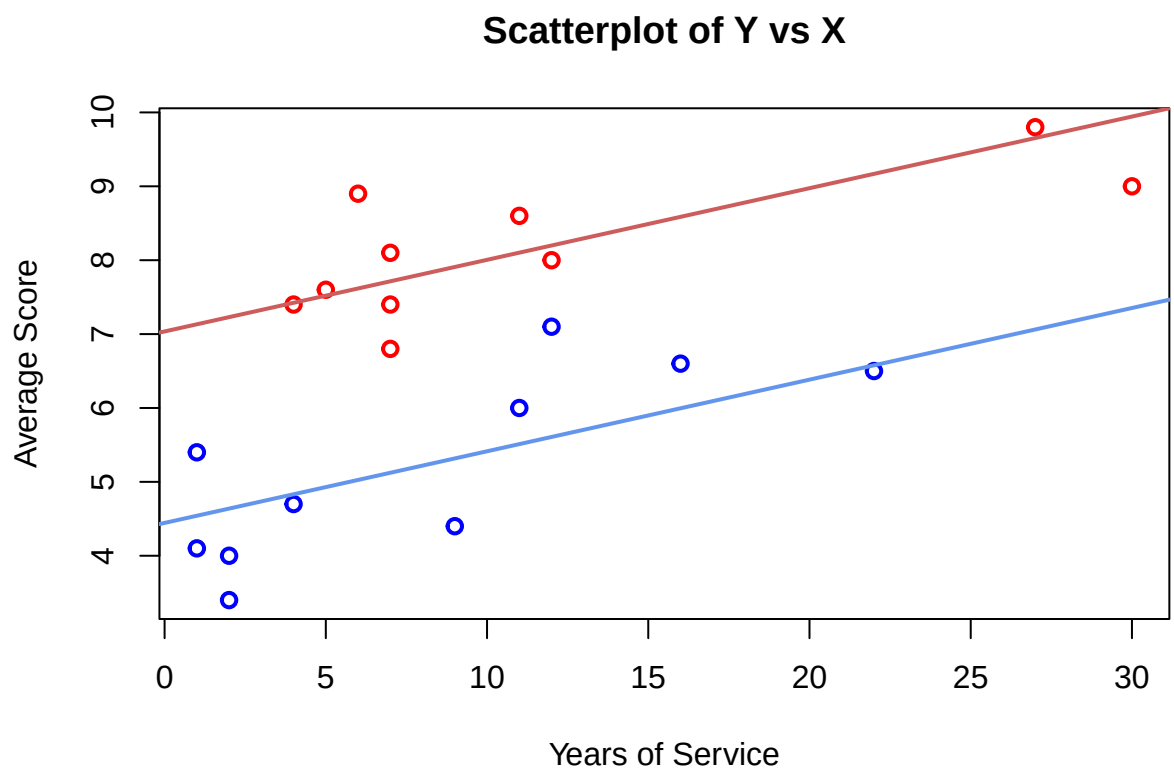
```
##
## Call:
## lm(formula = Y ~ X + C1.new + X:C1.new)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0280 -0.4430 -0.1206  0.3965  1.3300
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.32289    0.39824  18.388 3.48e-12 ***
## X            0.07216    0.02736   2.637   0.0179 *
## C1.new      -3.20289    0.54300  -5.898 2.25e-05 ***
## X:C1.new     0.06534    0.04443   1.470   0.1608
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7605 on 16 degrees of freedom
## Multiple R-squared:  0.8585, Adjusted R-squared:  0.832
## F-statistic: 32.37 on 3 and 16 DF,  p-value: 5.004e-07
```

```
result2 = lm(Y ~ C1.new + X)
summary(result2)
```

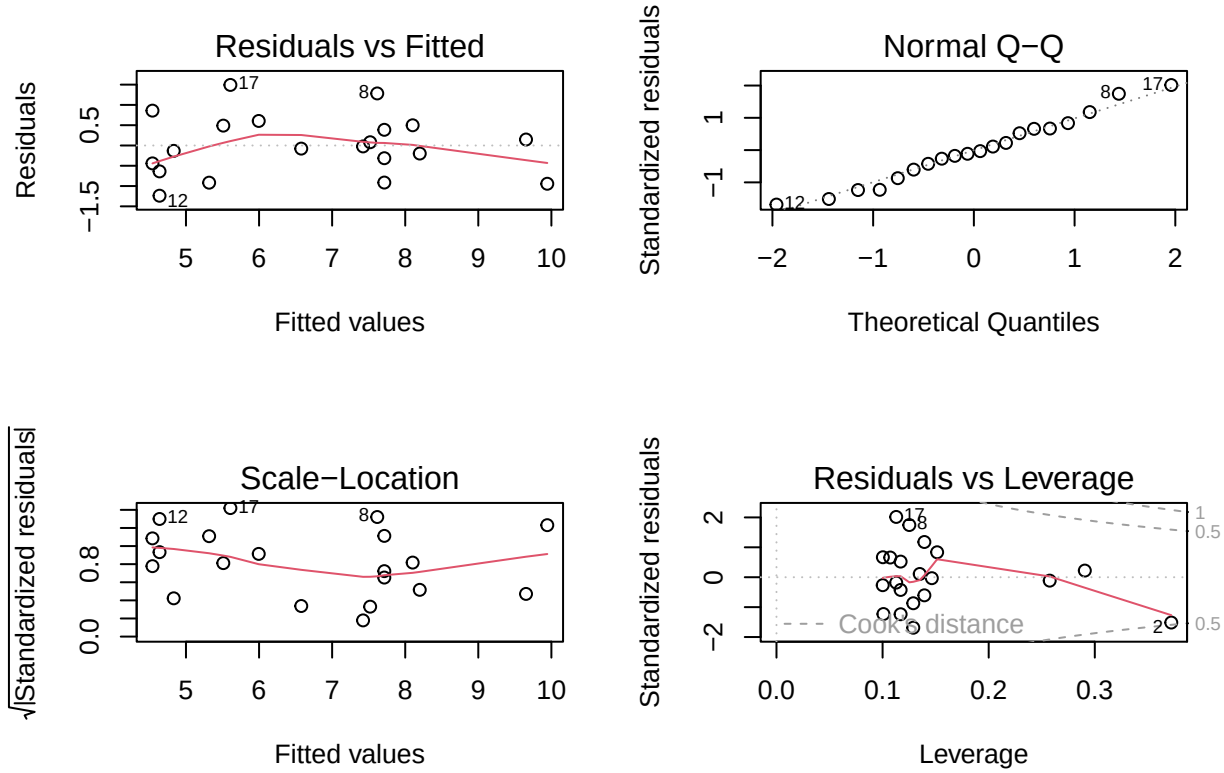
```
##
## Call:
## lm(formula = Y ~ C1.new + X)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.23832 -0.49061 -0.05023  0.49141  1.49221
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.03542    0.35862  19.618 4.10e-13 ***
## C1.new      -2.59099    0.36058  -7.186 1.52e-06 ***
## X            0.09695    0.02228   4.351 0.000435 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7861 on 17 degrees of freedom
## Multiple R-squared:  0.8394, Adjusted R-squared:  0.8205
## F-statistic: 44.44 on 2 and 17 DF,  p-value: 1.771e-07
```

```r
# interpretation of the model:
# modell for females: Y = 7.035 + 0.097 X
# modell males:       Y = 7.035 - 2.59 + 0.097 X = 4.44 + 0.097 X

plot(X, Y, main='Scatterplot of Y vs X', lwd=2,
     xlab='Years of Service', ylab='Average Score', col = col)

coef = result2$coef
abline(coef[1],coef[3],lwd=2,col='indianred')
abline(coef[1]+coef[2],coef[3],lwd=2,col='cornflowerblue')
```




```r
# diagnostics of the residuals
par(mfrow=c(2,2))
plot(result2)
```




```
shapiro.test(residuals(result2))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(result2)
## W = 0.9752, p-value = 0.8584
```

9.7.2 With two qualitative predictor

```
### Multiple linear regression with two qualitative predictors
# qualitative predictors:
C1
```

```
##  [1] "Female" "Female" "Female" "Female" "Female" "Female" "Female" "Female"
##  [9] "Female" "Female" "Male"   "Male"   "Male"   "Male"   "Male"   "Male"
## [17] "Male"   "Male"   "Male"   "Male"
```

```
C2
```

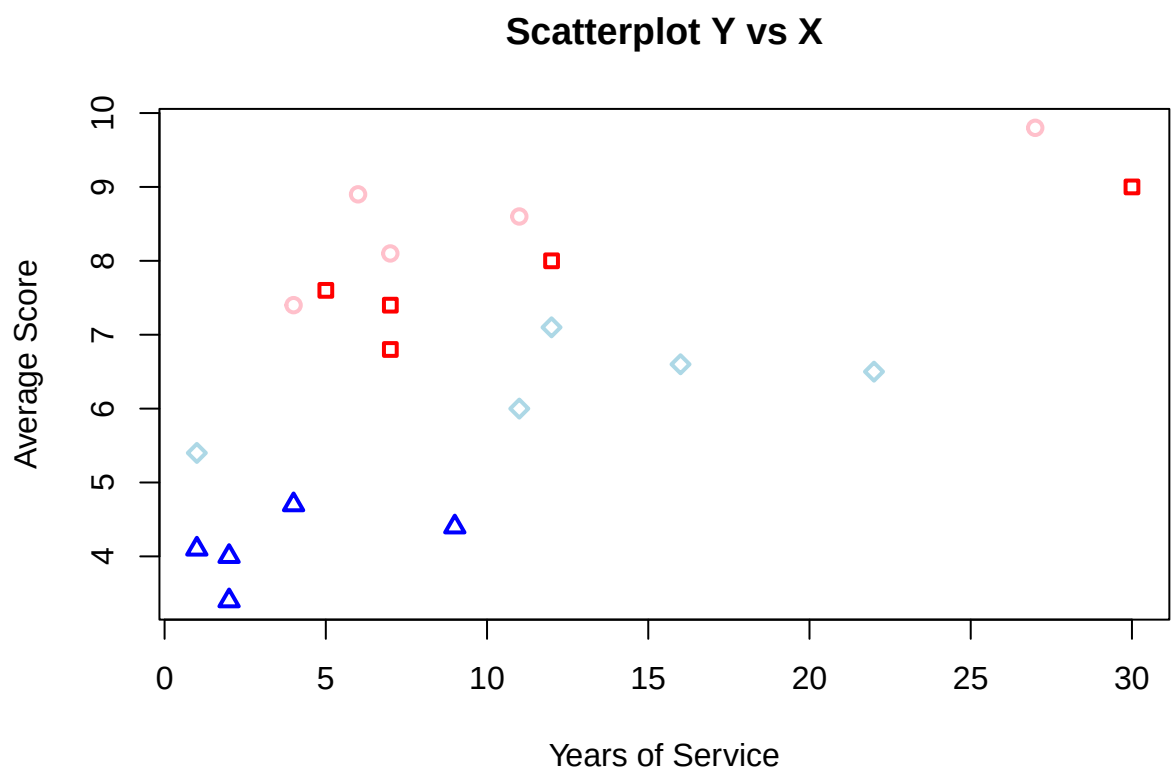
```
##  [1] "Nonwhite" "Nonwhite" "Nonwhite" "Nonwhite" "Nonwhite" "White"
##  [7] "White"    "White"    "White"    "White"    "Nonwhite" "Nonwhite"
## [13] "Nonwhite" "Nonwhite" "Nonwhite" "White"    "White"    "White"
## [19] "White"    "White"
```

```
C1.new
```

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

```
notwhite = which(C2=='Nonwhite')
white = which(C2=='White')
C2.new = rep(0,n)
C2.new[notwhite] = 0
C2.new[white] = 1

# 4 cases:
```

```r
# females white
FB = which(C1.new==0 & C2.new==1)
# females not white
FNB = which(C1.new==0 & C2.new==0)
# males white
MB = which(C1.new==1 & C2.new==1)
# males not white
MNB = which(C1.new==1 & C2.new==0)

# colors for the plot
col = rep(NA,n)
col[FB] = 'pink'
col[FNB] = 'red'
col[MB] = 'light blue'
col[MNB] = 'blue'
  # shape of the dots for the plot
shape = rep(0,n)
shape[FB] = 21
shape[FNB] = 22
shape[MB] = 23
shape[MNB] = 24

plot(X, Y, main='Scatterplot Y vs X', lwd=2,
     xlab='Years of Service', ylab='Average Score', col = col, pch = shape)
```




```r
result3 = lm(Y ~ X + C1.new + C2.new + X:C1.new + X:C2.new)
result3
```

```
##
## Call:
## lm(formula = Y ~ X + C1.new + C2.new + X:C1.new + X:C2.new)
##
## Coefficients:
## (Intercept)            X       C1.new       C2.new     X:C1.new     X:C2.new
##    6.621494     0.082729    -2.700704     1.306699     0.008501    -0.013545
```

```r
summary(result3)
```

```
##
## Call:
## lm(formula = Y ~ X + C1.new + C2.new + X:C1.new + X:C2.new)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.80493 -0.31983 -0.03909  0.24599  0.94029
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.621494   0.328461  20.159 9.66e-12 ***
## X            0.082729   0.022993   3.598  0.00291 **
## C1.new      -2.700704   0.380441  -7.099 5.35e-06 ***
## C2.new       1.306699   0.382630   3.415  0.00419 **
## X:C1.new     0.008501   0.033229   0.256  0.80180
## X:C2.new    -0.013545   0.031301  -0.433  0.67180
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5107 on 14 degrees of freedom
## Multiple R-squared:  0.9442, Adjusted R-squared:  0.9243
## F-statistic: 47.37 on 5 and 14 DF,  p-value: 2.785e-08
```

```r
result4 = lm(Y ~ X + C1.new + C2.new + X:C2.new)
result4
```

```
##
## Call:
## lm(formula = Y ~ X + C1.new + C2.new + X:C2.new)
##
## Coefficients:
## (Intercept)            X       C1.new       C2.new     X:C2.new
##     6.58419      0.08471     -2.62673      1.31742     -0.01191
```

```r
summary(result4)
```

```
##
## Call:
## lm(formula = Y ~ X + C1.new + C2.new + X:C2.new)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.79280 -0.31336 -0.07144  0.26699  0.95152
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.58419    0.28500  23.102 3.87e-13 ***
## X            0.08471    0.02097   4.039  0.00107 **
## C1.new      -2.62673    0.23940 -10.972 1.45e-08 ***
## C2.new       1.31742    0.36829   3.577  0.00275 **
## X:C2.new    -0.01191    0.02967  -0.401  0.69386
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4945 on 15 degrees of freedom
## Multiple R-squared:  0.9439, Adjusted R-squared:  0.929
## F-statistic: 63.13 on 4 and 15 DF,  p-value: 3.333e-09
```

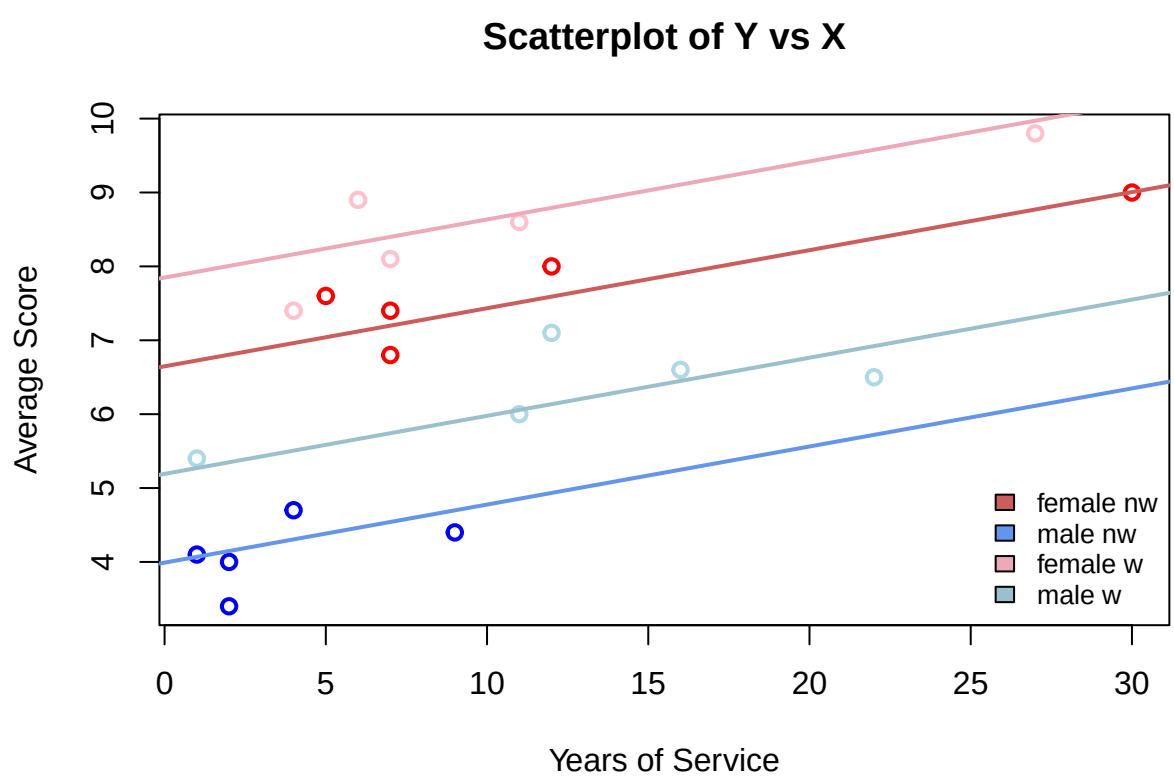
```r
result5 = lm(Y ~ X + C1.new + C2.new)
result5
```

```
##
## Call:
## lm(formula = Y ~ X + C1.new + C2.new)
##
## Coefficients:
## (Intercept)            X       C1.new       C2.new
##      6.6475       0.0786      -2.6570       1.2013
```

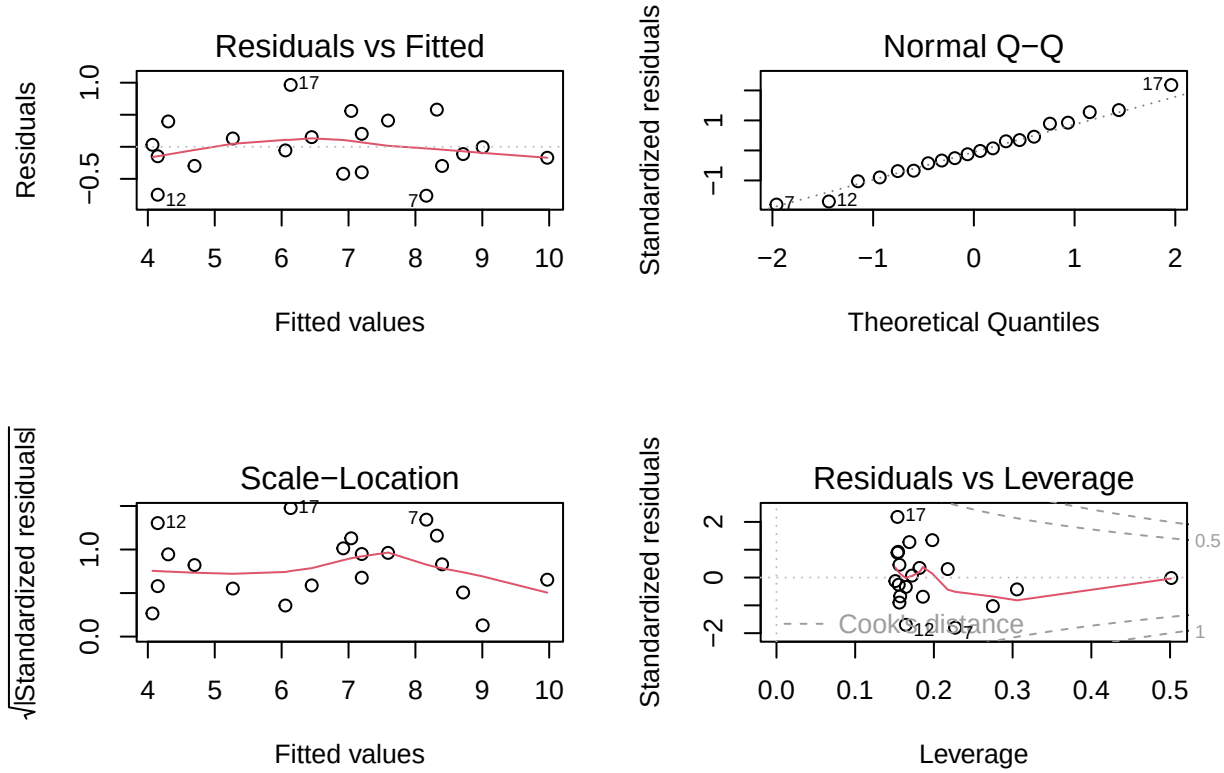
```r
summary(result5)
```

```
##
## Call:
## lm(formula = Y ~ X + C1.new + C2.new)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.76326 -0.29823 -0.03107  0.25044  0.96493
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.64754    0.23099  28.779 3.30e-15 ***
## X            0.07860    0.01406   5.591 4.06e-05 ***
## C1.new      -2.65702    0.22114 -12.015 2.02e-09 ***
## C2.new       1.20130    0.22180   5.416 5.71e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4814 on 16 degrees of freedom
## Multiple R-squared:  0.9433, Adjusted R-squared:  0.9327
## F-statistic: 88.77 on 3 and 16 DF,  p-value: 3.462e-10
```

```r
plot(X, Y, main='Scatterplot of Y vs X', lwd=2,
     xlab='Years of Service', ylab='Average Score', col = col)
legend("bottomright",c("female nw","male nw","female w","male w"),
       fill=c('indianred',"cornflowerblue","pink2","lightblue3"),bty="n", cex=0.8)

coef = result5$coef
abline(coef[1],coef[2],lwd=2,col='indianred')                  # female, nonwhite
abline(coef[1]+coef[3],coef[2],lwd=2,col='cornflowerblue')     # male, nonwhite
abline(coef[1]+coef[4],coef[2],lwd=2,col='pink2')              # female, white
abline(coef[1]+coef[3]+coef[4],coef[2],lwd=2,col='lightblue3') # male, white
```




```r
# diagnostics of the residuals
par(mfrow=c(2,2))
plot(result5)
```




```
shapiro.test(residuals(result5))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(result5)
## W = 0.98092, p-value = 0.9454
```

10 Lab 10 - Linear models

```
###---------------------###
### LAB 10 (16/05/2023) ###
###---------------------###

### TOPICS:
### Linear models

library(MASS)
library(car)
library(rgl)
library(glmnet)
```

```
## Loaded glmnet 4.1-7
```

```
# options(rgl.printRglwidget = TRUE)
# windows.options(height=6.5, width=11, reset=FALSE)
```

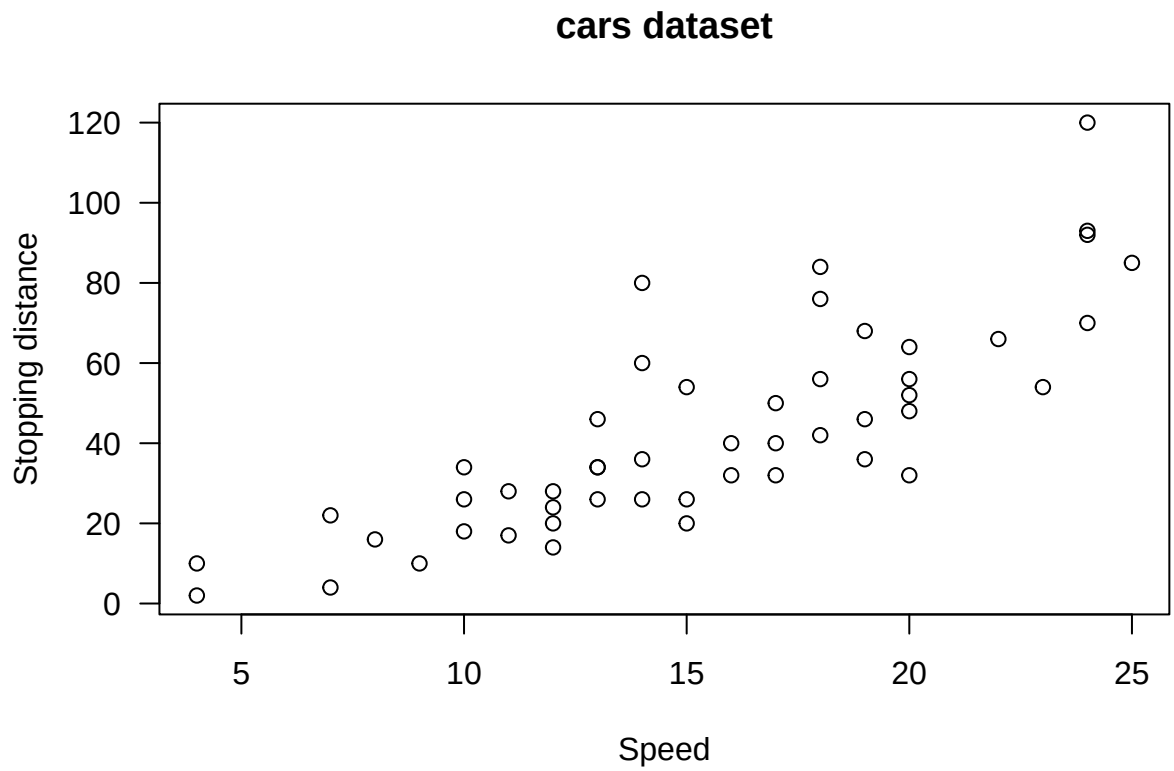
10.1 Problem of collinearity

```
#_____________________________________________________________________________
##### Problem of collinearity
#####--------------------------
#! Collinearity causes the estimates of our coefficients beta to have large variances
```

```r
### Example 1: Multiple linear regression
###----------------------------------------
### Dataset cars: distance taken to stop [ft] as a function of velocity [mph]
### for some cars in the 1920s

cars
```

```
##    speed dist
## 1      4    2
## 2      4   10
## 3      7    4
## 4      7   22
## 5      8   16
## 6      9   10
## 7     10   18
## 8     10   26
## 9     10   34
## 10    11   17
## 11    11   28
## 12    12   14
## 13    12   20
## 14    12   24
## 15    12   28
## 16    13   26
## 17    13   34
## 18    13   34
## 19    13   46
## 20    14   26
## 21    14   36
## 22    14   60
## 23    14   80
## 24    15   20
## 25    15   26
## 26    15   54
## 27    16   32
## 28    16   40
## 29    17   32
## 30    17   40
## 31    17   50
## 32    18   42
## 33    18   56
## 34    18   76
## 35    18   84
## 36    19   36
## 37    19   46
## 38    19   68
## 39    20   32
## 40    20   48
## 41    20   52
## 42    20   56
## 43    20   64
## 44    22   66
## 45    23   54
## 46    24   70
## 47    24   92
## 48    24   93
## 49    24  120
## 50    25   85
```

```r
plot(cars, xlab='Speed', ylab='Stopping distance', las=1, main="cars dataset")
```




```
n           = dim(cars)[[1]]
distance    = cars$dist
speed1      = cars$speed
speed2      = cars$speed^2

### Model:
### distance = beta_0 + beta_1 * speed + beta_2 * speed^2 + Eps
### (linear in the parameters!)

fm = lm(distance ~ speed1 + speed2)
summary(fm)
```

```
##
## Call:
## lm(formula = distance ~ speed1 + speed2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.720  -9.184  -3.188   4.628  45.152
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47014   14.81716   0.167    0.868
## speed1       0.91329    2.03422   0.449    0.656
## speed2       0.09996    0.06597   1.515    0.136
##
## Residual standard error: 15.18 on 47 degrees of freedom
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12
```

```
# Note: collinearity
#! As we have significance on the whole model (high R^2) but low significance
#! on the singular components (high pvalue)

# Variance inflation factor
# help(vif)

vif(fm)
```

```
##   speed1   speed2
## 24.61489 24.61489
```

```r
# Rule of thumb -> problem when VIF exceeds 10 (or 5 sometimes)
#! vif returns the R^2 obtained regressing a col with all the others; so good
#! for detecting collinearity, to see if one col can alredy be explained by others

# recall vif formula:
1/(1- summary(lm(speed1 ~ speed2))$r.squared )
```

```
## [1] 24.61489
```

```r
1/(1- summary(lm(speed2 ~ speed1))$r.squared )
```

```
## [1] 24.61489
```

10.1.1 Slt through PCA

```r
### A possible solution to colinearity: PCA
speed.pc = princomp(cbind(speed1,speed2), scores=TRUE)
summary(speed.pc)
```

```
## Importance of components:
##                             Comp.1       Comp.2
## Standard deviation     161.4944407 1.054526e+00
## Proportion of Variance   0.9999574 4.263644e-05
## Cumulative Proportion    0.9999574 1.000000e+00
```

```r
speed.pc$load
```

```
##
## Loadings:
##        Comp.1 Comp.2
## speed1         0.999
## speed2  0.999
##
##                Comp.1 Comp.2
## SS loadings       1.0    1.0
## Proportion Var    0.5    0.5
## Cumulative Var    0.5    1.0
```

```r
#! From the loadings we see that PC1 is morally just speed2

sp1.pc = speed.pc$scores[,1]
sp2.pc = speed.pc$scores[,2]

# Now we estimate the model by inserting the PCs instead of the original regressors
# Model: y = b0 + b1*PC1+ b2*PC2 + eps, eps~N(0,sigma^2)
fm.pc = lm(distance ~ sp1.pc + sp2.pc)

summary(fm.pc)
```

```
##
## Call:
## lm(formula = distance ~ sp1.pc + sp2.pc)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.720  -9.184  -3.188   4.628  45.152
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.98000    2.14622  20.026  < 2e-16 ***
## sp1.pc       0.12890    0.01329   9.700 8.49e-13 ***
## sp2.pc       0.90965    2.03525   0.447    0.657
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.18 on 47 degrees of freedom
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12
```

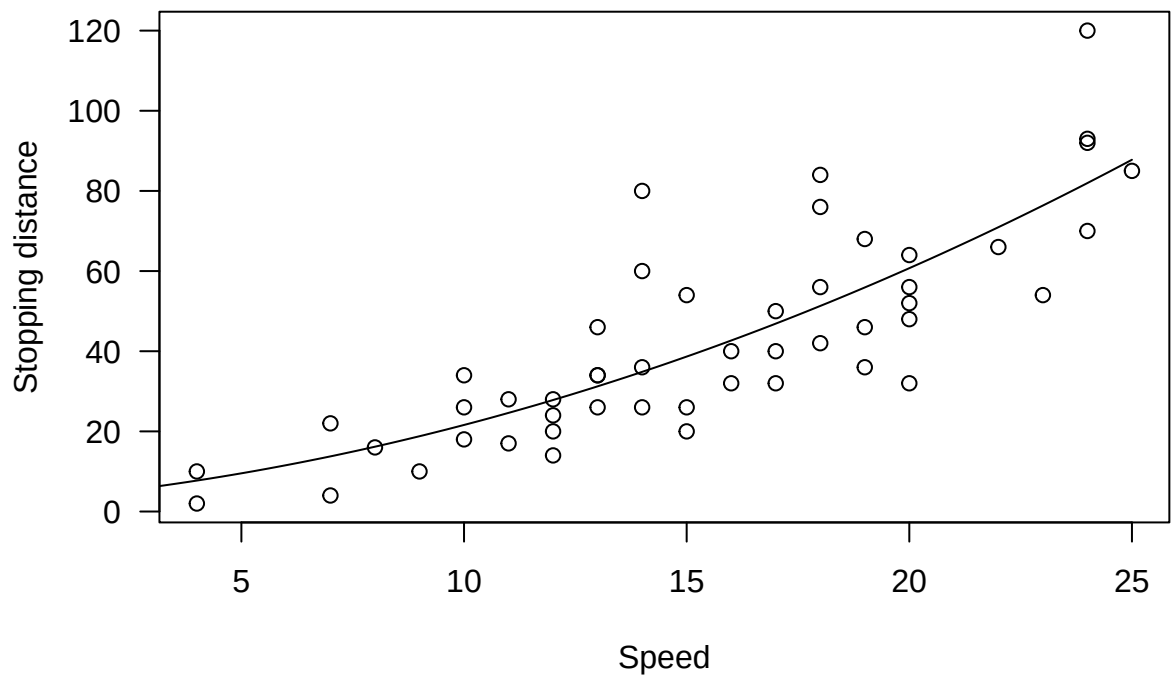
```r
# Note: we could have performed dimensionality reduction before
# estimating the model and then considered only the first PC.

# We can re-write the model as:
# Model:
# y= b0 + b1*      PC1                 + b2*      PC2                 + eps =
#  = b0 + b1*(e11*(Alluminium-m1)+e21*(Silicate-m2)) + b2*(e12*(Alluminium-m1) +
#    + e22*(Silicate-m2)) + eps =
#  = b0 - b1*e11*m1 - b2*e12*m1 - b1*e21*m2 - b2*e22*m2 +
#    + (b1*e11+b2*e12)*Alluminium + (b1*e21+b2*e22)*Silicate + eps
# where e.ij are the loadings, i=1,2, j=1,2.
# => We can compute the coefficients of the model which used the original regressors
#
m1 = mean(speed1)
m2 = mean(speed2)
beta0 = coefficients(fm.pc)[1] -
  coefficients(fm.pc)[2]*speed.pc$load[1,1]*m1 -
  coefficients(fm.pc)[3]*speed.pc$load[1,2]*m1 -
  coefficients(fm.pc)[2]*speed.pc$load[2,1]*m2 -
  coefficients(fm.pc)[3]*speed.pc$load[2,2]*m2
beta1 = coefficients(fm.pc)[2]*speed.pc$load[1,1] +
  coefficients(fm.pc)[3]*speed.pc$load[1,2]
beta2 = coefficients(fm.pc)[2]*speed.pc$load[2,1] +
  coefficients(fm.pc)[3]*speed.pc$load[2,2]

c(beta0=as.numeric(beta0),beta1=as.numeric(beta1),beta2=as.numeric(beta2))
```

```
##     beta0     beta1     beta2
## 2.4701378 0.9132876 0.0999593
```

```r
fm$coefficients
```

```
## (Intercept)      speed1      speed2
##   2.4701378   0.9132876   0.0999593
```

```r
#! We are looking at the same data, just rotating the coordinate system,
#! with the lm with the PC

x = seq(0, 25, len=100)
plot(cars, xlab='Speed', ylab='Stopping distance', las=1)
lines(x, beta0+beta1*x+beta2*x^2)
```




```r
# Reduce the model:
#! easy doable now with the PCs
fm.pc = lm(distance ~ sp1.pc)
summary(fm.pc)
```

10.1.1.1 Model reduction

```
##
## Call:
## lm(formula = distance ~ sp1.pc)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.449  -9.205  -3.595   5.074  45.859
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.98000    2.12825  20.195  < 2e-16 ***
## sp1.pc       0.12890    0.01318   9.781  5.2e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.05 on 48 degrees of freedom
## Multiple R-squared:  0.6659, Adjusted R-squared:  0.659
## F-statistic: 95.68 on 1 and 48 DF,  p-value: 5.195e-13
```

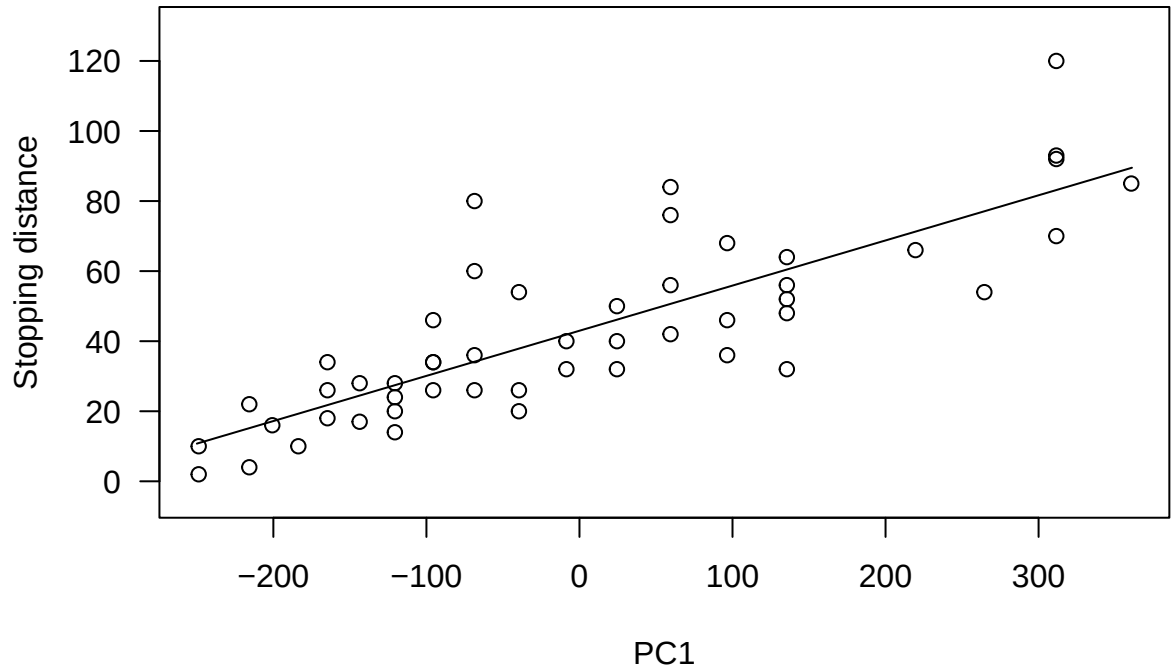
```r
# We can re-write the model as:
# Model: y = b0 + b1*      PC1                  + eps =
#          = b0 + b1*(e11*(Alluminium-m1)1+e21*(Silicate-m2)) + eps =
#          = b0 - b1*e11*m1 - b2*e21*m2 + b1*e11*Alluminium + b1*e21*Silicate + eps
beta0 = coefficients(fm.pc)[1] -
  coefficients(fm.pc)[2]*speed.pc$load[1,1]*m1 -
  coefficients(fm.pc)[2]*speed.pc$load[2,1]*m2
beta1 = coefficients(fm.pc)[2]*speed.pc$load[1,1]
beta2 = coefficients(fm.pc)[2]*speed.pc$load[2,1]

c(beta0=as.numeric(beta0),beta1=as.numeric(beta1),beta2=as.numeric(beta2))
```

```
##       beta0       beta1       beta2
## 8.831088116 0.004092603 0.128839907
```

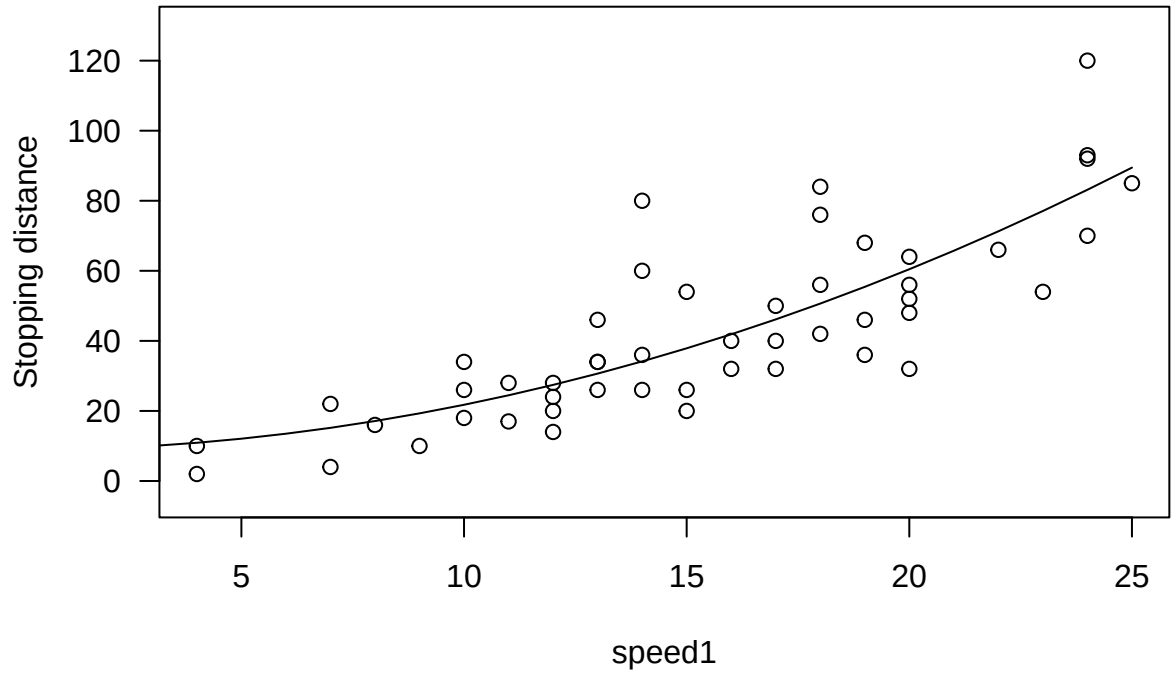
```
fm$coefficients
```

```
## (Intercept)      speed1      speed2
##   2.4701378   0.9132876   0.0999593
```

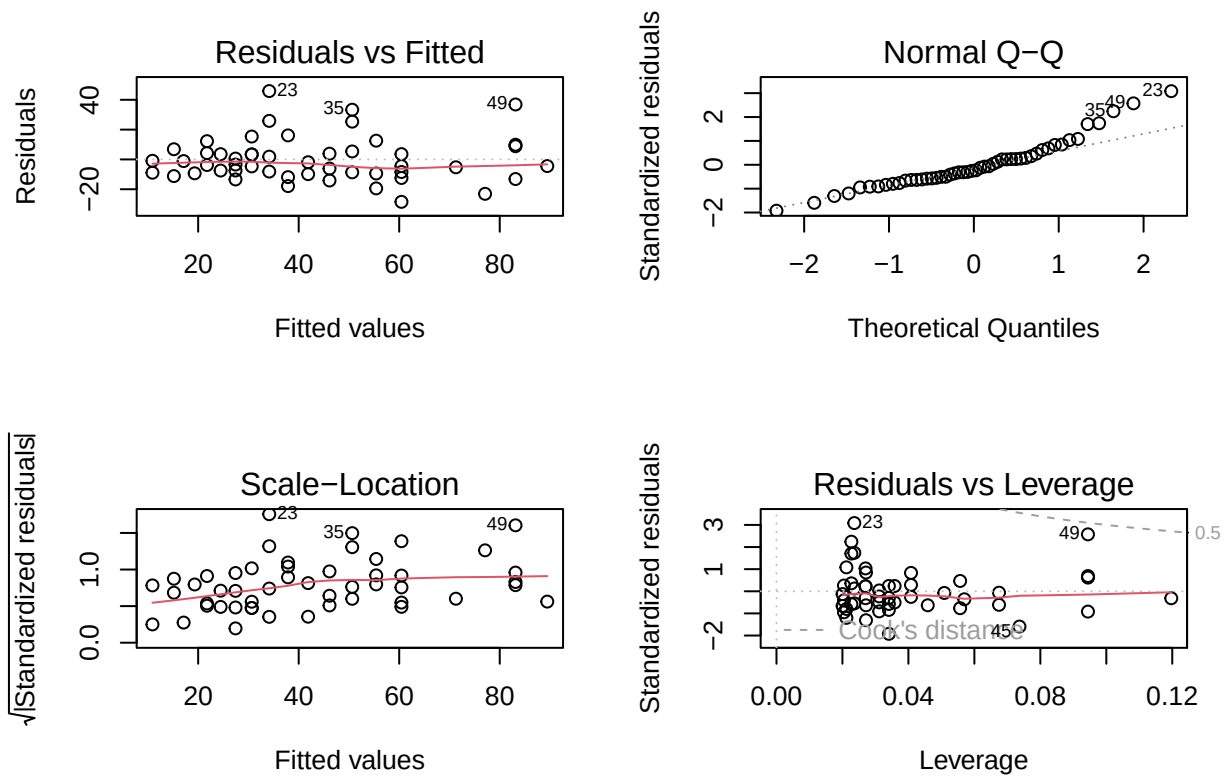
```
#! Now are different as we reduced the model, so we now are working on different data

plot(sp1.pc,distance, xlab='PC1', ylab='Stopping distance', las=1,
     xlim=c(-250,361), ylim=c(-5,130))
x = seq(-250,361,by=1)
b = coef(fm.pc)
lines(x, b[1]+b[2]*x)
```




```
plot(speed1,distance, ylab='Stopping distance', las=1, ylim=c(-5,130))
x = seq(0,25,by=1)
lines(x, beta0 + beta1*x + beta2*x^2)
```




```
# diagnostics of the residuals
par(mfrow=c(2,2))
plot(fm.pc)
```



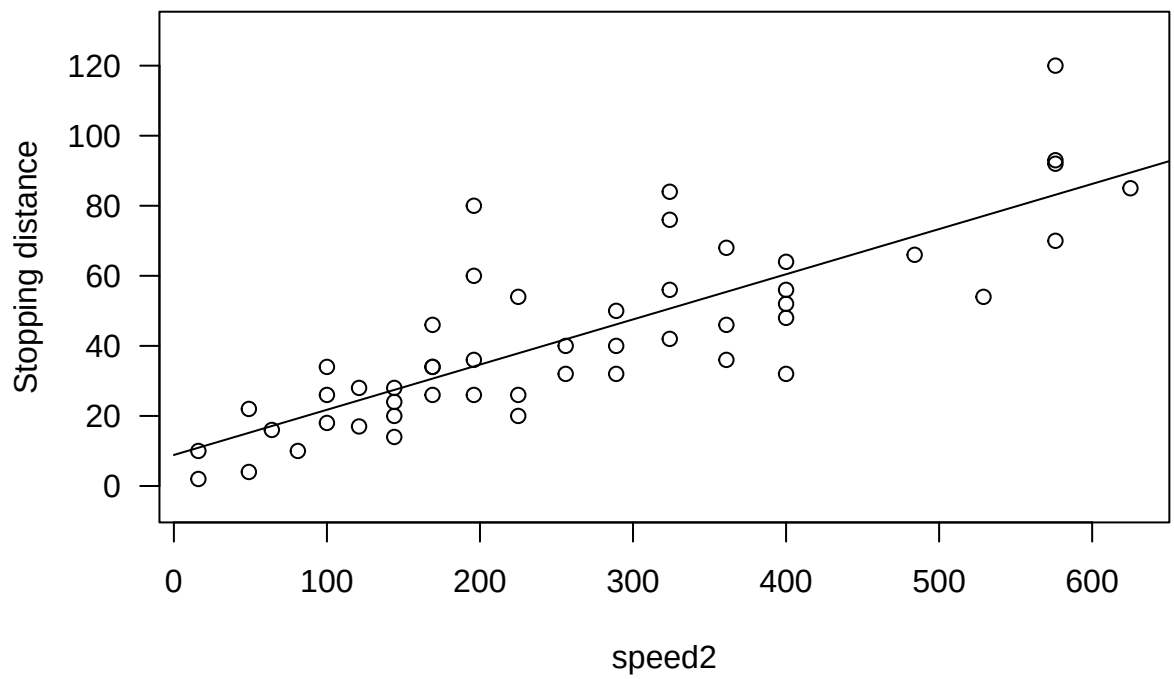
```r
shapiro.test(residuals(fm.pc))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fm.pc)
## W = 0.93034, p-value = 0.005694
```

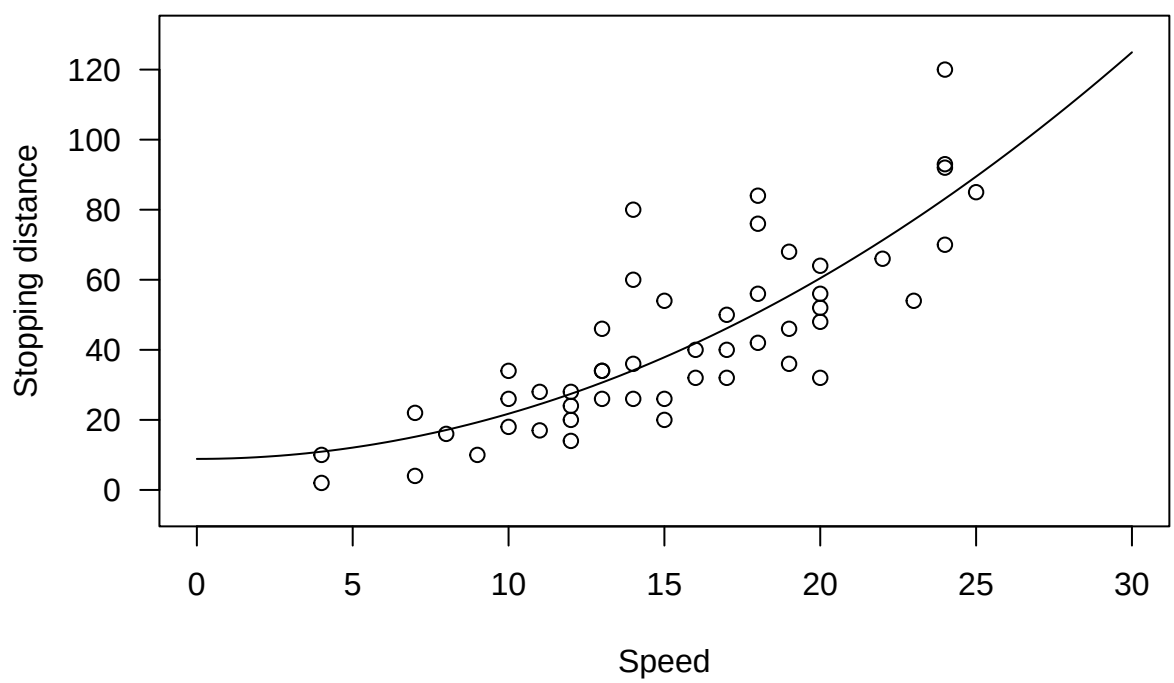
```r
# The first PC is basically speed2 (centered with respect to the mean),
# hence we could just consider the regressor speed2
#! As a original col is more interpretable than PCs, so if the difference
#! is not that much take that, like here speed2

#! Usually howevere if there is a quadratic term we should mantain the linear term,
#! as they tell an interaction. But if some other knowledge (like here physics) we
#! know that there is no linear effect, remove it and use just the quadratic ok

fm.2 = lm(distance ~ speed2)
summary(fm.2)
```

```
##
## Call:
## lm(formula = distance ~ speed2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.448  -9.211  -3.594   5.076  45.862
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.86005    4.08633   2.168   0.0351 *
## speed2       0.12897    0.01319   9.781  5.2e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.05 on 48 degrees of freedom
## Multiple R-squared:  0.6659, Adjusted R-squared:  0.6589
## F-statistic: 95.67 on 1 and 48 DF,  p-value: 5.2e-13
```

```r
# Note: when using as regressors polynomials of the same variable (ie, in
# the framework of polynomial regression), some authors recommend instead to
# keep all the orders lower than the maximum that one wants to keep (eg, if
# keeping the maximum order 3, they recommend keeping also all the terms
# of order 2 and 1, for a "hierarchy" principle)

plot(speed2, distance, xlab='speed2', ylab='Stopping distance', las=1,
     xlim=c(15,626), ylim=c(-5,130))
x = seq(0,650,by=1)
b = coef(fm.2)
lines(x, b[1]+b[2]*x)
```

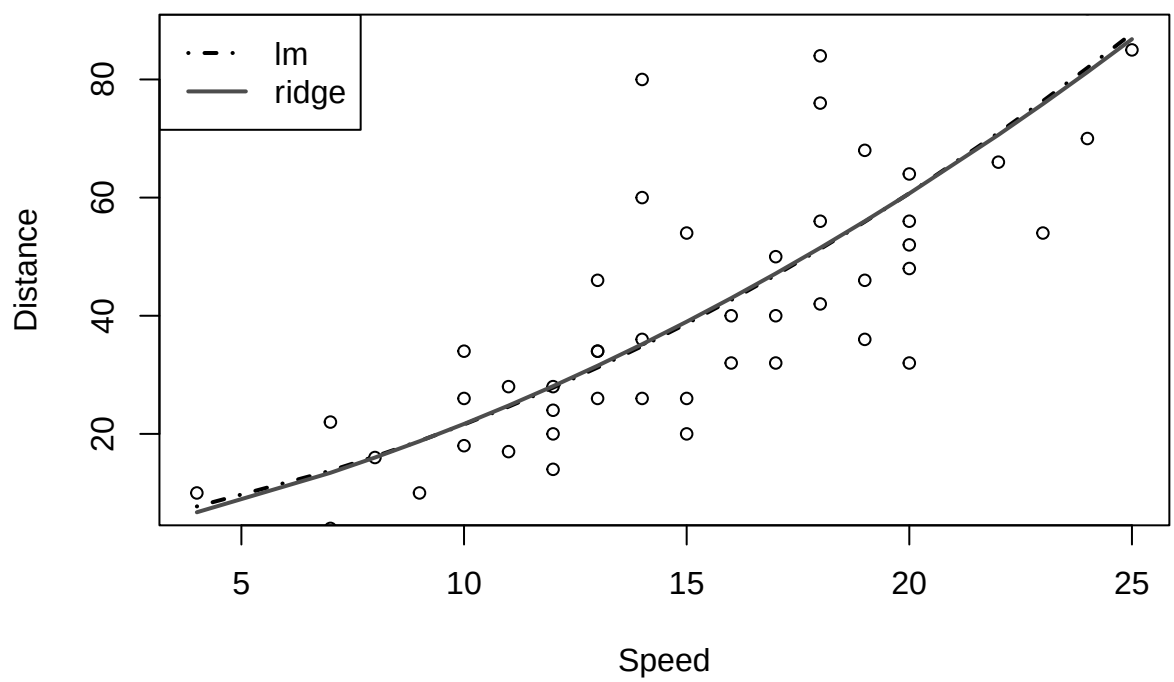



```
plot(cars, xlab='Speed', ylab='Stopping distance', las=1, xlim=c(0,30), ylim=c(-5,130))
x = seq(0,30,by=0.1)
b = coef(fm.2)
lines(x, b[1]+b[2]*x^2)
```



10.1.2 Slt through ridge regression

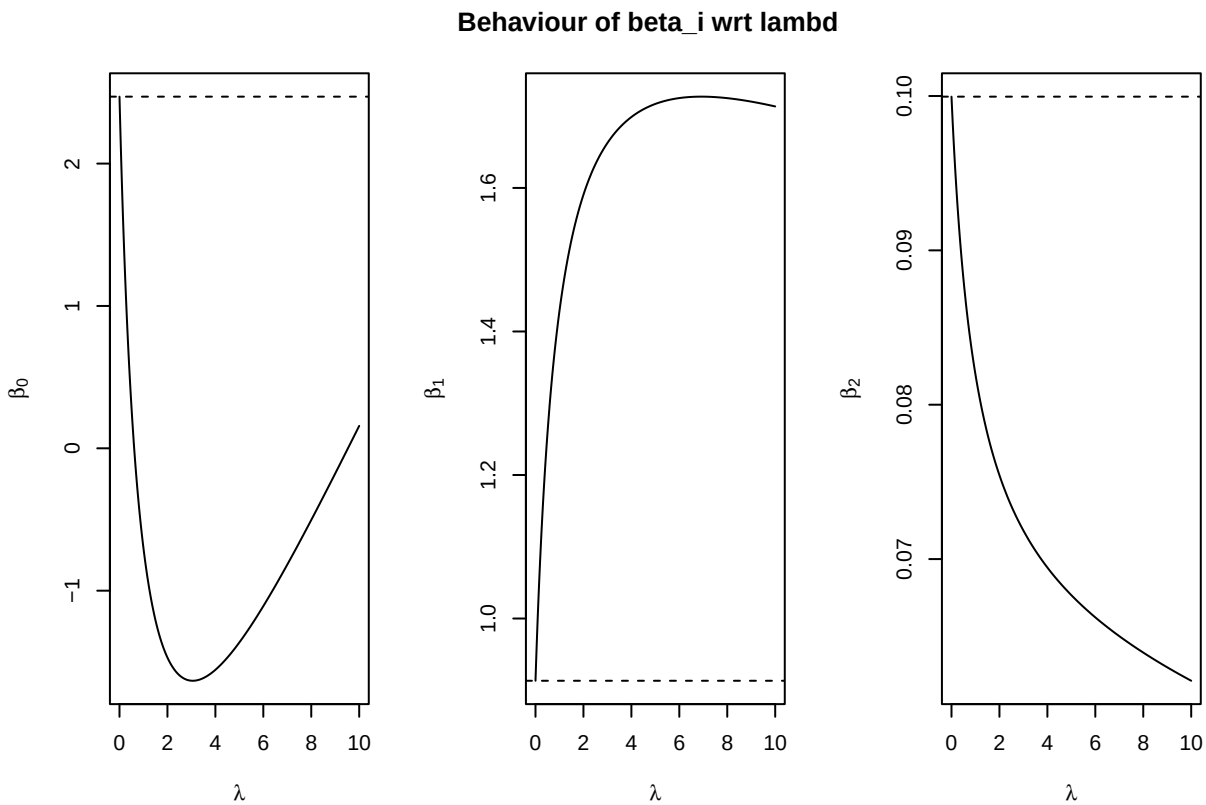
```
### Otherwise, another possible solution to colinearity: ridge regression
#! The idea is to artificially shrinks the range in which the estimators beta can
#! vary, so that we dont get large variance and also we get smoother models.
#! Like more regular models, so we reduce the risk of overfit
# help(lm.ridge)

# Fix lambda
lambda = .5
#! controls how much we want to control the L2 norm of our coefficients
#! In practice we always have to tune it, try different values and find the
#! good one by cross validation

fit.ridge = lm.ridge(distance ~ speed1 + speed2, lambda = lambda)
# Note: to fit the model, R automatically centers X and y wrt their mean.

coef.ridge = coef(fit.ridge)
yhat.lm = cbind(rep(1,n), speed1, speed2)%*%coef(fm)  # LM fitted values
yhat.r = cbind(rep(1,n), speed1, speed2)%*%coef.ridge # ridge fitted values

plot(speed1, yhat.lm, type='l', lty=4, lwd=2, ylab='Distance',xlab='Speed')
points(speed1, distance, pch=1, cex=.8)
matlines(speed1, yhat.r, type='l', lty=1,col=grey.colors(length(lambda)), lwd=2)
legend("topleft",c("lm","ridge"),lty=c(4,1),
       col=c("black",grey.colors(length(lambda))),lwd=2)
```




```
# Repeat for a grid of lambda's
lambda.c = seq(0,10,0.01)
fit.ridge = lm.ridge(distance ~ speed1 + speed2, lambda = lambda.c)

par(mfrow=c(1,3))
plot(lambda.c,coef(fit.ridge)[,1], type='l', xlab=expression(lambda),
     ylab=expression(beta[0]))
abline(h=coef(fm)[1], lty=2)
plot(lambda.c,coef(fit.ridge)[,2], type='l', xlab=expression(lambda),
```

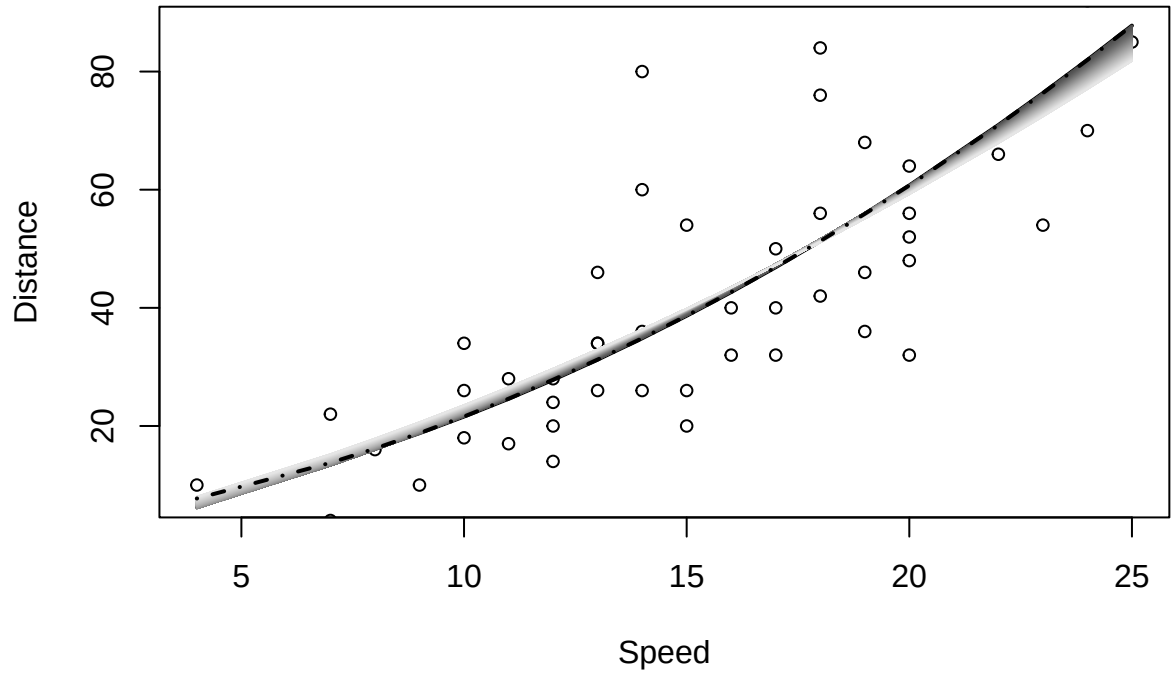
```
     ylab=expression(beta[1]))
title("Behaviour of beta_i wrt lambda")
abline(h=coef(fm)[2], lty=2)
plot(lambda.c,coef(fit.ridge)[,3], type='l', xlab=expression(lambda),
     ylab=expression(beta[2]))
abline(h=coef(fm)[3], lty=2)
```

10.1.2.1 Grid of lambda




```
#! Dashed lines are the OLS (clasical lm) method, from which in effect ridge
#! starts, ie for lambda=0 we get still them

yhat.lm = cbind(rep(1,n), speed1, speed2)%*%coef(fm)
```

```
plot(speed1, yhat.lm, type='l', lty=1, lwd=2, ylab='Distance',
     xlab='Speed')
points(speed1, distance, pch=1, cex=.8)
yhat.r = NULL
for(i in 1:length(lambda.c))
  yhat.r=cbind(yhat.r, cbind(rep(1,n), speed1, speed2)%*%coef(fit.ridge)[i,])
matlines(speed1, yhat.r, type='l', lty=1,
         col=grey.colors(length(lambda.c)))
lines(speed1, yhat.lm, type='l', lty=4, lwd=2, ylab='Distance',
      xlab='Speed')
```




```
# Choice of the optimal lambda, eg, via cross-validation
# GCV -> Generalized Cross Validation
select(fit.ridge)
```

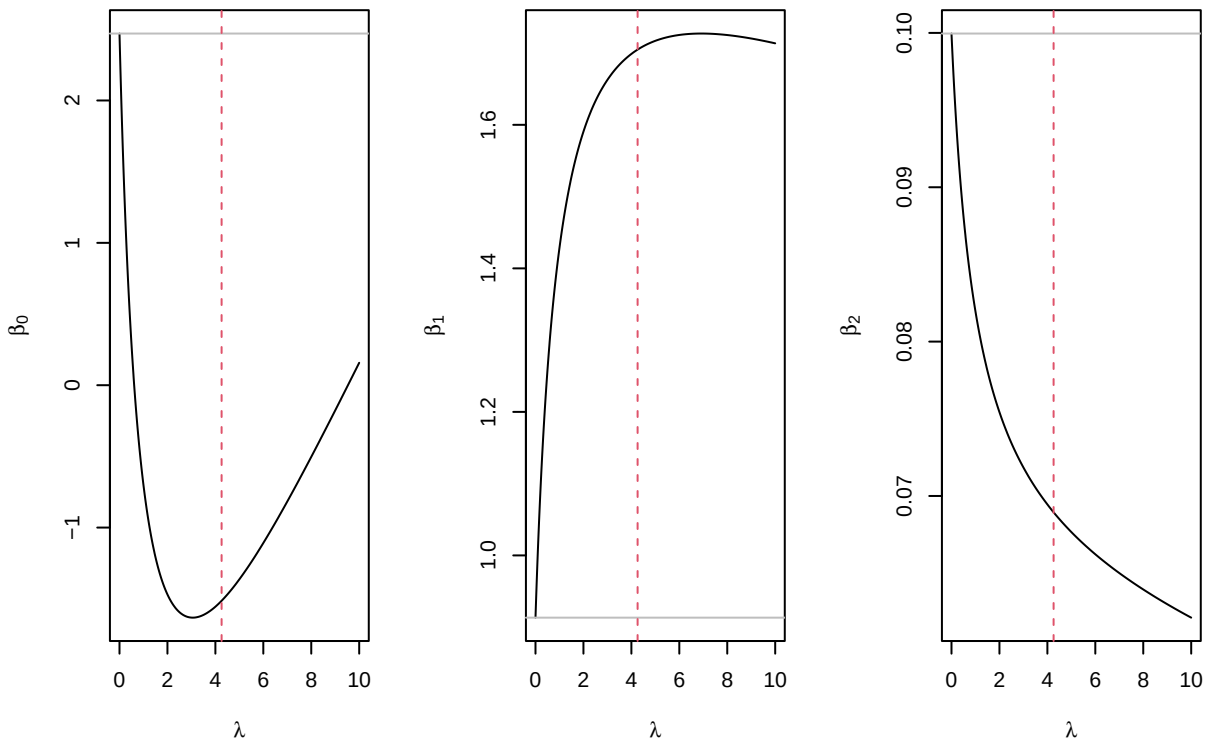
10.1.2.2 Choose lambda with CV

```
## modified HKB estimator is 0
## modified L-W estimator is 0
## smallest value of GCV  at 4.26
```

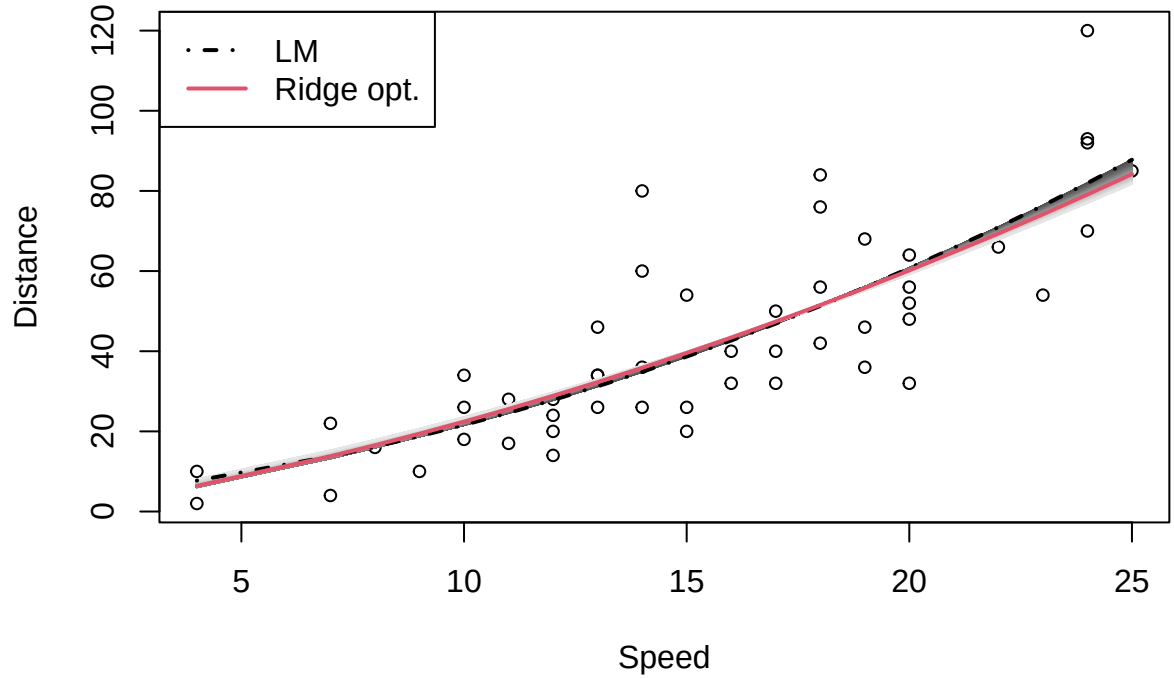
```
# or
lambda.opt = lambda.c[which.min(fit.ridge$GCV)]
lambda.opt
```

```
## [1] 4.26
```

```
#! or manual K-fold cross validation

par(mfrow=c(1,3))
plot(lambda.c,coef(fit.ridge)[,1], type='l', xlab=expression(lambda),
     ylab=expression(beta[0]))
abline(h=coef(fm)[1], lty=1, col='grey')
abline(v=lambda.opt, col=2, lty=2)
plot(lambda.c,coef(fit.ridge)[,2], type='l', xlab=expression(lambda),
     ylab=expression(beta[1]))
abline(h=coef(fm)[2], lty=1, col='grey')
abline(v=lambda.opt, col=2, lty=2)
plot(lambda.c,coef(fit.ridge)[,3], type='l', xlab=expression(lambda),
     ylab=expression(beta[2]))
abline(h=coef(fm)[3], lty=1, col='grey')
abline(v=lambda.opt, col=2, lty=2)
```




```
plot(speed1, distance, pch=1, cex=.8, ylab='Distance',
     xlab='Speed')
matlines(speed1, yhat.r, type='l', lty=1,
         col=grey.colors(length(lambda.c)))
lines(speed1, yhat.lm, type='l', lty=4, lwd=2, ylab='Distance',
      xlab='Speed')
lines(speed1, yhat.r[,which.min(fit.ridge$GCV)], type='l', lty=1, lwd=2,
      col=2, ylab='Distance', xlab='Speed')
legend("topleft", c('LM', 'Ridge opt.' ), lty=c(4,1), col=c(1,2), lwd=2)
```




```
coef.ridge = coef(fit.ridge)[which.min(fit.ridge$GCV),]
coef.ridge
```

```
##                   speed1      speed2
## -1.51415846  1.70485617  0.06894229
```

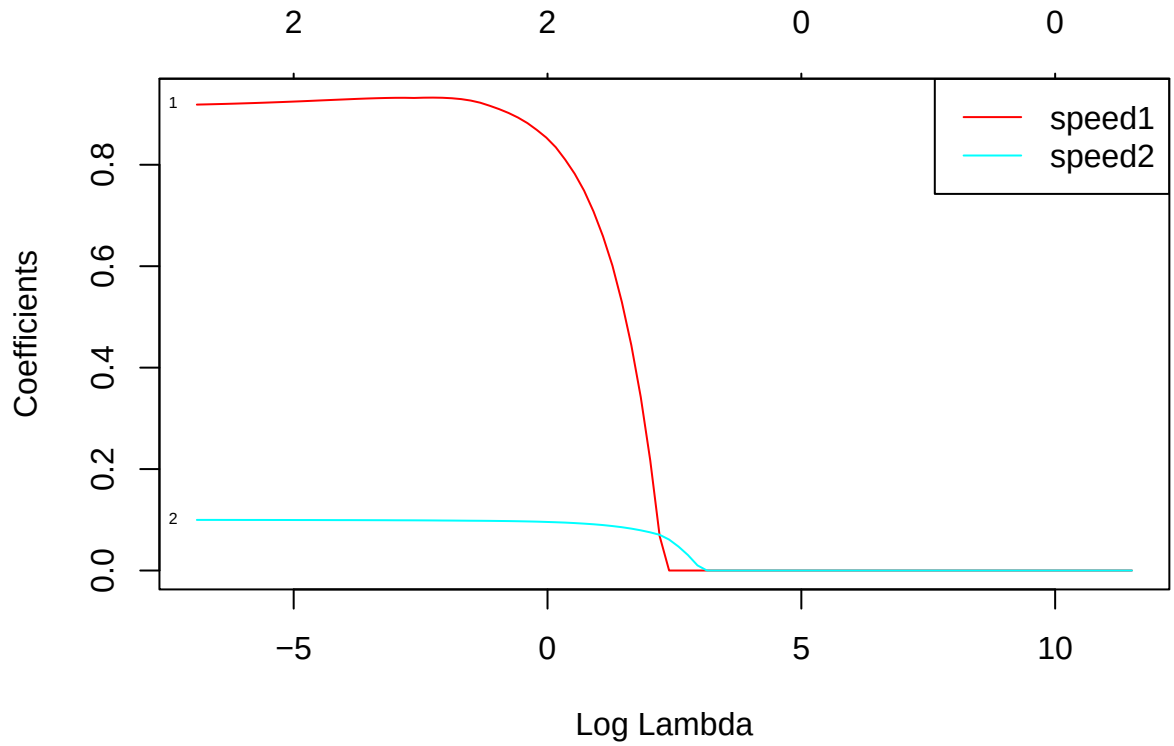
```
# vs
fm$coefficients
```

```
## (Intercept)      speed1      speed2
##   2.4701378   0.9132876   0.0999593
```

```
#! Classical (OLS) lm is invariant wrt tranformations: from meters to centimeters
#! the beta get equally scaled. While with ridge and lasso no, those conversion
#! change the output.
#! So with them we could also ok center the values, but also scale wrt the variance
```

10.1.3 Slt through lasso/elasticnet regression

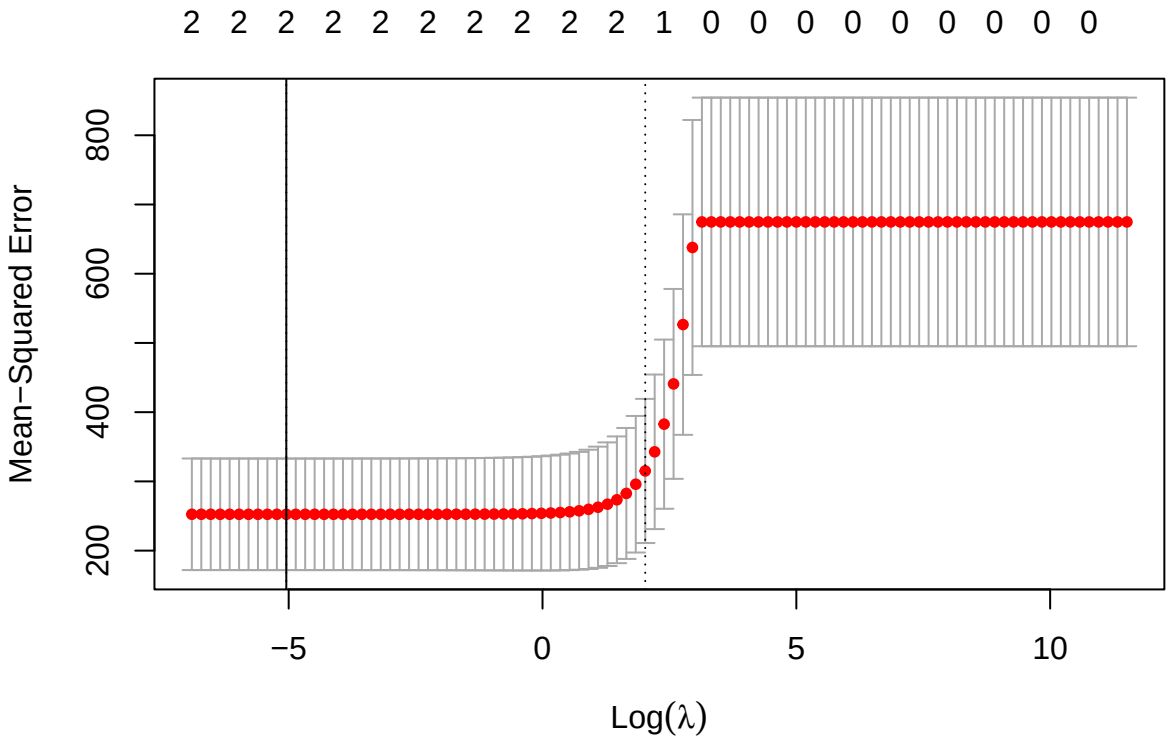
```
### Otherwise, another possible solution to colinearity, which also performs
#! variables selection: lasso(1) or elasticnect(mix) or ridge(0) regression
# help(glmnet)

# Build the matrix of predictors
x = model.matrix(distance~speed1+speed2)[,-1]
# Build the vector of response/target
y = distance

# Let's set a (log-)grid of candidate lambda's for the estimate
lambda.grid = 10^seq(5,-3,length=100)
fit.lasso = glmnet(x,y, lambda = lambda.grid)
# default: alpha=1 -> lasso
# [note: alpha=0 -> ridge regression]

plot(fit.lasso,xvar='lambda',label=TRUE, col = rainbow(dim(x)[2]))
legend('topright', dimnames(x)[[2]], col =  rainbow(dim(x)[2]), lty=1, cex=1)
```




```
# Let's set lambda via cross validation
cv.lasso = cv.glmnet(x,y,lambda=lambda.grid) # default: 10-fold CV

bestlam.lasso = cv.lasso$lambda.min
bestlam.lasso
```

```
## [1] 0.006428073
```

```
plot(cv.lasso)
abline(v=log(bestlam.lasso), lty=1)
```




```
#! We get the estimate for the MSE, but also confidence intervals
#! The dashed line identifies the elbow zone, looking from left to right, which
#! says which lambda we should choose for a model "correctly" relaxed

# Get the coefficients for the optimal lambda
coef.lasso = predict(fit.lasso, s=bestlam.lasso, type = 'coefficients')[1:3,]
coef.lasso
```

```
## (Intercept)      speed1      speed2
##  2.40283823  0.92441235  0.09956612
```

10.2 Multiple linear regression

```
#_____________________________________________________________________________
##### Multiple linear regression
#####-------------------------------------------------------

data = read.table('Lab10/concrete.txt', header=T)

head(data)
```

```
##   Alluminium Silicate Alluminium_ferrite Silicate_bicalcium Hardness_concrete
## 1          7       16                  6                 60              78.5
## 2          1       19                 15                 52              74.3
## 3         11       56                  8                 20             104.3
## 4         11       31                  8                 47              87.6
## 5          7       52                  6                 33              95.9
## 6         11       55                  9                 22             109.2
```

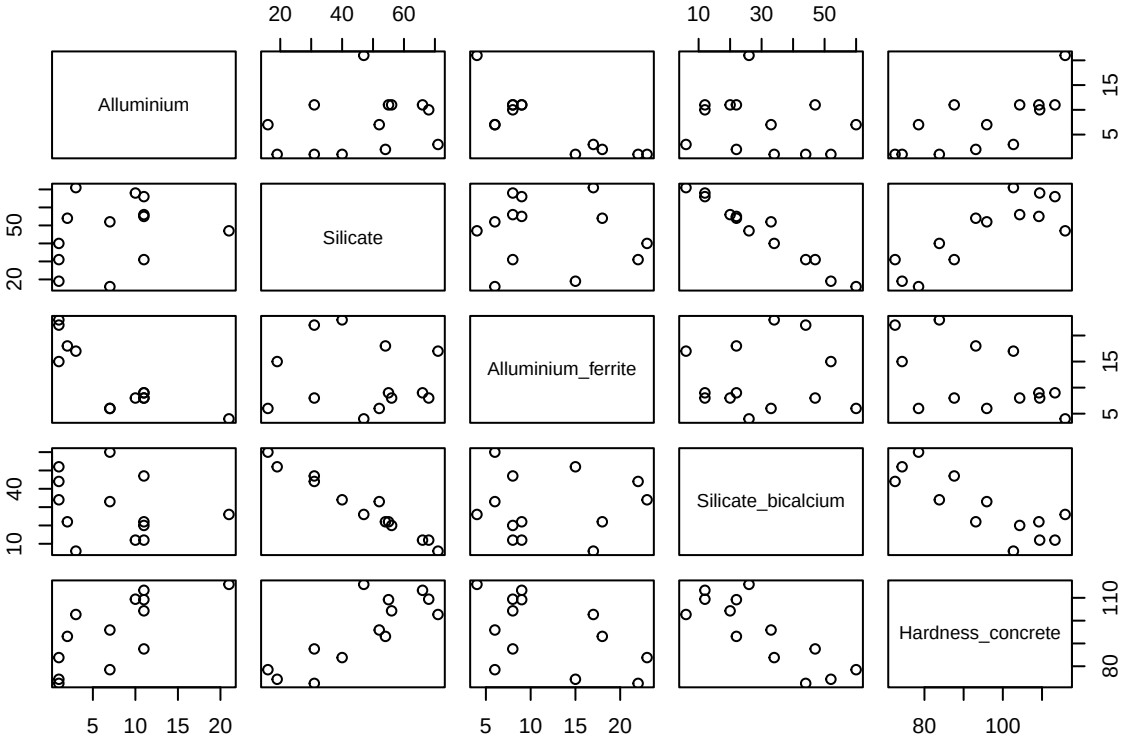
```
dim(data)
```

```
## [1] 13  5
```

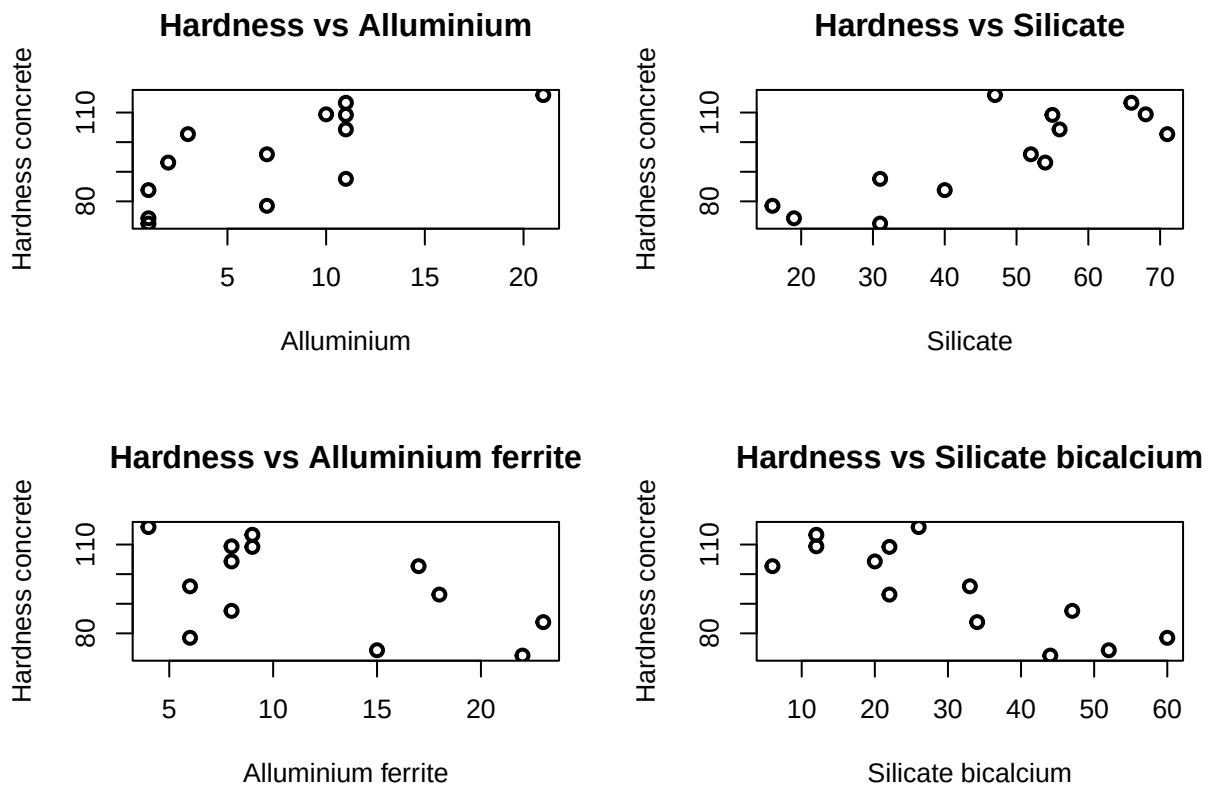
```
names(data)
```

```
## [1] "Alluminium"         "Silicate"           "Alluminium_ferrite"
## [4] "Silicate_bicalcium" "Hardness_concrete"
```

```r
#! Pair plot, first thing to do when dealing with multivariate data
pairs(data)
```




```r
#! We see silicate or silicate_bicalcium will manifest some good relation with
#! our target hardness, and also are correlated so we may suffer collinearity

attach(data)
y = Hardness_concrete
# detach(data)

par(mfrow=c(2,2))
plot(Alluminium, y, main='Hardness vs Alluminium', lwd=2,
     xlab='Alluminium', ylab='Hardness concrete')
plot(Silicate, y, main='Hardness vs Silicate', lwd=2,
     xlab='Silicate', ylab='Hardness concrete')
plot(Alluminium_ferrite, y, main='Hardness vs Alluminium ferrite', lwd=2,
     xlab='Alluminium ferrite', ylab='Hardness concrete')
plot(Silicate_bicalcium, y, main='Hardness vs Silicate bicalcium', lwd=2,
     xlab='Silicate bicalcium', ylab='Hardness concrete')
```




```r
## Multiple linear regression
result = lm(y ~ Alluminium + Silicate + Alluminium_ferrite + Silicate_bicalcium)
summary(result)
```

```
##
## Call:
## lm(formula = y ~ Alluminium + Silicate + Alluminium_ferrite +
##     Silicate_bicalcium)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0990 -1.7178  0.2919  1.2055  3.7526
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        124.4809    26.7557   4.653  0.00164 **
## Alluminium           0.9739     0.2835   3.435  0.00889 **
## Silicate            -0.1405     0.2891  -0.486  0.63996
## Alluminium_ferrite  -0.4974     0.2751  -1.808  0.10820
## Silicate_bicalcium  -0.7974     0.3214  -2.481  0.03805 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.484 on 8 degrees of freedom
## Multiple R-squared:  0.9818, Adjusted R-squared:  0.9727
## F-statistic:   108 on 4 and 8 DF,  p-value: 5.385e-07
```

```r
vif(result)
```

```
##         Alluminium           Silicate Alluminium_ferrite Silicate_bicalcium
##           5.408208          52.818202           6.035697          56.260964
```

10.2.1 PCA regression

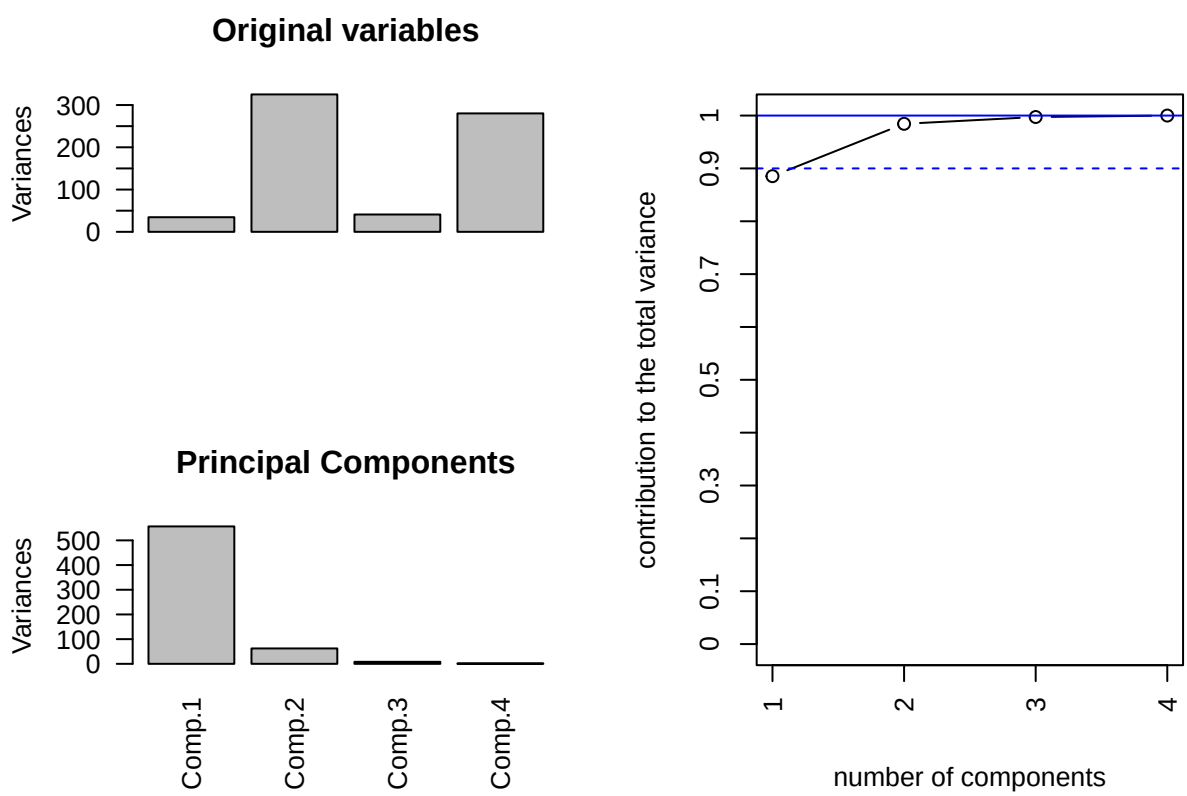
```r
#### PCA regression
result.pc = princomp(cbind(Alluminium,Silicate,Alluminium_ferrite,Silicate_bicalcium),
                     scores=TRUE)
summary(result.pc)
```

```
## Importance of components:
##                            Comp.1     Comp.2     Comp.3      Comp.4
## Standard deviation     23.5887588 7.89074012 2.84498593 1.312101401
## Proportion of Variance  0.8853169 0.09906587 0.01287801 0.002739198
## Cumulative Proportion   0.8853169 0.98438279 0.99726080 1.000000000
```

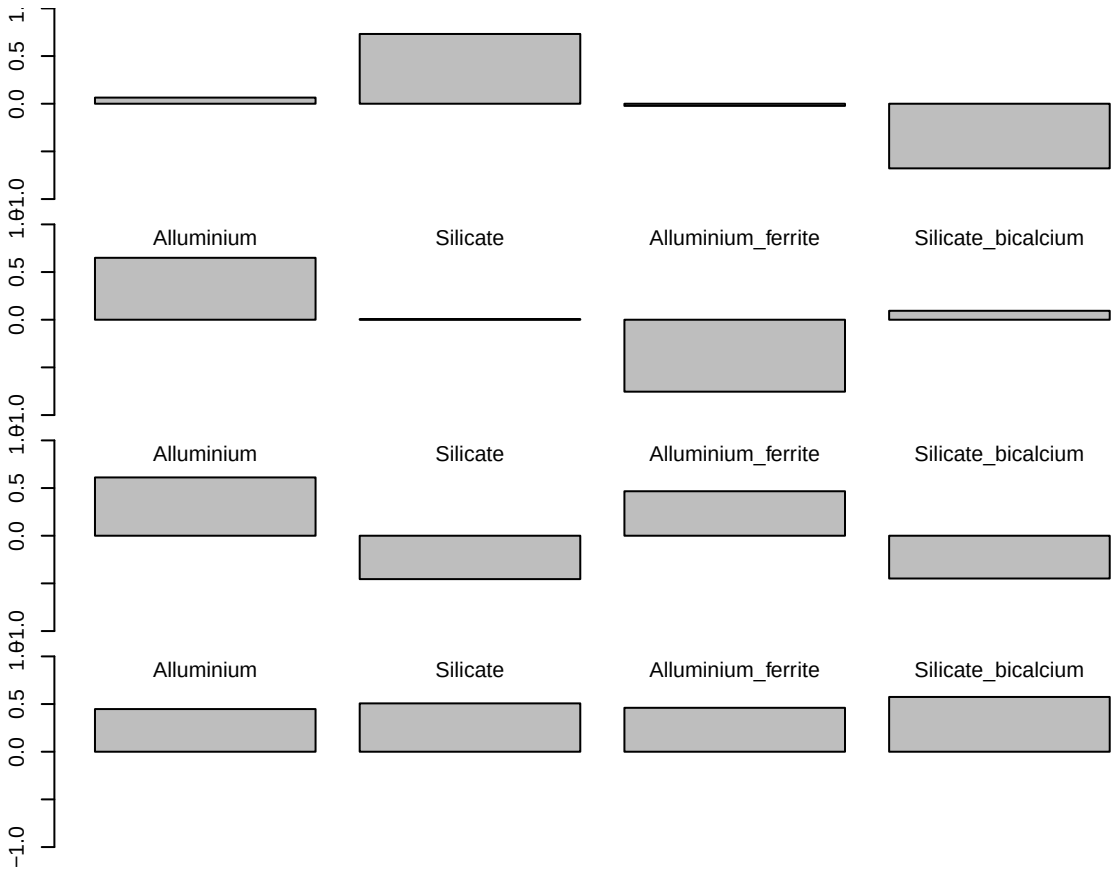
```
#! With only 2 PCs we explained 98% of variability
#! PC1 contrasts silicate and silicate_bicalcium
#! PC2 contrasts the others, alluminium and alluminium_ferrite
result.pc$load
```

```
##
## Loadings:
##                    Comp.1 Comp.2 Comp.3 Comp.4
## Alluminium                 0.649  0.611  0.448
## Silicate            0.732        -0.455  0.507
## Alluminium_ferrite        -0.755  0.467  0.461
## Silicate_bicalcium -0.678        -0.449  0.574
##
##                 Comp.1 Comp.2 Comp.3 Comp.4
## SS loadings       1.00   1.00   1.00   1.00
## Proportion Var    0.25   0.25   0.25   0.25
## Cumulative Var    0.25   0.50   0.75   1.00
```

```
# Explained variance

layout(matrix(c(2,3,1,3),2,byrow=T))
barplot(result.pc$sdev^2, las=2, main='Principal Components', ylab='Variances')
barplot(c(sd(Alluminium),sd(Silicate),sd(Alluminium_ferrite),
          sd(Silicate_bicalcium))^2, las=2, main='Original variables', ylab='Variances')
plot(cumsum(result.pc$sdev^2)/sum(result.pc$sde^2), type='b', axes=F,
     xlab='number of components', ylab='contribution to the total variance', ylim=c(0,1))
abline(h=1, col='blue')
abline(h=0.9, lty=2, col='blue')
box()
axis(2,at=0:10/10,labels=0:10/10)
axis(1,at=1:4,labels=1:4,las=2)
```



```r
#! From the first plot we may could have scaled the variables, as we see some of
#! them with much larger variance

# Loadings
par(mar = c(1,4,0,2), mfrow = c(4,1))
for(i in 1:4) barplot(result.pc$load[,i], ylim = c(-1, 1))
```




```r
# Dimensionality reduction: select first two PCs:
pc1 = result.pc$scores[,1]
pc2 = result.pc$scores[,2]

# Now we estimate the model using the first two PCs as regressors.
# Model: y = b0 + b1*PC1+ b2*PC2 + eps, eps~N(0,sigma^2)
fm.pc = lm(y ~ pc1 + pc2)

summary(fm.pc)
```

```
##
## Call:
## lm(formula = y ~ pc1 + pc2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.623 -2.245 -1.312  1.587  5.094
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 95.42308    0.94690 100.775 2.27e-16 ***
## pc1          0.51195    0.04014  12.754 1.64e-07 ***
## pc2          0.93214    0.12000   7.768 1.52e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.414 on 10 degrees of freedom
## Multiple R-squared:  0.9571, Adjusted R-squared:  0.9485
## F-statistic: 111.5 on 2 and 10 DF,  p-value: 1.456e-07
```

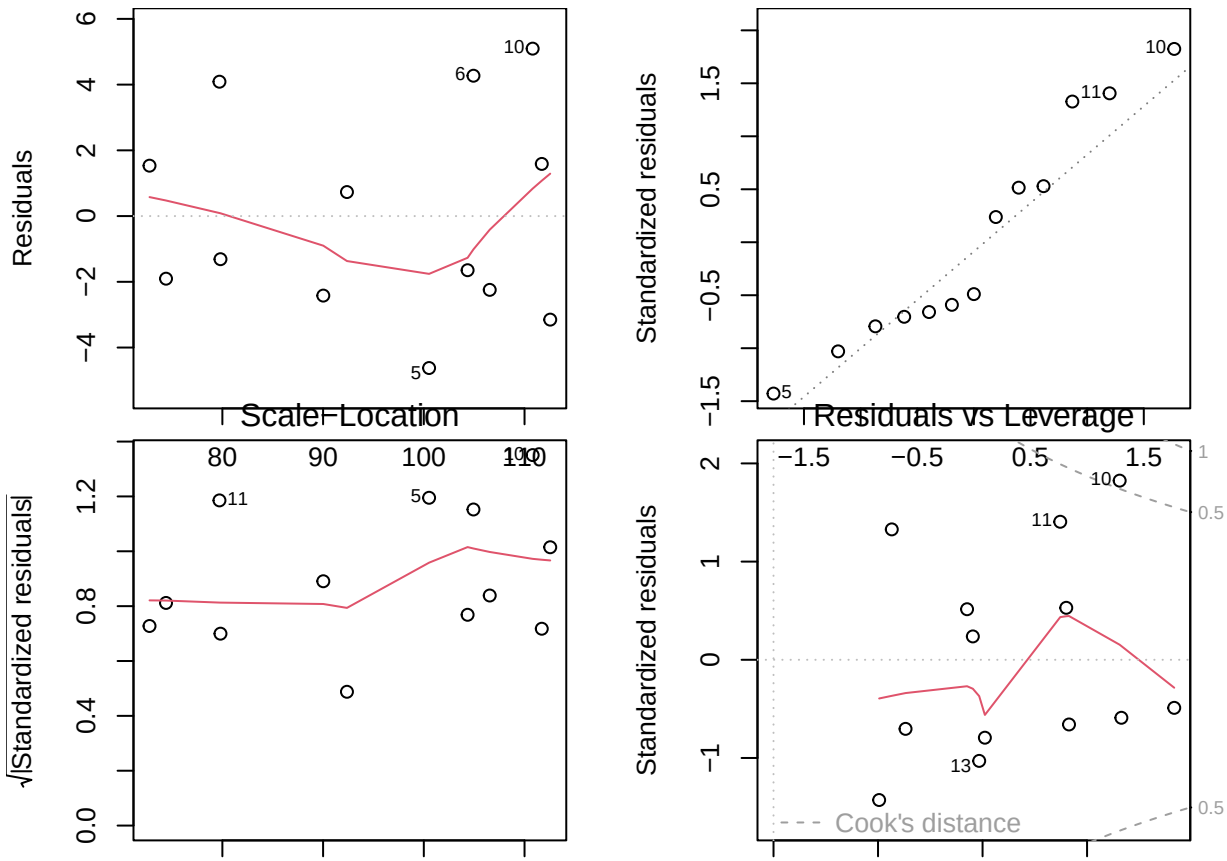
```r
# We can re-write the model as:
# Model:
# y= b0 + b1*PC1 + b2*PC2 + eps =
#  = b0 + b1*(e11*(Alluminium-m1)+e21*(Silicate-m2)+e31*(Alluminium_ferrite-m3)+
#  e41*(Silicate_bicalcium-m4)) +
#       + b2*(e12*(Alluminium-m1)+e22*(Silicate-m2)+e32*(Alluminium_ferrite-m3)+
#       e42*(Silicate_bicalcium-m4)) + eps =
#  = b0 - b1*e11*m1 - b2*e12*m1 - b1*e21*m2 - b2*e22*m2 +
#       - b1*e31*m3 - b2*e32*m3 - b1*e41*m4 - b2*e42*m4 +
#       + (b1*e11+b2*e12)*Alluminium + (b1*e21+b2*e22)*Silicate +
#       + (b1*e31+b2*e32)*Alluminium_ferrite + (b1*e41+b2*e42)*Silicate_bicalcium + eps
# where e.ij are the loadings, i=1,2,3,4, j=1,2.
# => We can compute the coefficients of the model which used the original
#    regressors
m1 = mean(Alluminium)
m2 = mean(Silicate)
m3 = mean(Alluminium_ferrite)
m4 = mean(Silicate_bicalcium)
beta0 = coefficients(fm.pc)[1] -
  coefficients(fm.pc)[2]*result.pc$load[1,1]*m1 -
  coefficients(fm.pc)[3]*result.pc$load[1,2]*m1 -
  coefficients(fm.pc)[2]*result.pc$load[2,1]*m2 -
  coefficients(fm.pc)[3]*result.pc$load[2,2]*m2 -
  coefficients(fm.pc)[2]*result.pc$load[3,1]*m3 -
  coefficients(fm.pc)[3]*result.pc$load[3,2]*m3 -
  coefficients(fm.pc)[2]*result.pc$load[4,1]*m4 -
  coefficients(fm.pc)[3]*result.pc$load[4,2]*m4
beta1 = coefficients(fm.pc)[2]*result.pc$load[1,1] +
  coefficients(fm.pc)[3]*result.pc$load[1,2]
beta2 = coefficients(fm.pc)[2]*result.pc$load[2,1] +
  coefficients(fm.pc)[3]*result.pc$load[2,2]
beta3 = coefficients(fm.pc)[2]*result.pc$load[3,1] +
  coefficients(fm.pc)[3]*result.pc$load[3,2]
beta4 = coefficients(fm.pc)[2]*result.pc$load[4,1] +
  coefficients(fm.pc)[3]*result.pc$load[4,2]

c(beta0=as.numeric(beta0),beta1=as.numeric(beta1),beta2=as.numeric(beta2),
  beta3=as.numeric(beta3),beta4=as.numeric(beta4))
```

```
##      beta0      beta1      beta2      beta3      beta4
## 89.1310486  0.6383398  0.3804914 -0.7150276 -0.2597465
```

```r
result$coefficients
```

```
##        (Intercept)         Alluminium           Silicate Alluminium_ferrite
##        124.4808741          0.9739087         -0.1405100         -0.4973948
## Silicate_bicalcium
##         -0.7973591
```

```r
#! coeff not that far from each other


# diagnostics of the residuals
par(mfrow=c(2,2))
plot(fm.pc)
```



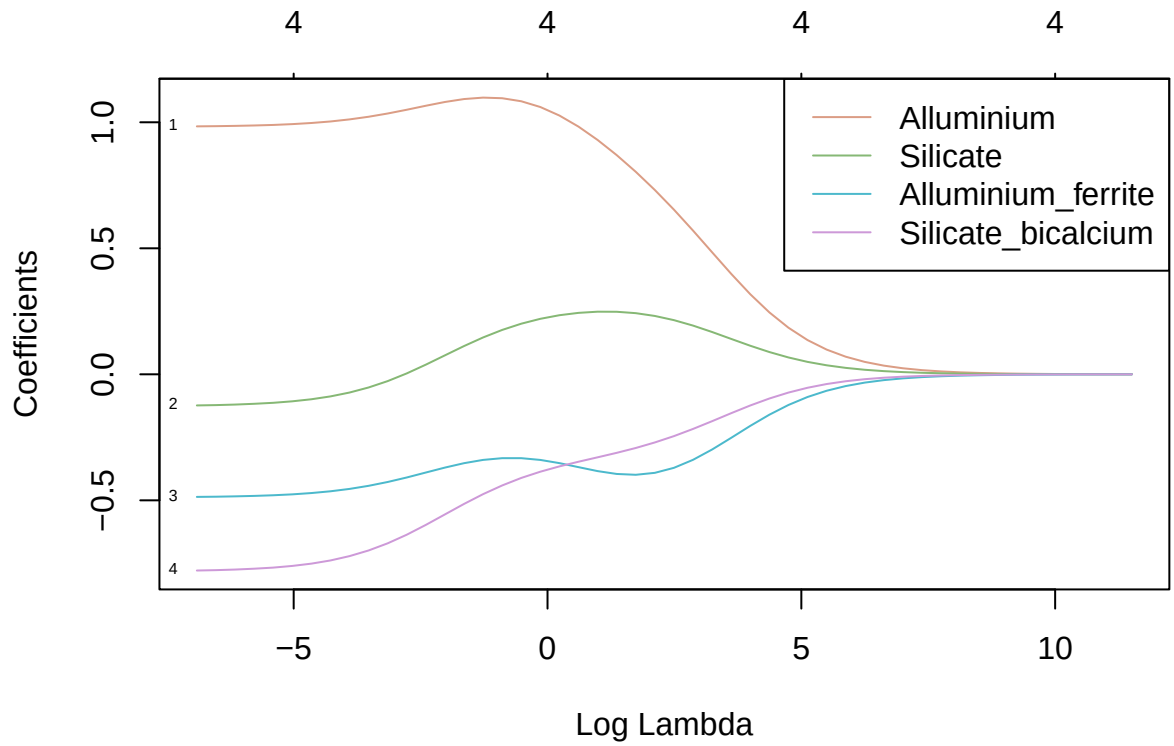

```
shapiro.test(residuals(fm.pc))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fm.pc)
## W = 0.92873, p-value = 0.328
```

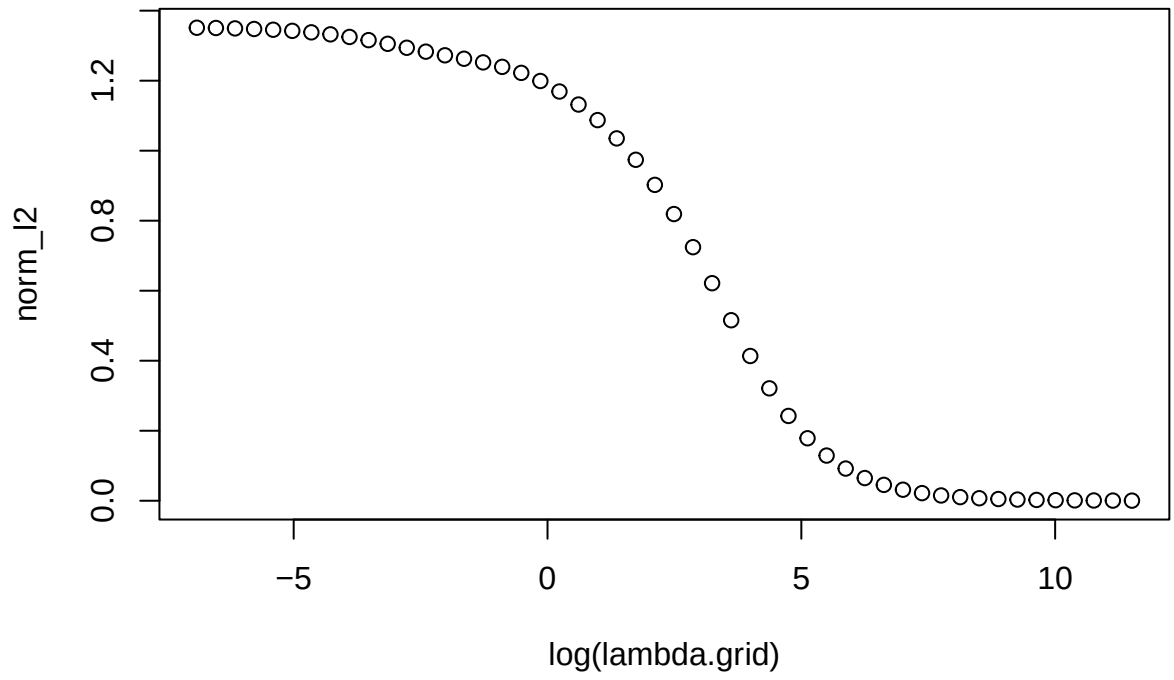
```
### Ridge and Lasso regression with glmnet
# matrix of predictors
x = model.matrix(y ~ Alluminium + Silicate + Alluminium_ferrite + Silicate_bicalcium)[,-1]
# vector of response
y = y
lambda.grid = 10^seq(5,-3,length=50)
```

10.2.2 Ridge regression

```
# Ridge regression
# alpha=0 -> ridge
fit.ridge = glmnet(x,y, lambda = lambda.grid, alpha=0)
# Also possible to let glmnet compute its own lambda sequence (length 100,
# between lambda.max*10^-4 and lambda.max where lambda.max is derived from
# the data)

col.ramp = fun_colori(4,123)
plot(fit.ridge,xvar='lambda',label=TRUE, col = col.ramp)
legend('topright', dimnames(x)[[2]], col =  col.ramp, lty=1, cex=1)
```




```
norm_l2 = NULL
for(i in 1:50) norm_l2 = c(norm_l2,sqrt(sum((fit.ridge$beta[,i])^2)))
plot(log(lambda.grid),norm_l2)
```



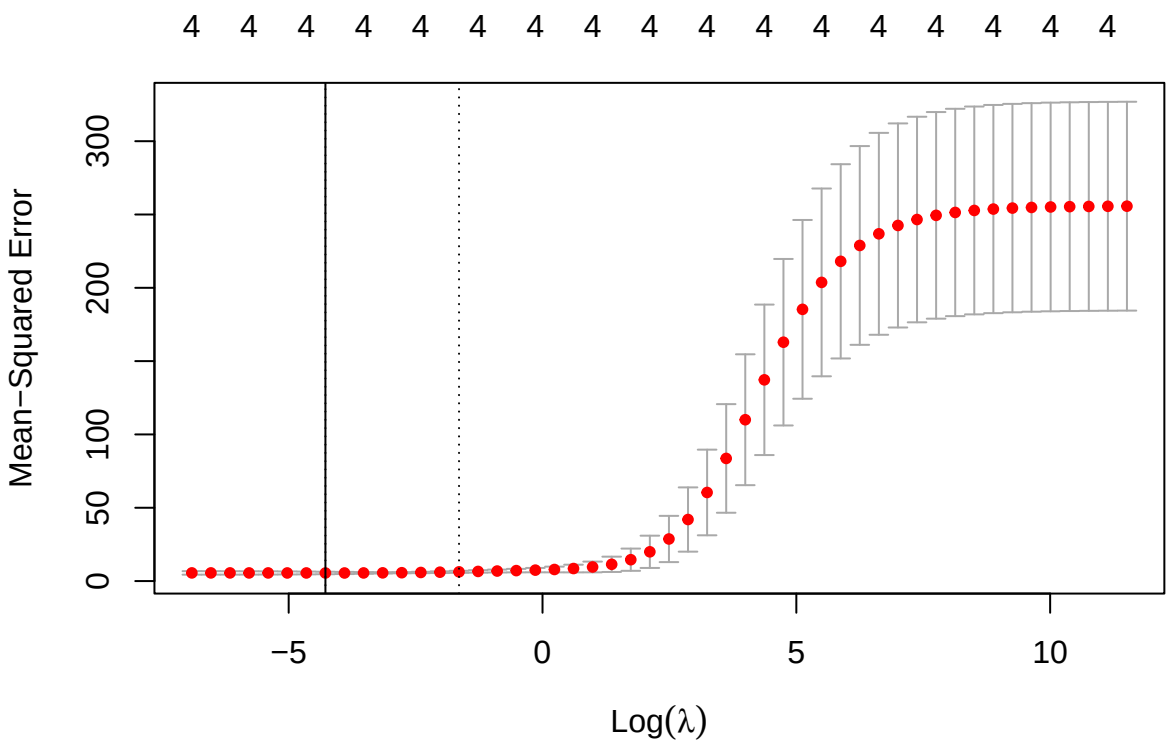
```r
# Let's set lambda via CV
set.seed(1)
cv.ridge = cv.glmnet(x,y,alpha=0,nfolds=3,lambda=lambda.grid)

bestlam.ridge = cv.ridge$lambda.min
bestlam.ridge
```

```
## [1] 0.01389495
```

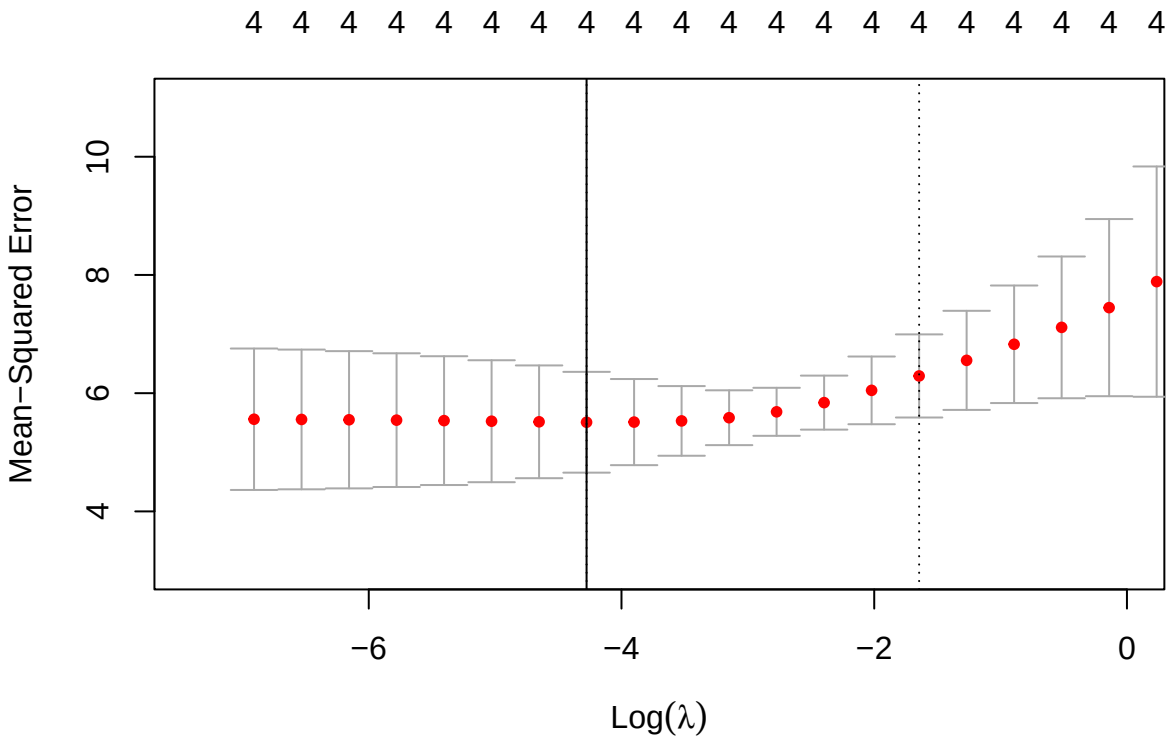
```r
log(bestlam.ridge)
```

```
## [1] -4.276229
```

```r
plot(cv.ridge)
abline(v=log(bestlam.ridge), lty=1)
```



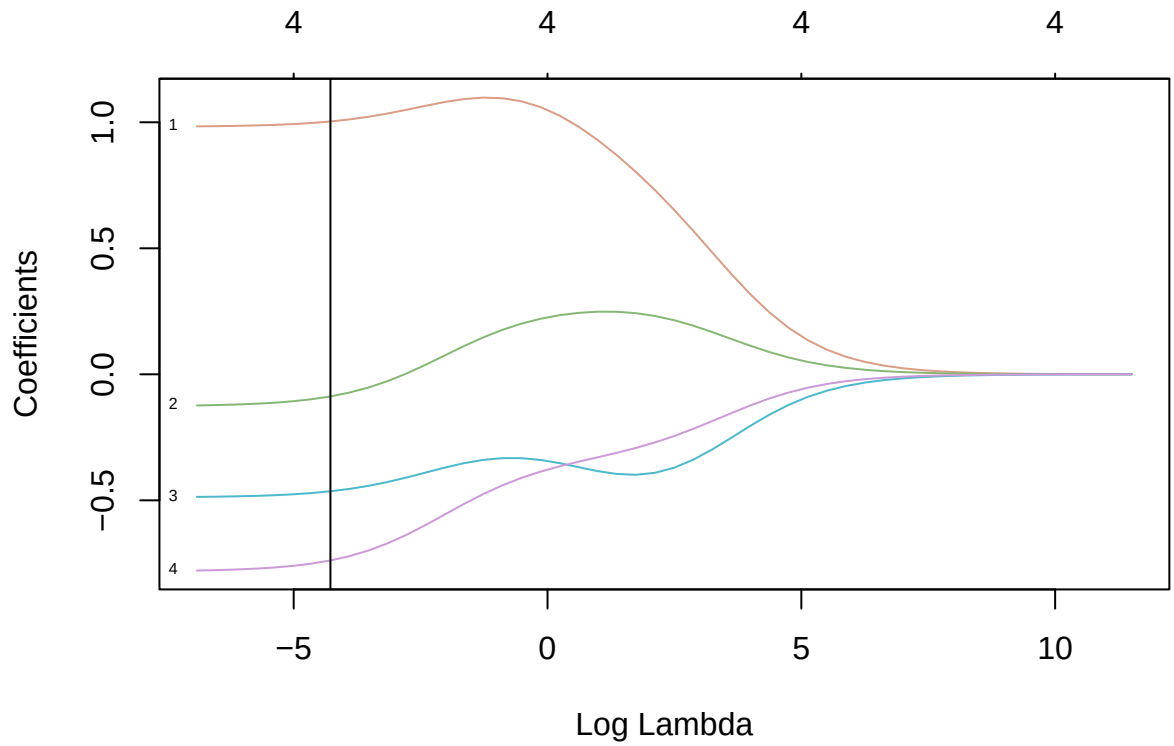

```r
# zoming on that left interesting region
plot(cv.ridge,xlim=c(-7.4,0),ylim=c(3,11))
abline(v=log(bestlam.ridge), lty=1)
```




```r
# Get the coefficients for the optimal lambda
coef.ridge = predict(fit.ridge, s=bestlam.ridge, type = 'coefficients')[1:5,]
coef.ridge
```

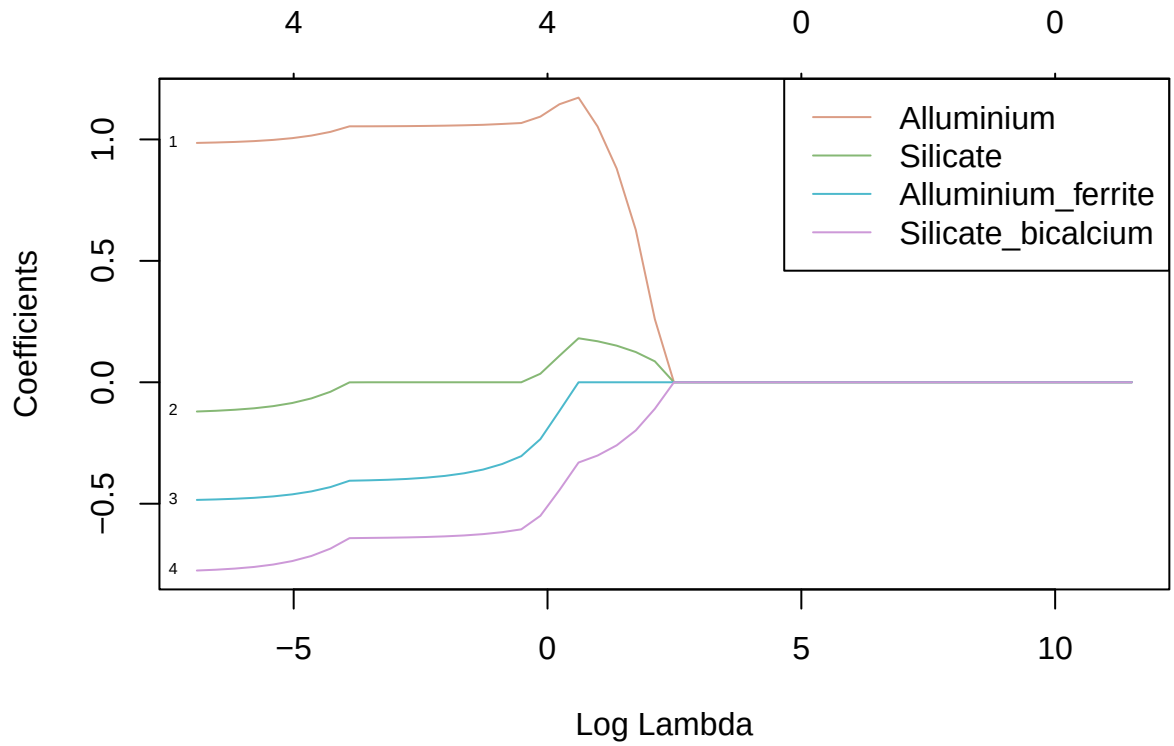
```
##        (Intercept)         Alluminium           Silicate Alluminium_ferrite
##       119.64979497         1.00319395        -0.08774654        -0.46421077
## Silicate_bicalcium
##        -0.73861152
```

```r
plot(fit.ridge,xvar='lambda',label=TRUE, col = col.ramp)
abline(v=log(bestlam.ridge))
```

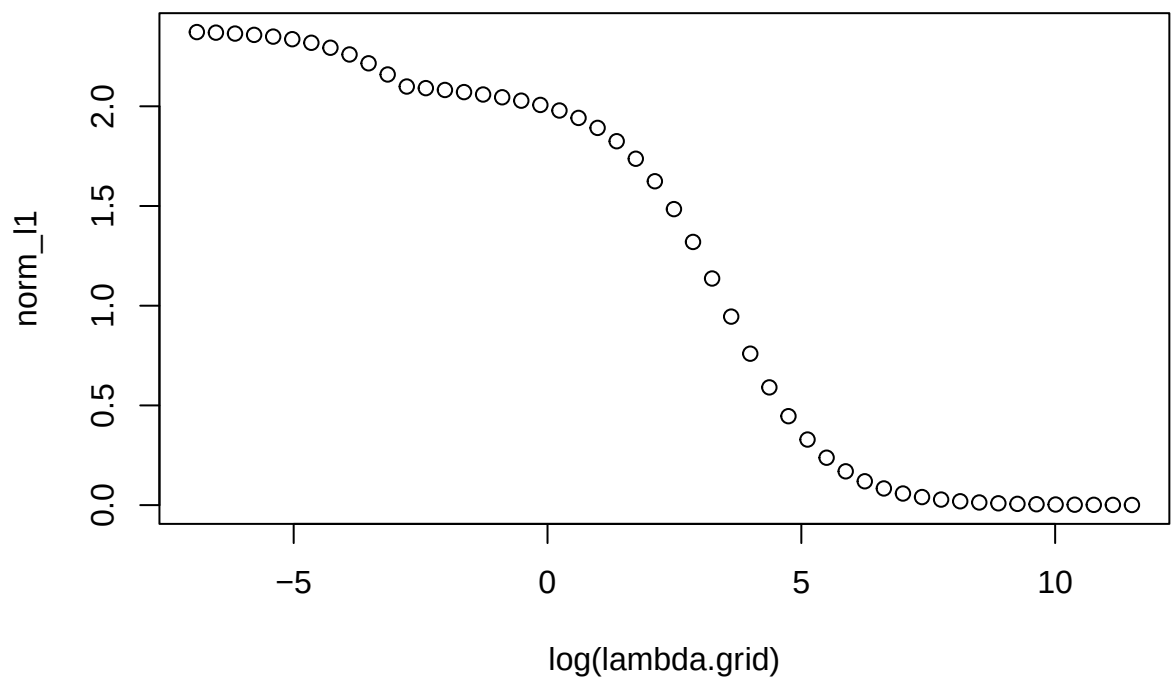


10.2.3 Lasso regression

```
### Lasso regression
# alpha=1 -> lasso
fit.lasso = glmnet(x,y, lambda = lambda.grid, alpha=1)

plot(fit.lasso,xvar='lambda',label=TRUE, col = col.ramp)
legend('topright', dimnames(x)[[2]], col =  col.ramp, lty=1, cex=1)
```



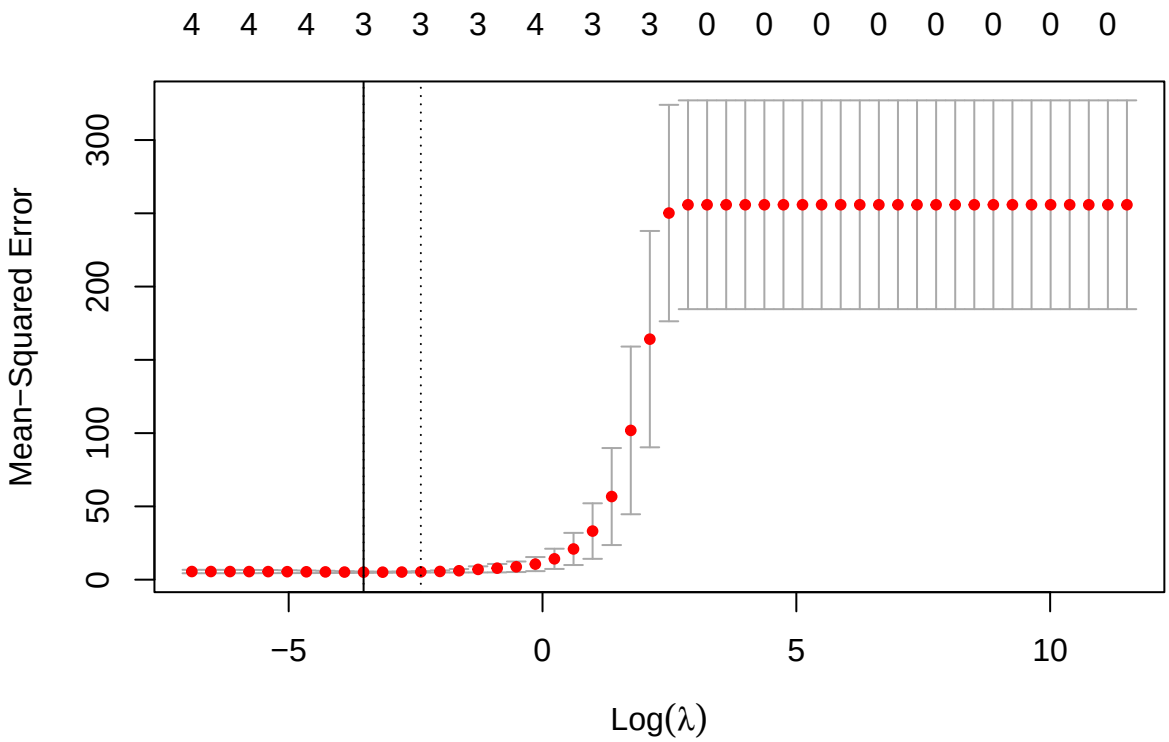

```
norm_l1 = NULL
for(i in 1:50)
  norm_l1 = c(norm_l1,sum(abs(fit.ridge$beta[,i])))

plot(log(lambda.grid),norm_l1)
```



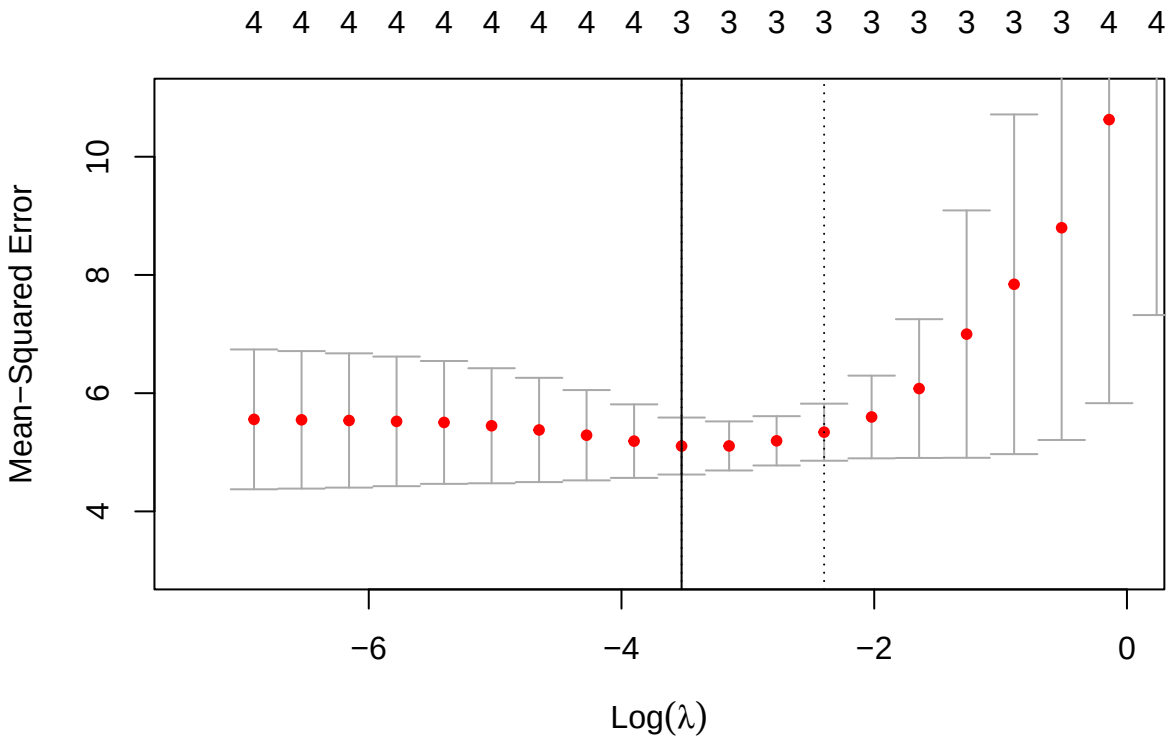
```r
# Let's set lambda via CV
set.seed(1)
cv.lasso = cv.glmnet(x,y,alpha=1,nfolds=3,lambda=lambda.grid)

bestlam.lasso = cv.lasso$lambda.min
bestlam.lasso
```

```
## [1] 0.02947052
```

```r
plot(cv.lasso)
abline(v=log(bestlam.lasso), lty=1)
```



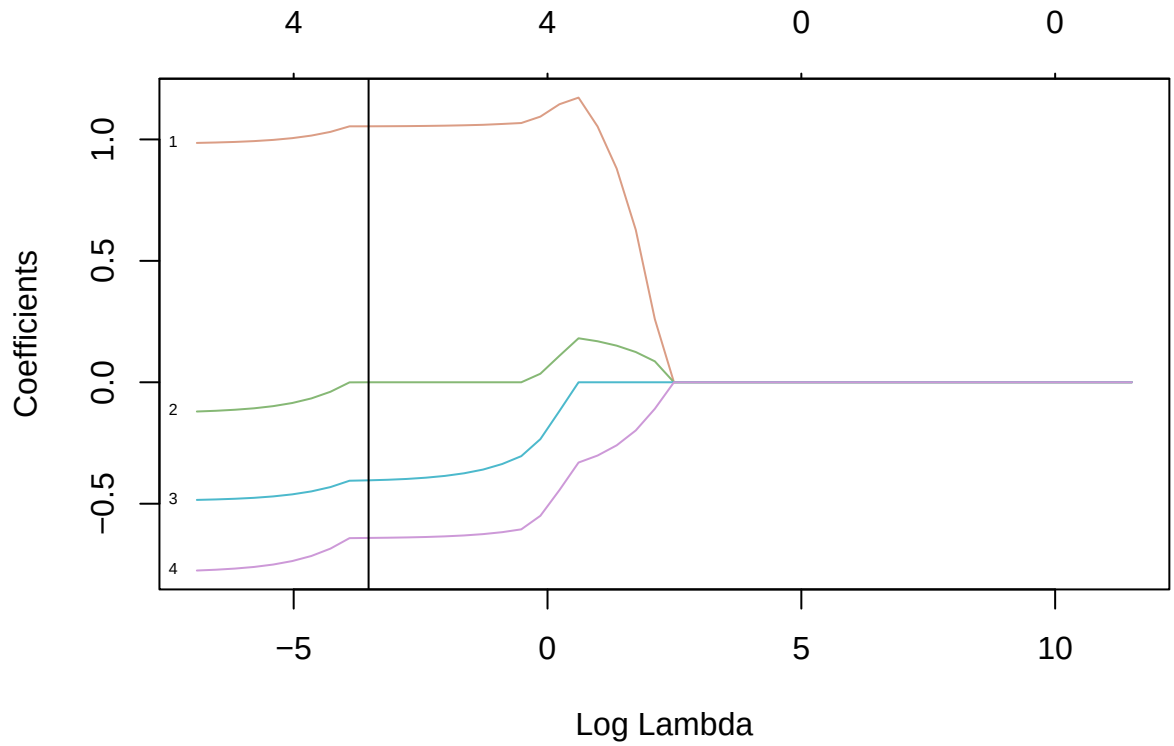

```r
# zoming on that left interesting region
plot(cv.lasso,xlim=c(-7.4,0),ylim=c(3,11))
abline(v=log(bestlam.lasso), lty=1)
```




```
#! dashed line: 3 coeff, so simpler model, with almost the same MSE so still good
#! wrt the optimal one, full line, which has 4 coeff (no ok still 4 actually)

# Get the coefficients for the optimal lambda
coef.lasso = predict(fit.lasso, s=bestlam.lasso, type = 'coefficients')[1:5,]
coef.lasso
```

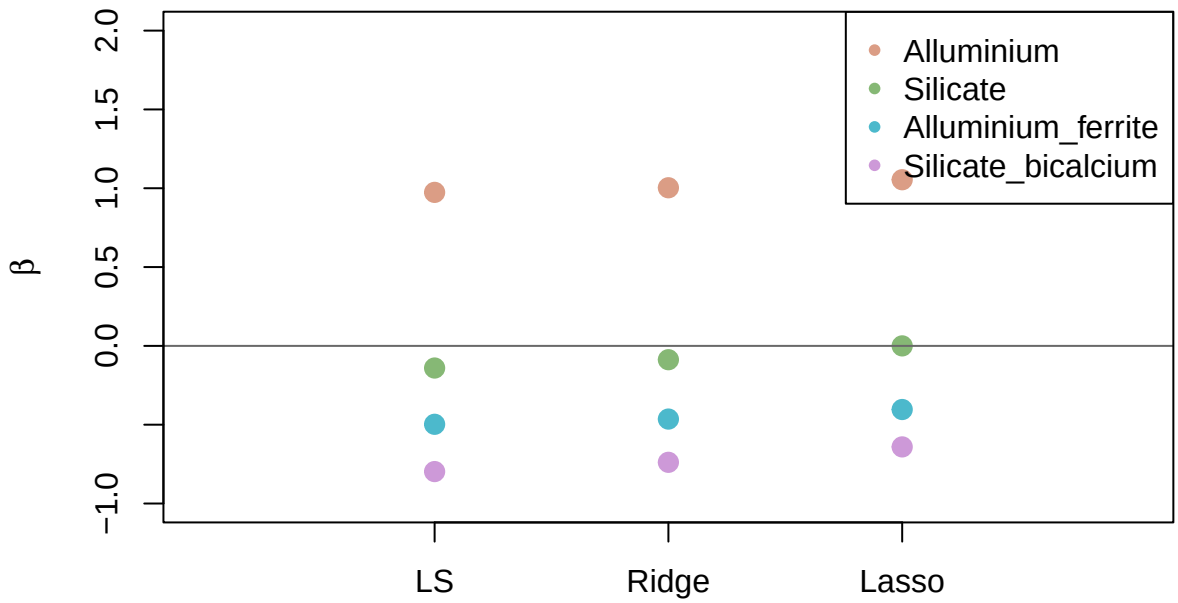
```
##        (Intercept)         Alluminium           Silicate Alluminium_ferrite
##        111.5358126          1.0540267          0.0000000         -0.4037632
## Silicate_bicalcium
##         -0.6408472
```

```
plot(fit.lasso,xvar='lambda',label=TRUE, col = col.ramp)
abline(v=log(bestlam.lasso))
```



10.2.4 Comparison

```
# Compare coefficients estimates for LS, Ridge and Lasso

plot(rep(0, dim(x)[2]), coef(lm(y~x))[-1], col=col.ramp, pch=20,
     xlim=c(-1,3), ylim=c(-1,2), xlab='', ylab=expression(beta),
     axes=F, cex=2)
points(rep(1, dim(x)[2]), coef.ridge[-1], col=col.ramp, pch=20, cex=2)
points(rep(2, dim(x)[2]), coef.lasso[-1], col=col.ramp, pch=20, cex=2)
abline(h=0, col='grey41', lty=1)
box()
axis(2)
axis(1, at=c(0,1,2), labels = c('LS', 'Ridge', 'Lasso'))
legend('topright', dimnames(x)[[2]], col =  col.ramp, pch=20, cex=1)
```




```r
# l2 norm
sqrt(sum((coef(lm(y~x))[-1])^2)) # OLS
```

```
## [1] 1.360671
```

```r
sqrt(sum((coef.ridge[-1])^2))    # ridge better
```

```
## [1] 1.332342
```

```r
# l1 norm
sum(abs(coef(lm(y~x))[-1])) # OLS
```

```
## [1] 2.409173
```

```r
sum(abs(coef.lasso[-1]))    # lasso better
```

```
## [1] 2.098637
```

```r
### Variable selection
result = lm(y ~ Alluminium + Silicate + Alluminium_ferrite + Silicate_bicalcium)
summary(result)
```

```
##
## Call:
## lm(formula = y ~ Alluminium + Silicate + Alluminium_ferrite +
##     Silicate_bicalcium)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0990 -1.7178  0.2919  1.2055  3.7526
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        124.4809    26.7557   4.653  0.00164 **
## Alluminium           0.9739     0.2835   3.435  0.00889 **
## Silicate            -0.1405     0.2891  -0.486  0.63996
```

```
## Alluminium_ferrite  -0.4974     0.2751  -1.808  0.10820
## Silicate_bicalcium  -0.7974     0.3214  -2.481  0.03805 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.484 on 8 degrees of freedom
## Multiple R-squared:  0.9818, Adjusted R-squared:  0.9727
## F-statistic:   108 on 4 and 8 DF,  p-value: 5.385e-07
```

```
result1 = lm(y ~ Alluminium + Alluminium_ferrite + Silicate_bicalcium)
summary(result1)
```

```
##
## Call:
## lm(formula = y ~ Alluminium + Alluminium_ferrite + Silicate_bicalcium)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9323 -1.8090  0.4806  1.1398  3.7771
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        111.68441    4.56248  24.479 1.52e-09 ***
## Alluminium           1.05185    0.22368   4.702  0.00112 **
## Alluminium_ferrite  -0.41004    0.19923  -2.058  0.06969 .
## Silicate_bicalcium  -0.64280    0.04454 -14.431 1.58e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.377 on 9 degrees of freedom
## Multiple R-squared:  0.9813, Adjusted R-squared:  0.975
## F-statistic: 157.3 on 3 and 9 DF,  p-value: 4.312e-08
```

```
result2 = lm(y ~ Alluminium + Silicate_bicalcium)
summary(result2)
```

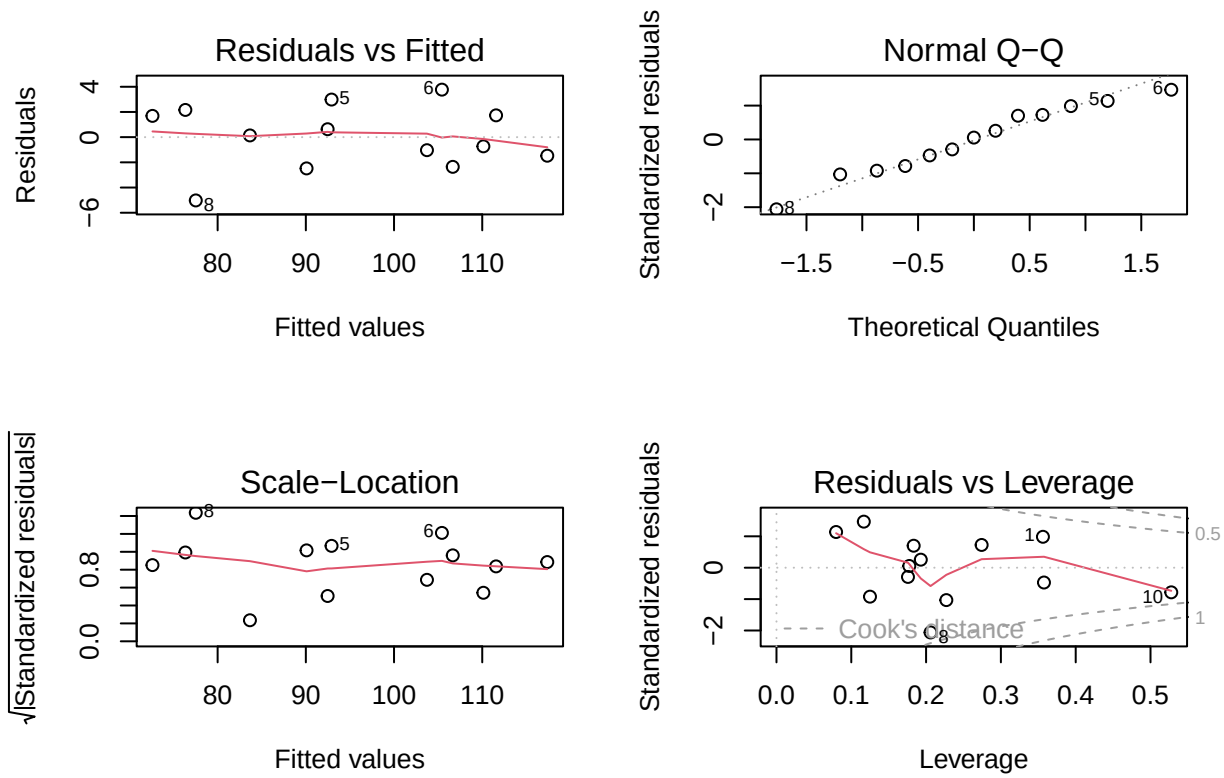
```
##
## Call:
## lm(formula = y ~ Alluminium + Silicate_bicalcium)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.0234 -1.4737  0.1371  1.7305  3.7701
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        103.09738    2.12398   48.54 3.32e-13 ***
## Alluminium           1.43996    0.13842   10.40 1.11e-06 ***
## Silicate_bicalcium  -0.61395    0.04864  -12.62 1.81e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.734 on 10 degrees of freedom
## Multiple R-squared:  0.9725, Adjusted R-squared:  0.967
## F-statistic: 176.6 on 2 and 10 DF,  p-value: 1.581e-08
```

```
# alternatively
#! We left out Silicate and Alluminium_ferrite, so we check if their
#! parameters can be considered both zeros, vs there is at least one still zero
linearHypothesis(result, rbind(c(0,0,1,0,0),
                               c(0,0,0,1,0)),
                 c(0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Silicate = 0
## Alluminium_ferrite = 0
##
## Model 1: restricted model
## Model 2: y ~ Alluminium + Silicate + Alluminium_ferrite + Silicate_bicalcium
##
```

```
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     10 74.762
## 2      8 49.378  2    25.384 2.0563 0.1903
```

```
#! high pvalue, we accept H0, those beta could be considered really 0, so we
#! can discard the full model

# diagnostics of the residuals

par(mfrow=c(2,2))
plot(result2)
```




```
shapiro.test(residuals(result2))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(result2)
## W = 0.9751, p-value = 0.9468
```

10.3 Model/Variable Selection

```
#-----------------------------------------------------------------------------
##### Model/Variable Selection
#####-------------
#! From the original variables we have to build a design matrix Z, but there
#! are lots of combinations:
#!  - which subset of features select
#!  - which transformations, functions, to apply (singularly or grouply ecc)

library(ISLR)

# Hitters dataset, about baseball players
# help(Hitters)
head(Hitters)
```

```
##                    AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Andy Allanson       293   66     1   30  29    14     1    293    66      1
## -Alan Ashby          315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis         479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga    321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133     19
##                    CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Andy Allanson        30   29     14      A        E     446      33     20
## -Alan Ashby          321  414    375      N        W     632      43     10
## -Alvin Davis         224  266    263      A        W     880      82     14
## -Andre Dawson        828  838    354      N        E     200      11      3
## -Andres Galarraga     48   46     33      N        E     805      40      4
## -Alfredo Griffin     501  336    194      A        W     282     421     25
##                    Salary NewLeague
## -Andy Allanson         NA         A
## -Alan Ashby         475.0         N
## -Alvin Davis        480.0         A
## -Andre Dawson       500.0         N
## -Andres Galarraga    91.5         N
## -Alfredo Griffin    750.0         A
```

```
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
dim(Hitters)
```

```
## [1] 322  20
```

```
# remove NA's
sum(is.na(Hitters$Salary))
```

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

```
Hitters = na.omit(Hitters)
dim(Hitters)
```

```
## [1] 263  20
```

```
sum(is.na(Hitters))
```

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

```
### Subset Selection Methods
library(leaps)
# help(regsubsets)
```

10.3.1 Exhaustive search

```
# Best Subset Selection: Exhaustive Search
#! we look at all the possible subsets, 2^r
regfit.full = regsubsets(Salary~., data=Hitters)
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
```

```
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
```

```r
#! That matrix tells that: if we want a model
#!  - with just 1 features, (row 1) select CRBI
#!  - with 2 features, (row 2) CRBI and Hits
#!  - ecc, with k features, look at variables selected at row k

#! nvmax: maximum number of variables/features up to which go
regfit.full = regsubsets(Salary~., data=Hitters, nvmax=19)
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: exhaustive
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
```

```
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 )  " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
##           CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```
reg.summary = summary(regfit.full)
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
reg.summary$which
```

```
##    (Intercept) AtBat  Hits HmRun  Runs   RBI Walks Years CAtBat CHits CHmRun
## 1         TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 2         TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 3         TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 4         TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 5         TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  FALSE FALSE  FALSE
## 6         TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  FALSE FALSE  FALSE
## 7         TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE  TRUE   TRUE
## 8         TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  FALSE FALSE   TRUE
## 9         TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE FALSE  FALSE
## 10        TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE FALSE  FALSE
## 11        TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE FALSE  FALSE
## 12        TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE   TRUE FALSE  FALSE
## 13        TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE   TRUE FALSE  FALSE
## 14        TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE   TRUE FALSE  FALSE
## 15        TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE   TRUE  TRUE  FALSE
## 16        TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE   TRUE  TRUE  FALSE
## 17        TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE   TRUE  TRUE  FALSE
## 18        TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE   TRUE  TRUE  FALSE
## 19        TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE   TRUE  TRUE   TRUE
##    CRuns  CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  FALSE  TRUE  FALSE   FALSE     FALSE   FALSE   FALSE  FALSE      FALSE
## 2  FALSE  TRUE  FALSE   FALSE     FALSE   FALSE   FALSE  FALSE      FALSE
## 3  FALSE  TRUE  FALSE   FALSE     FALSE    TRUE   FALSE  FALSE      FALSE
## 4  FALSE  TRUE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 5  FALSE  TRUE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 6  FALSE  TRUE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 7  FALSE FALSE  FALSE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 8   TRUE FALSE   TRUE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 9   TRUE  TRUE   TRUE   FALSE      TRUE    TRUE   FALSE  FALSE      FALSE
## 10  TRUE  TRUE   TRUE   FALSE      TRUE    TRUE    TRUE  FALSE      FALSE
```

```
## 11  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE  FALSE      FALSE
## 12  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE  FALSE      FALSE
## 13  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE      FALSE
## 14  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE      FALSE
## 15  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE      FALSE
## 16  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE      FALSE
## 17  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE       TRUE
## 18  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE       TRUE
## 19  TRUE  TRUE   TRUE    TRUE      TRUE    TRUE    TRUE   TRUE       TRUE
```

```
reg.summary$rsq   # r-squared
```

```
##  [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
##  [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
## [15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159
```

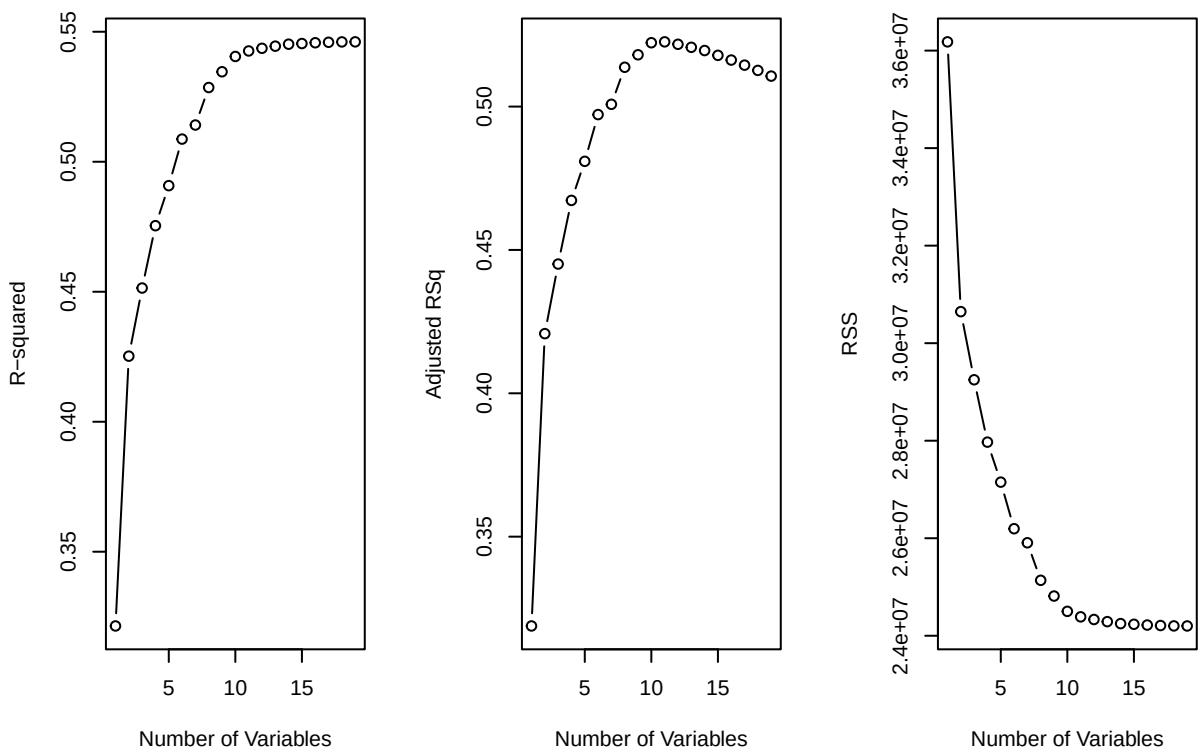
```
reg.summary$adjr2 # adjusted r-squared
```

```
##  [1] 0.3188503 0.4208024 0.4450753 0.4672734 0.4808971 0.4972001 0.5007849
##  [8] 0.5137083 0.5180572 0.5222606 0.5225706 0.5217245 0.5206736 0.5195431
## [15] 0.5178661 0.5162219 0.5144464 0.5126097 0.5106270
```

```
reg.summary$rss   # residual sum of squares
```

```
##  [1] 36179679 30646560 29249297 27970852 27149899 26194904 25906548 25136930
##  [9] 24814051 24500402 24387345 24333232 24289148 24248660 24235177 24219377
## [17] 24209447 24201837 24200700
```

```
par(mfrow=c(1,3))
plot(reg.summary$rsq,xlab="Number of Variables",ylab="R-squared",type="b")
plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="b")
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="b")
```




```
#! extract coefficient estimates associated with the models, providing
#! how much features we want to select
which.max(reg.summary$adjr2)
```

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

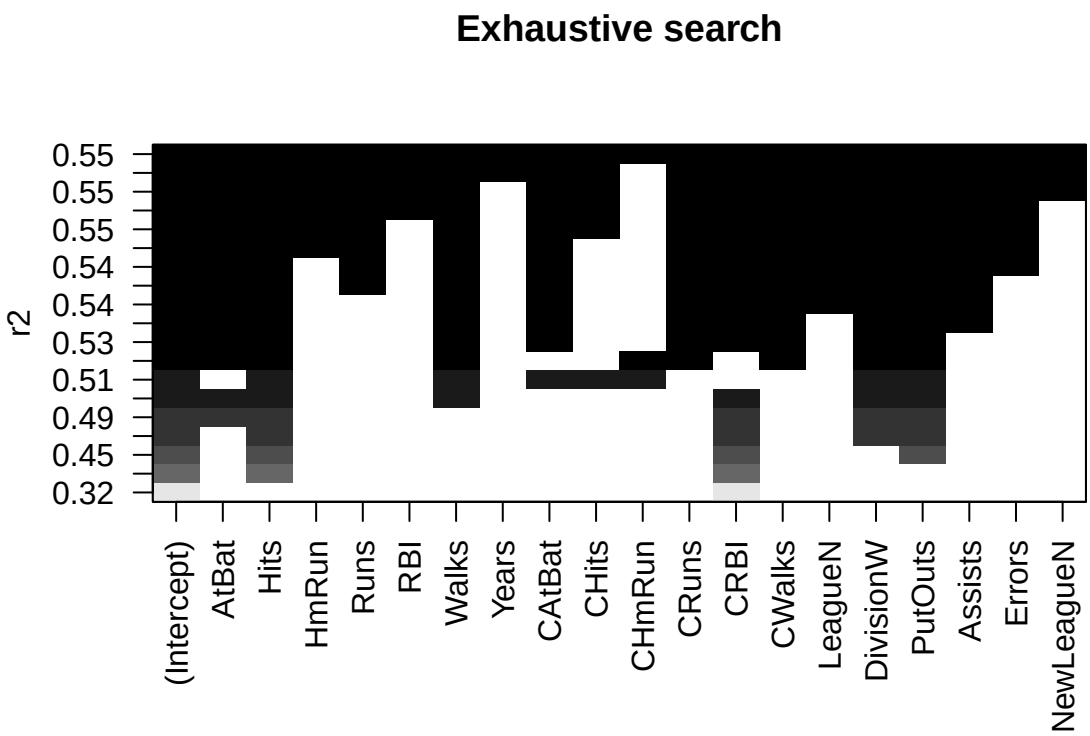
```r
coef(regfit.full,11)
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns
##  135.7512195   -2.1277482    6.9236994    5.6202755   -0.1389914    1.4553310
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##    0.7852528   -0.8228559   43.1116152 -111.1460252    0.2894087    0.2688277
```

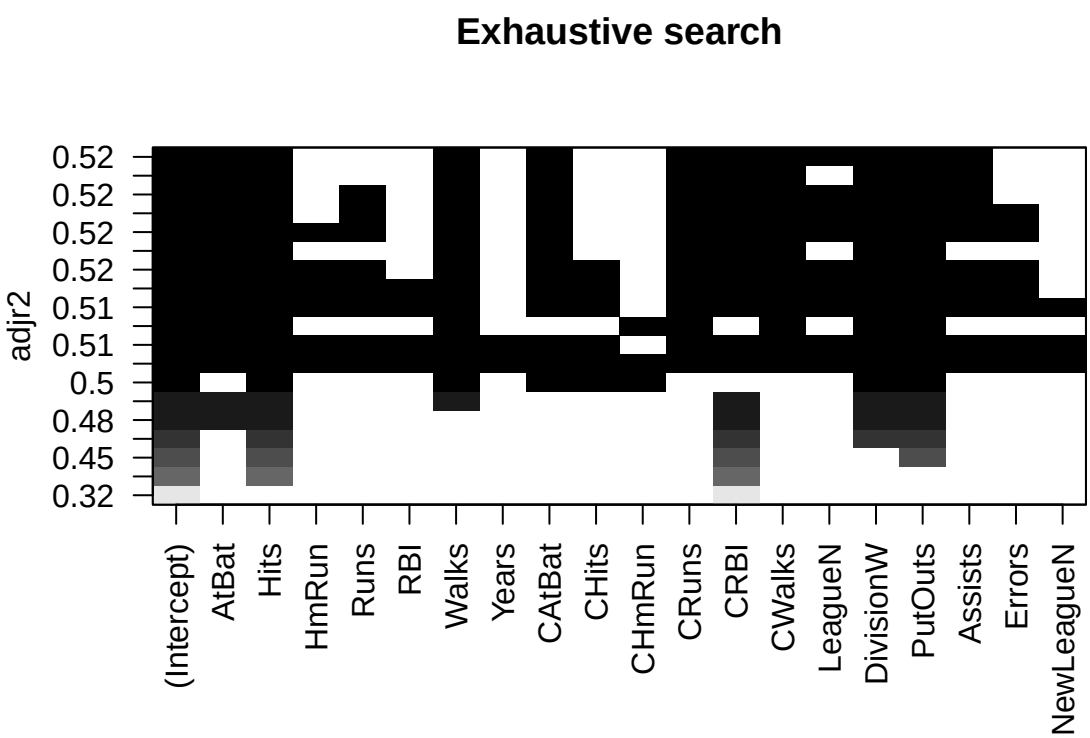
```r
coef(regfit.full,6)
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI    DivisionW
##   91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169 -122.9515338
##      PutOuts
##    0.2643076
```

```r
# graphical table of best subsets
# help(plot.regsubsets)

plot(regfit.full,scale="r2",main="Exhaustive search")
```




```r
plot(regfit.full,scale="adjr2",main="Exhaustive search")
```



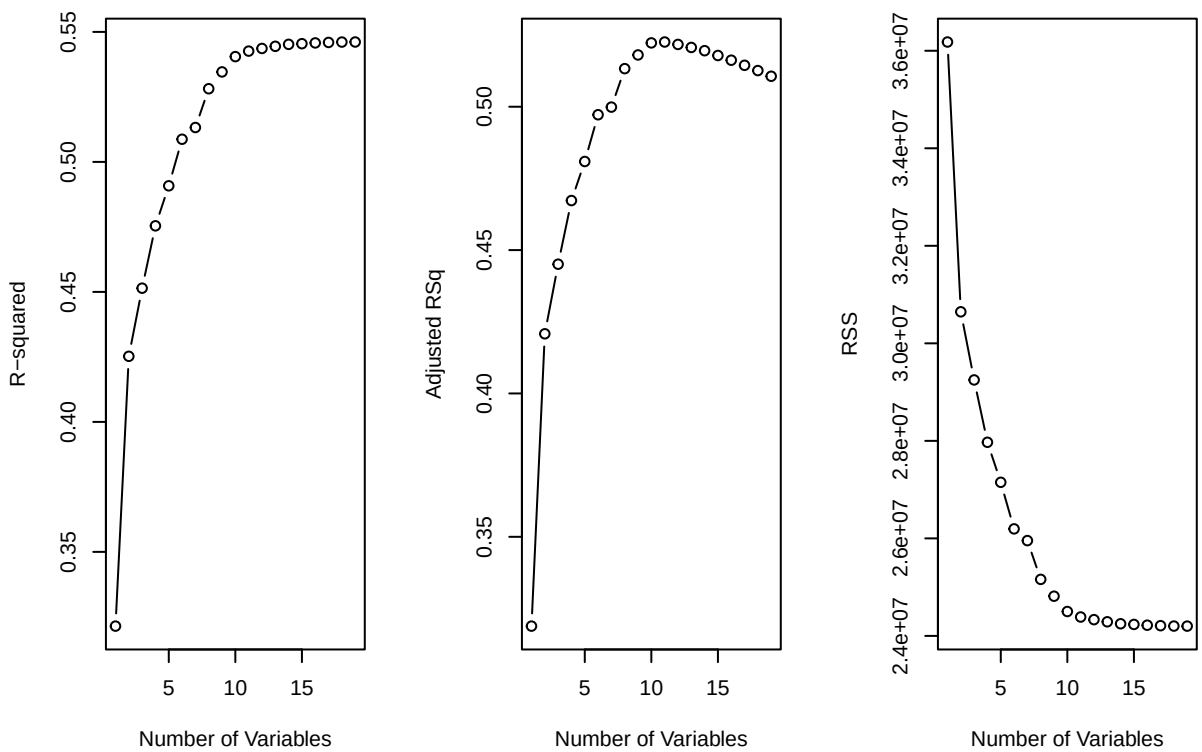
10.3.2 Forward and backward selection

```
# Forward and Backward Stepwise Selection
regfit.fwd = regsubsets(Salary~., data=Hitters, nvmax=19, method="forward")
#! backward: start with complete and remove the lowest significant feature
#! forward: add up features according to R^2, so when a feat enters it stays forever
summary(regfit.fwd)
```

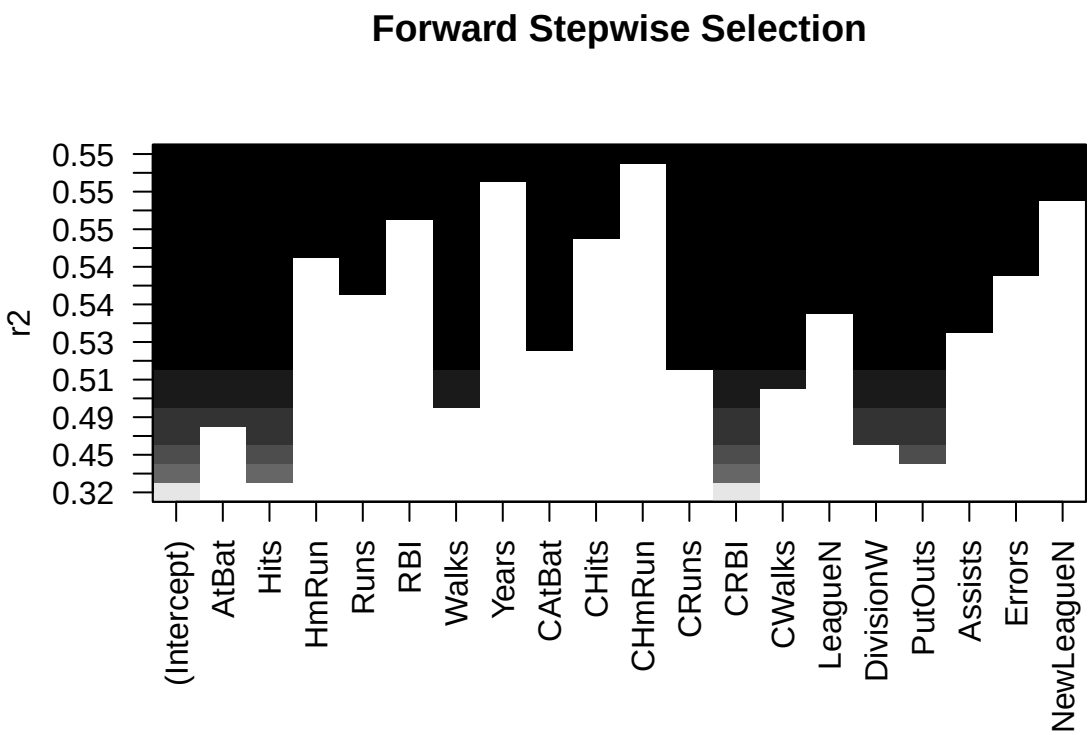
```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: forward
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
```

```
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
##           CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

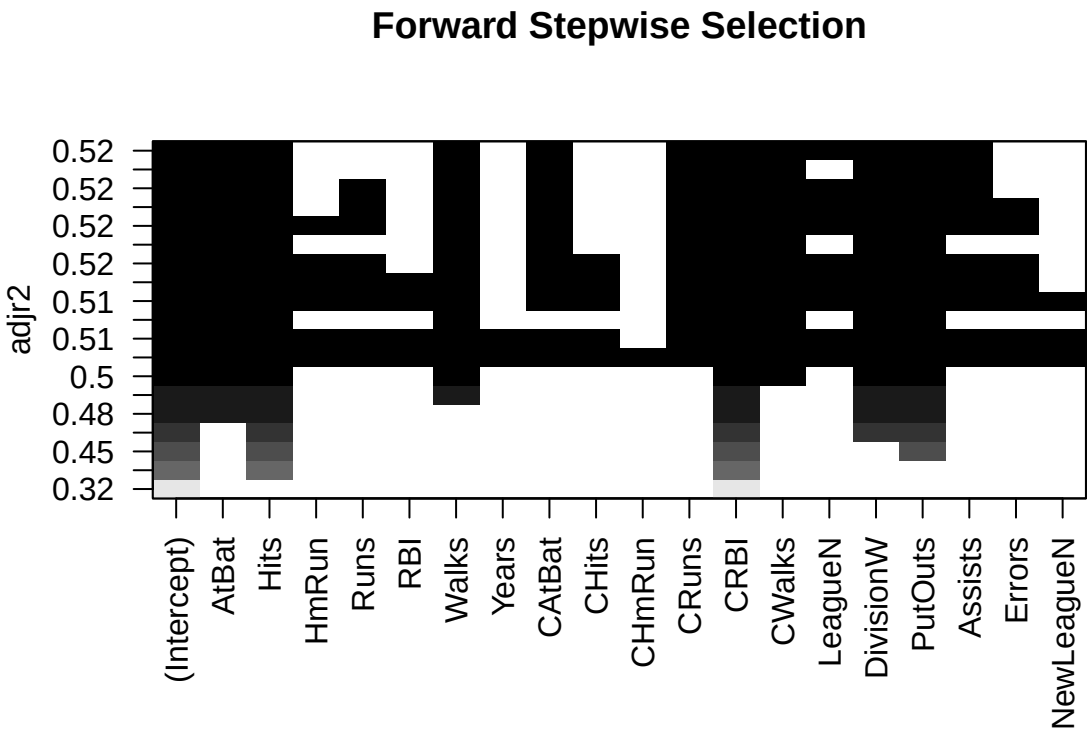
```
par(mfrow=c(1,3))
plot(summary(regfit.fwd)$rsq,xlab="Number of Variables",ylab="R-squared",type="b")
plot(summary(regfit.fwd)$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="b")
plot(summary(regfit.fwd)$rss,xlab="Number of Variables",ylab="RSS",type="b")
```



```r
plot(regfit.fwd,scale="r2",main="Forward Stepwise Selection")
```



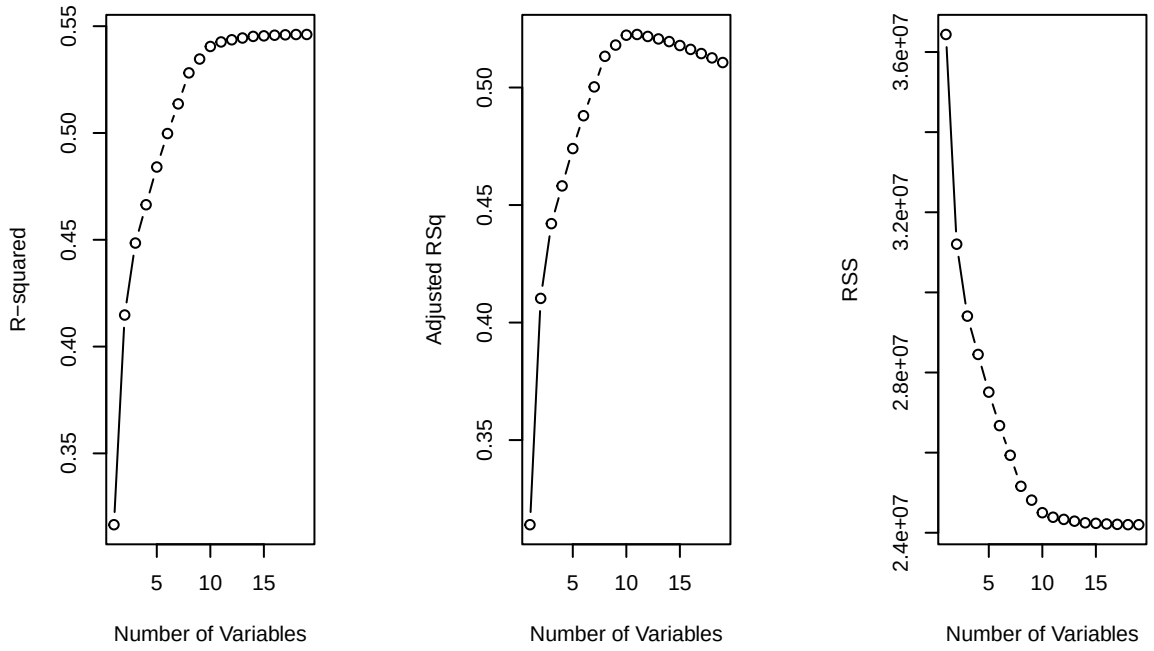

```r
plot(regfit.fwd,scale="adjr2",main="Forward Stepwise Selection")
```



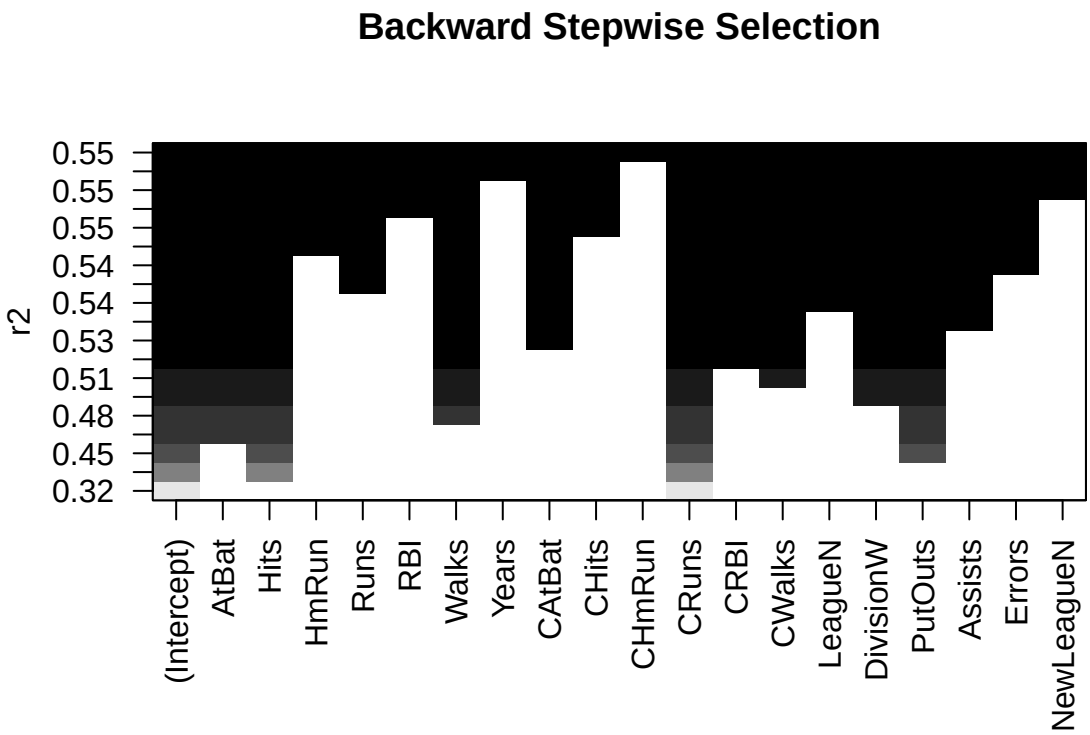
```r
regfit.bwd = regsubsets(Salary~.,data=Hitters,nvmax=19,method="backward")
summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "backward")
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: backward
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    "*"   " "
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    "*"   " "
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    "*"   " "
## 4  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    "*"   " "
## 5  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   " "
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   " "
## 7  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   " "
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
##           CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 5  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

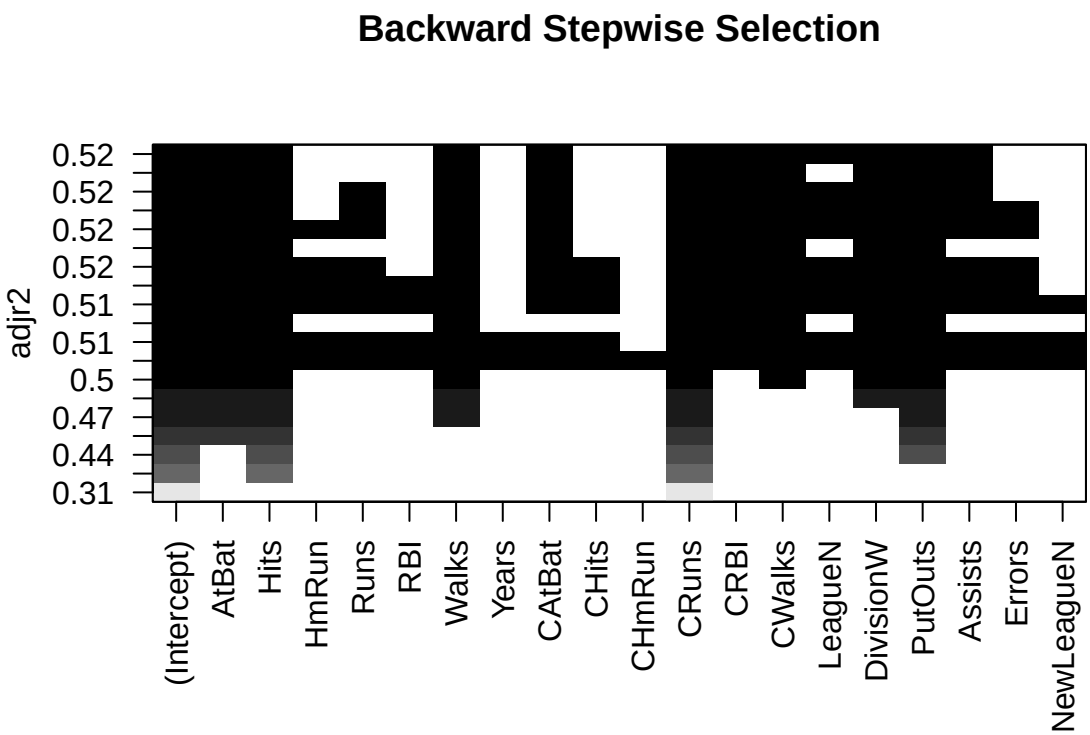
```r
par(mfrow=c(1,3))
plot(summary(regfit.bwd)$rsq,xlab="Number of Variables",ylab="R-squared",type="b")
plot(summary(regfit.bwd)$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="b")
plot(summary(regfit.bwd)$rss,xlab="Number of Variables",ylab="RSS",type="b")
```




```
plot(regfit.bwd,scale="r2",main="Backward Stepwise Selection")
```



```r
plot(regfit.bwd,scale="adjr2",main="Backward Stepwise Selection")
```




```r
coef(regfit.full,7) # Exhaustive search
```

```
##  (Intercept)         Hits        Walks       CAtBat        CHits       CHmRun
##   79.4509472    1.2833513    3.2274264   -0.3752350    1.4957073    1.4420538
##    DivisionW      PutOuts
## -129.9866432    0.2366813
```

```r
coef(regfit.fwd,7)  # Forward Stepwise Selection
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI       CWalks
##  109.7873062   -1.9588851    7.4498772    4.9131401    0.8537622   -0.3053070
##    DivisionW      PutOuts
## -127.1223928    0.2533404
```

```r
coef(regfit.bwd,7)  # Backward Stepwise Selection
```

```
##  (Intercept)        AtBat         Hits        Walks        CRuns       CWalks
##  105.6487488   -1.9762838    6.7574914    6.0558691    1.1293095   -0.7163346
##    DivisionW      PutOuts
## -116.1692169    0.3028847
```

10.3.3 Comparison

```r
### Choosing among models using the k-fold cross-validation approach
### (exhaustive search)
k = 10

set.seed(1)
folds = sample(1:k, nrow(Hitters), replace=TRUE)
folds
```

```
##   [1]  9  4  7  1  2  7  2  3  1  5  5 10  6 10  7  9  5  5  9  9  5  5  2 10  9
##  [26]  1  4  3  6 10 10  6  4  4 10  9  7  6  9  8  9  7  8  6 10  7  3 10  6  8
##  [51]  2  2  6  6  1  3  3  8  6  7  6  8  7  1  4  8  9  9  7  4  7  6  1  5  6
##  [76]  1  9  7  7  3  6  2 10 10  7  3  2 10  1 10 10  8 10  5  7  8  5  6  8  1
## [101]  3 10  3  1  6  6  4  9  5  1  3  6  3  7  3  3  1  9  2  8  6  1  2  7  7
## [126]  4  9  8  3  5  3  4  2  1  7  9 10 10  2  2  3  1  2  3  3  3  8  9  2 10
## [151]  8 10  4  5  9  5  7  5  6  4  2  1  3  8  9  6  1  4  5  9  5  8  4  1  9
## [176]  5  1  5  4 10 10  9  8  5  5  6  6  2  2  8  4 10  8  5  5  8  8  7  4  4
## [201]  1 10  4  9  9  9  9  6  6  4  3  3  9  9  7  9  5  7  4  4 10  8  1 10  2
## [226] 10  1  1  4  5  5  6  9  8  5  1  2  1  8  5  8 10  7  7  2  9  4  2  5  2
## [251]  4  3  6  9  7  5  5  1  1 10  1  3 10
```

```
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
## 28 22 24 24 30 26 25 24 31 29
```

```
# function that performs the prediction for regsubsets
predict.regsubsets = function(object, newdata, id) {
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id=id)
  xvars = names(coefi)
  mat[,xvars]%*%coefi
}

cv.errors = matrix(NA, k, 19, dimnames=list(NULL, paste(1:19)))

for(j in 1:k) {
  best.fit = regsubsets(Salary~., data=Hitters[folds!=j,], nvmax=19)
  for(i in 1:19) {
    pred = predict.regsubsets(best.fit, Hitters[folds==j,], id=i)
    cv.errors[j,i] = mean( (Hitters$Salary[folds==j]-pred)^2 )
  }
}

cv.errors
```

```
##               1         2         3         4         5         6         7
##  [1,]  98623.24 115600.61 120884.31 113831.63 120728.51 122922.93 155507.25
##  [2,] 155320.11 100425.87 168838.35 159729.47 145895.71 123555.25 119983.35
##  [3,] 124151.77  68833.50  69392.29  77221.37  83802.82  70125.41  68997.77
##  [4,] 232191.41 279001.29 294568.10 288765.81 276972.83 260121.22 276413.09
##  [5,] 115397.35  96807.44 108421.66 104933.55  99561.69  86103.05  89345.61
##  [6,] 103839.30  75652.50  69962.31  58291.91  65893.45  64215.56  65800.88
##  [7,]  85793.95  78506.34  80541.35  84213.50  87140.78  80669.98  86247.85
##  [8,] 273084.54 235423.66 230706.43 205624.81 223867.35 205559.00 206556.77
##  [9,] 178316.69 163857.60 142998.75 120697.54 115261.58 108791.71 102320.25
## [10,] 131492.54  95111.58 104956.38  96978.66  91377.54  73322.28  71687.88
##               8         9        10        11        12        13        14
##  [1,] 137753.36 149198.01 153332.89 155702.91 155842.88 158755.87 156037.17
##  [2,]  96609.16  99057.32  80375.78  91290.74  92292.69 100498.84 101562.45
##  [3,]  64143.70  65813.14  65120.27  68160.94  70263.77  69765.81  68987.54
##  [4,] 259923.88 270151.18 263492.31 259154.53 269017.80 265468.90 269666.65
##  [5,]  87693.15  91631.88  88763.37  89801.07  91070.44  92429.43  92821.15
##  [6,]  61413.45  60200.70  59599.54  59831.90  60081.48  59662.51  60618.91
##  [7,]  89643.01  92081.37  89057.16  88102.28  90885.98  95025.99  99172.32
##  [8,] 182678.23 179783.18 179916.36 173790.82 180508.48 185424.38 183257.89
##  [9,]  89418.23  84366.23  80188.72  80665.49  82509.05  85078.50  89243.38
## [10,]  66524.07  63281.82  62320.54  66011.39  65086.73  66097.41  73444.48
##               15        16        17        18        19
##  [1,] 157739.46 155548.96 156688.01 156860.92 156976.98
##  [2,] 104621.38 100922.27 102198.69 105318.26 106064.89
##  [3,]  69471.32  69294.21  69199.91  68866.84  69195.74
##  [4,] 265518.87 267240.44 267771.74 267670.66 267717.80
##  [5,]  95849.81  96513.70  95209.20  94952.21  94951.70
##  [6,]  62540.03  62776.81  62717.77  62354.97  62268.97
##  [7,]  99314.04 100192.99 100302.79  99772.60 100659.75
##  [8,] 183331.01 185159.62 183643.63 182587.35 183436.78
##  [9,]  90478.82  91782.20  91723.08  91140.55  91344.82
## [10,]  72351.39  71311.99  71393.23  71333.19  71417.75
```

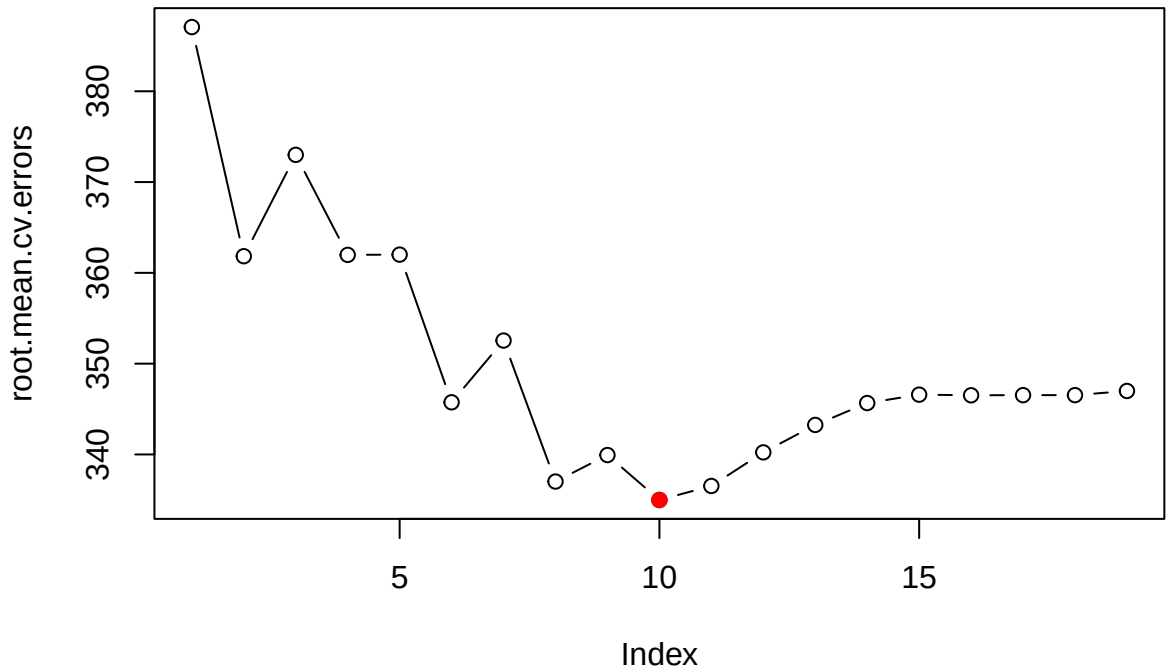
```r
root.mean.cv.errors = sqrt(apply(cv.errors,2,mean)) # average over the columns
root.mean.cv.errors
```

```
##        1        2        3        4        5        6        7        8
## 387.0673 361.8315 372.9973 361.9790 362.0086 345.7436 352.5423 337.0164
##        9       10       11       12       13       14       15       16
## 339.9360 334.9876 336.5282 340.2292 343.2503 345.6605 346.5856 346.5174
##       17       18       19
## 346.5325 346.5339 346.9921
```

```r
plot(root.mean.cv.errors,type='b')
which.min(root.mean.cv.errors)
```

```
## 10
## 10
```

```r
points(which.min(root.mean.cv.errors),
       root.mean.cv.errors[which.min(root.mean.cv.errors)], col='red',pch=19)
```




```r
# estimation on the full dataset
reg.best = regsubsets(Salary~., data=Hitters, nvmax=19)
coef(reg.best,10)
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns
##  162.5354420   -2.1686501    6.9180175    5.7732246   -0.1300798    1.4082490
##         CRBI       CWalks    DivisionW      PutOuts      Assists
##    0.7743122   -0.8308264 -112.3800575    0.2973726    0.2831680
```

```r
### k-fold cross validation after model selection (WRONG WAY!)
#! Before we were looking at the best model with 1,2,..,19 features wrt each
#! cv dataset.
#! Here we first select the variables more interesting for the data, and then
#! we perform cv on it.

best.fit = regsubsets(Salary~., data=Hitters, nvmax=19)
summary(best.fit)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: exhaustive
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 )  " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
##           CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```r
cv.errors_wrong = matrix(NA, k, 19, dimnames=list(NULL, paste(1:19)))
for(j in 1:k)
  for(i in 1:19){
    covariate = which(summary(best.fit)$which[i,-c(1,19)])
    mod = lm(Salary~., data=Hitters[folds!=j, c(covariate,19)])
    pred = predict(mod,Hitters)[folds==j]
    cv.errors_wrong[j,i] = mean( (Hitters$Salary[folds==j]-pred)^2)
  }
```

```r
cv.errors_wrong
```

```
##               1         2         3         4         5         6         7
##  [1,]  98623.24 100025.08 108594.60 113831.63 109928.48 122922.93 133223.96
##  [2,] 126554.96 100425.87  90243.27  85000.05  79143.84  75732.01 107282.84
##  [3,] 110257.03  68833.50  53791.82  55087.62  52797.37  59367.05  60602.13
##  [4,] 232191.41 279001.29 294568.10 272134.61 270315.96 260121.22 248007.23
##  [5,] 115397.35  96807.44 108421.66 104933.55  99561.69  86103.05  85638.42
##  [6,] 103839.30  75652.50  62963.06  58291.91  57685.35  64215.56  61562.62
##  [7,]  85793.95  78506.34  80541.35  84213.50  87140.78  80669.98  74185.42
##  [8,] 273084.54 215931.04 194759.90 191579.63 175494.24 170802.69 199833.42
##  [9,] 154315.15 120983.23 109399.88 102000.51  96120.50  91172.85  88804.98
## [10,] 131492.54  95111.58  79491.06  70383.63  76217.30  73322.28  66180.97
##               8         9        10        11        12        13        14
##  [1,] 131554.97 126071.10 131818.11 128937.50 130158.61 130158.61 129333.46
##  [2,]  71251.97  74656.56  80375.78  81379.62  81788.48  81788.48  84078.46
##  [3,]  64143.70  65813.14  65120.27  68160.94  70263.77  70263.77  69619.78
##  [4,] 252181.50 252222.15 245373.30 243108.69 258276.16 258276.16 257687.96
##  [5,]  82807.54  81658.21  88763.37  89801.07  89072.77  89072.77  89338.77
##  [6,]  58787.34  60200.70  59599.54  59831.90  59339.64  59339.64  60199.38
##  [7,]  84872.47  84106.87  89057.16  88102.28  87260.53  87260.53  90250.98
##  [8,] 182678.23 179783.18 172950.31 173790.82 172480.25 172480.25 172602.36
##  [9,]  88048.51  84366.23  80188.72  80665.49  82209.73  82209.73  82106.64
## [10,]  62257.89  63281.82  62320.54  61619.06  65086.73  65086.73  64647.76
##              15        16        17        18        19
##  [1,] 134743.16 134745.29 140117.06 141918.84 156803.71
##  [2,]  98090.98  99941.11 100768.60 102719.83 103028.27
##  [3,]  69720.79  69408.24  68622.08  68690.01  68863.06
##  [4,] 258337.65 258871.49 259389.89 261059.13 261023.89
##  [5,]  90512.85  93016.76  94603.75  95209.20  95199.86
##  [6,]  60557.16  60207.20  60975.37  61103.98  62372.10
##  [7,]  90045.24  89667.03  93765.08  98890.97 100069.89
##  [8,] 172777.92 175270.52 174182.90 179569.37 182587.35
##  [9,]  82636.23  82639.90  85552.16  85420.29  91140.55
## [10,]  65668.34  69228.92  68102.60  68237.91  71532.13
```

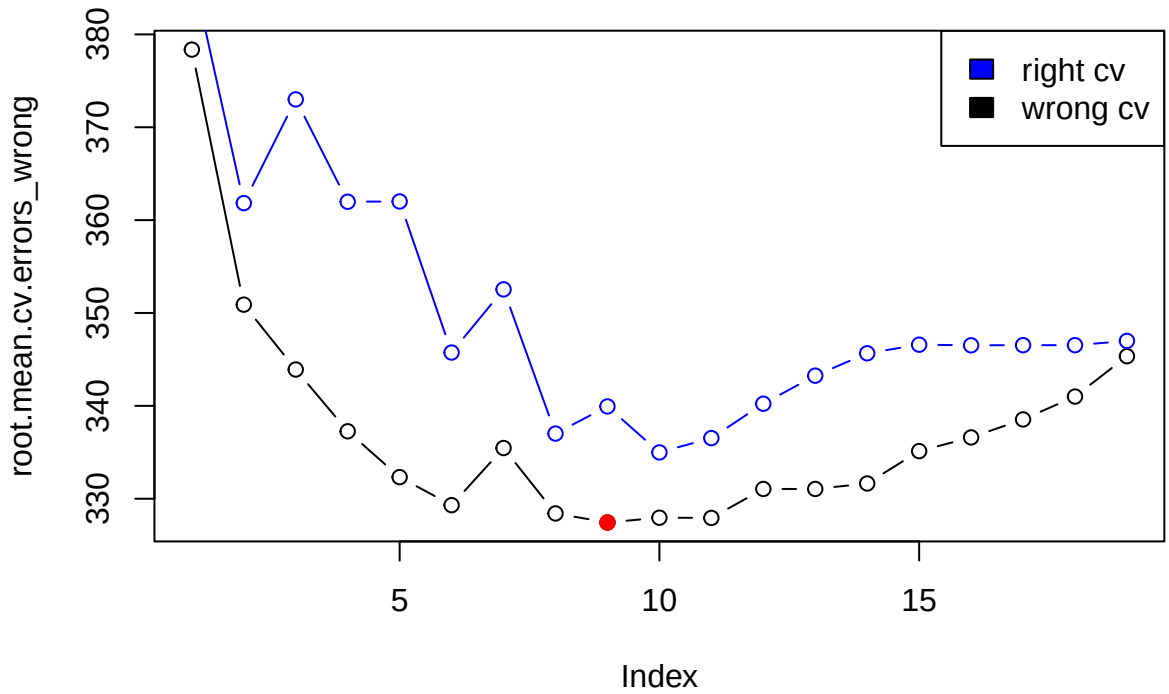
```r
root.mean.cv.errors_wrong = sqrt(apply(cv.errors_wrong,2,mean)) # average over the columns
root.mean.cv.errors_wrong
```

```
##        1        2        3        4        5        6        7        8
## 378.3582 350.8957 343.9149 337.2620 332.3260 329.3068 335.4582 328.4180
##        9       10       11       12       13       14       15       16
## 327.4385 327.9584 327.9325 331.0493 331.0493 331.6422 335.1254 336.6001
##       17       18       19
## 338.5380 341.0014 345.3434
```

```r
plot(root.mean.cv.errors_wrong,type='b')
which.min(root.mean.cv.errors_wrong)
```

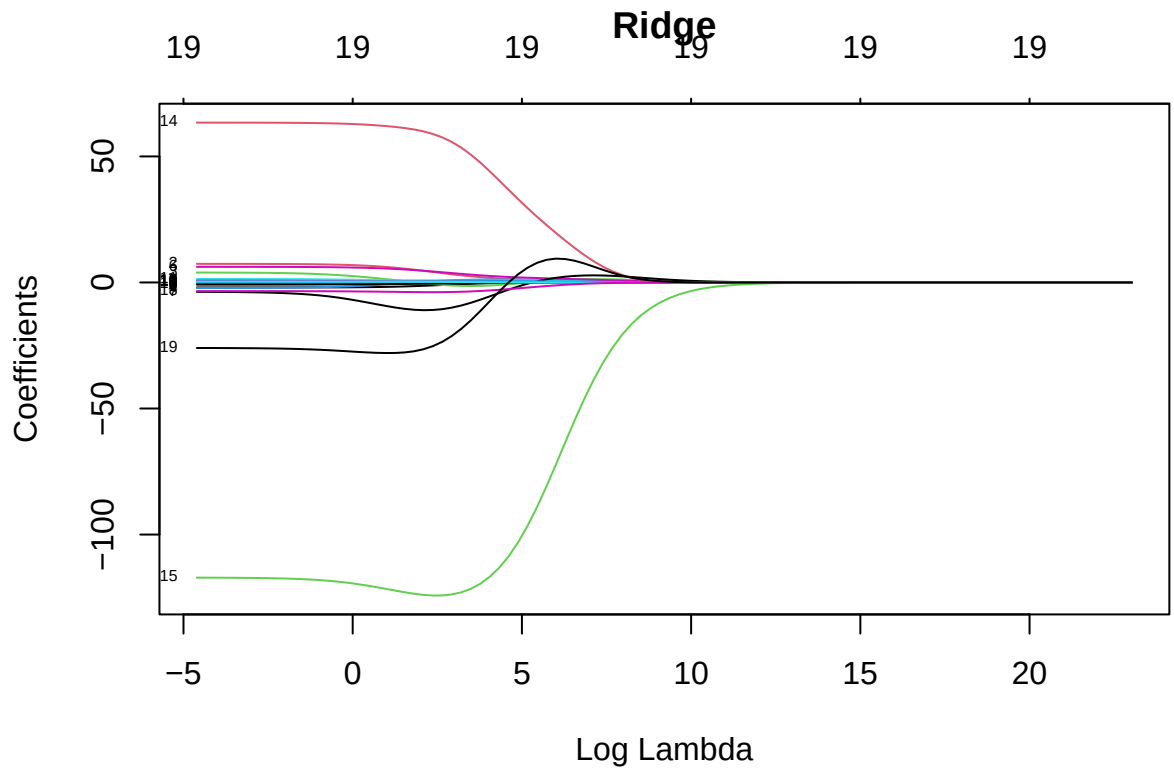
```
## 9
## 9
```

```r
points(which.min(root.mean.cv.errors_wrong),
       root.mean.cv.errors_wrong[which.min(root.mean.cv.errors_wrong)],
       col='red',pch=19)
points(root.mean.cv.errors,type='b',col='blue')
legend("topright",c("right cv","wrong cv"),fill=c("blue","black"))
```



10.3.4 Ridge and lasso regression

```
### Ridge and Lasso regression with glmnet
x = model.matrix(Salary~.,Hitters)[,-1] # predictor matrix
y = Hitters$Salary # response
grid = 10^seq(10,-2,length=100) # grid of lambda

# Ridge regression
ridge.mod = glmnet(x,y,alpha=0,lambda=grid)

plot(ridge.mod,xvar='lambda',label=TRUE, main="Ridge")
```



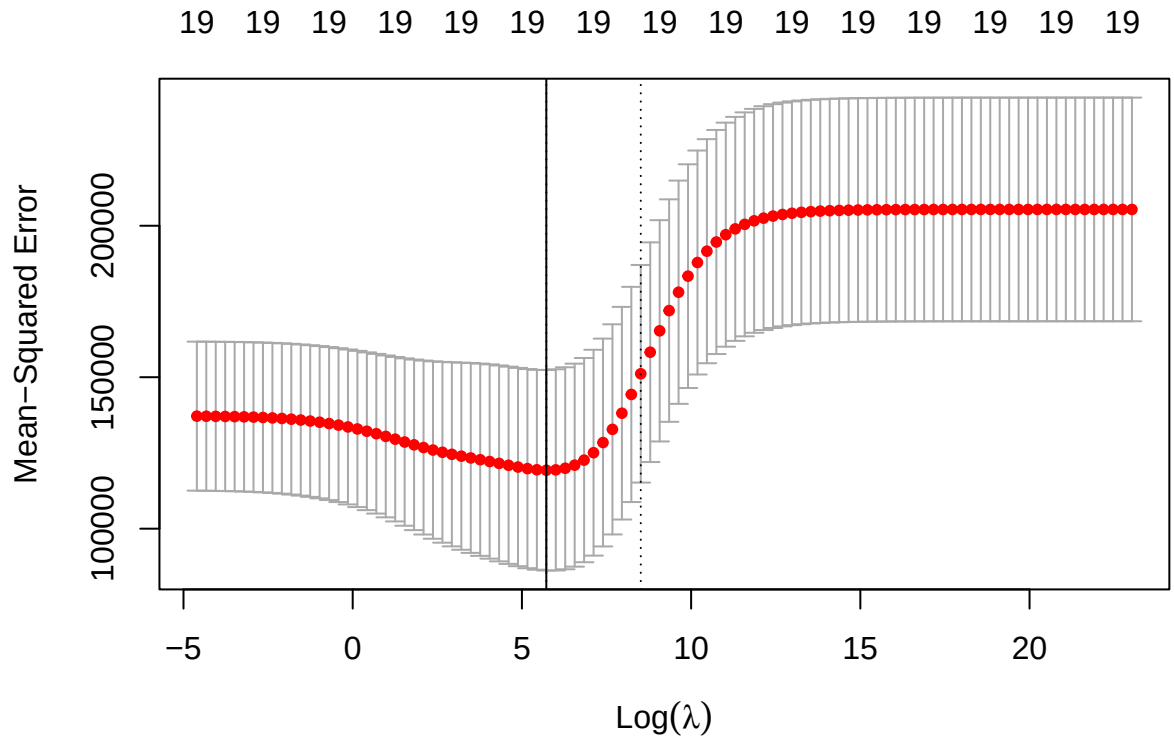

```r
# choosing the parameter lambda
set.seed(123)
cv.out = cv.glmnet(x,y,alpha=0,nfold=3,lambda=grid)

plot(cv.out)

bestlam.ridge = cv.out$lambda.min
bestlam.ridge
```

```
## [1] 305.3856
```

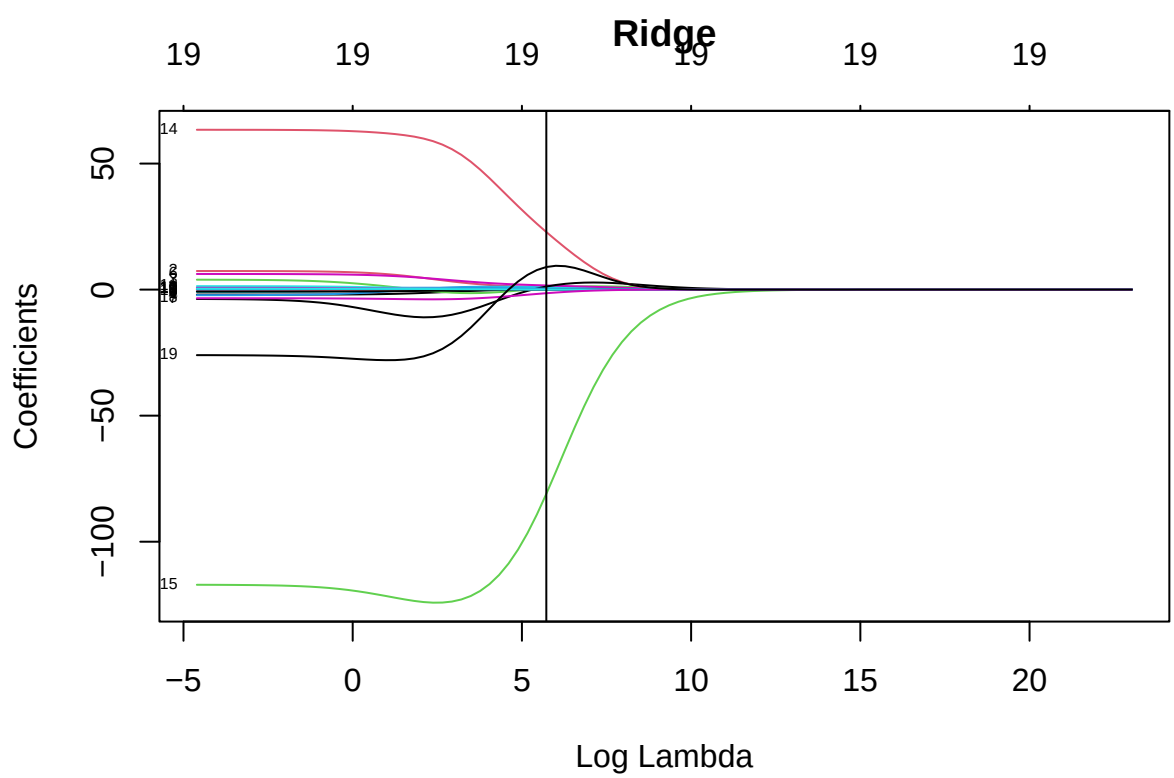
```r
log(bestlam.ridge)
```

```
## [1] 5.721575
```

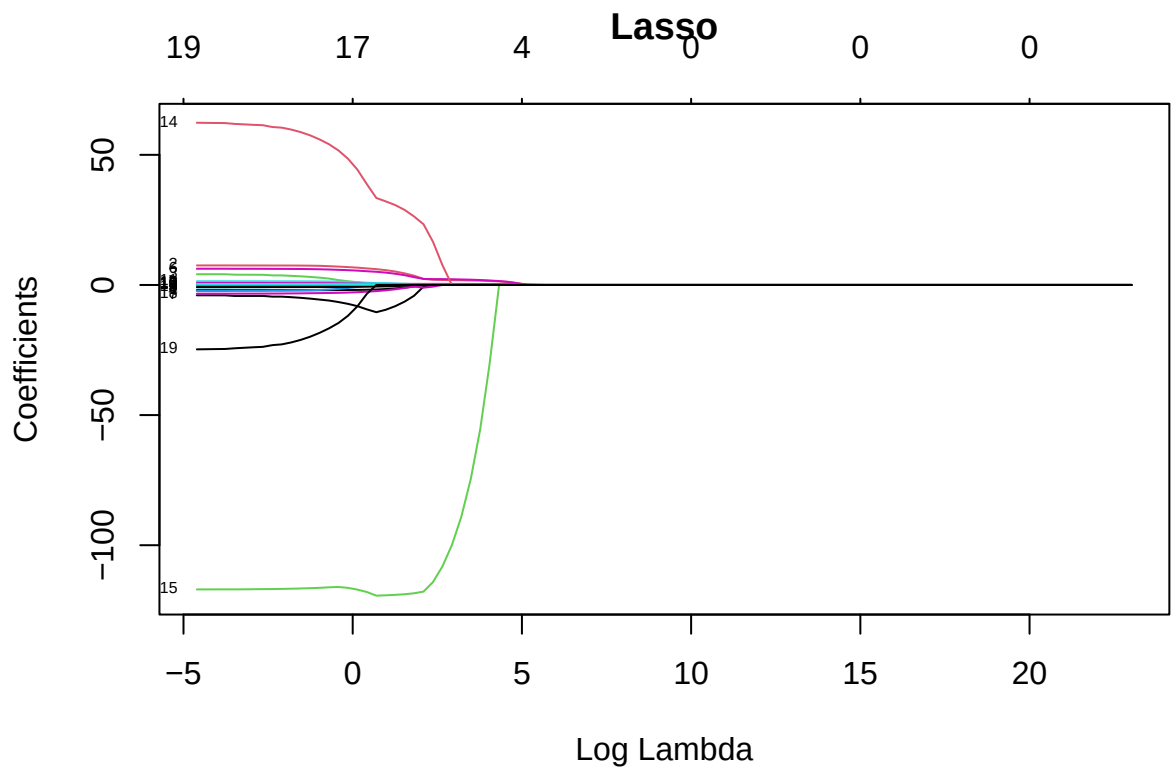
```r
abline(v=log(bestlam.ridge))
```




```
plot(ridge.mod,xvar='lambda',label=TRUE, main="Ridge")
abline(v=log(bestlam.ridge))
```



```r
# Lasso regression
lasso.mod = glmnet(x,y,alpha=1,lambda=grid)

plot(lasso.mod,xvar='lambda',label=TRUE, main="Lasso")
```



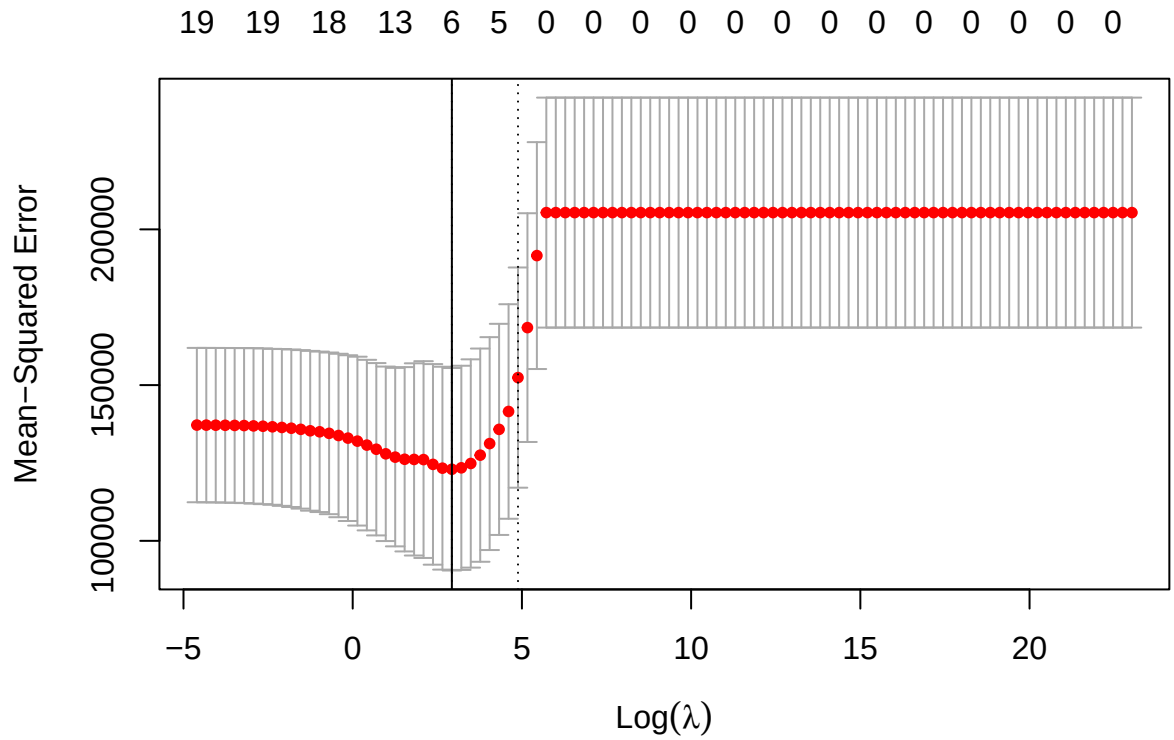

```r
# choosing the parameter lambda
set.seed(123)
cv.out = cv.glmnet(x,y,alpha=1,nfold=3,lambda=grid)

plot(cv.out)

bestlam.lasso = cv.out$lambda.min
bestlam.lasso
```

```
## [1] 18.73817
```

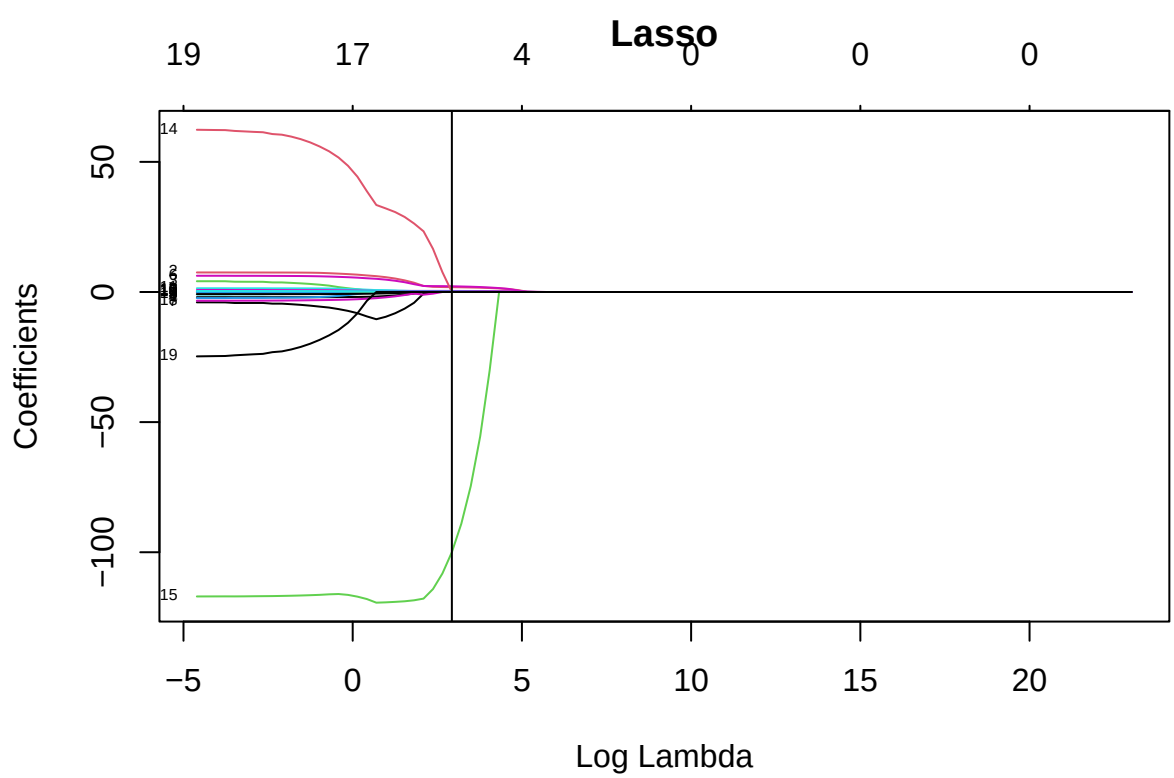
```r
log(bestlam.lasso)
```

```
## [1] 2.930563
```

```r
abline(v=log(bestlam.lasso))
```




```
plot(lasso.mod,xvar='lambda',label=TRUE, main="Lasso")
abline(v=log(bestlam.lasso))
```



10.3.5 Comparison

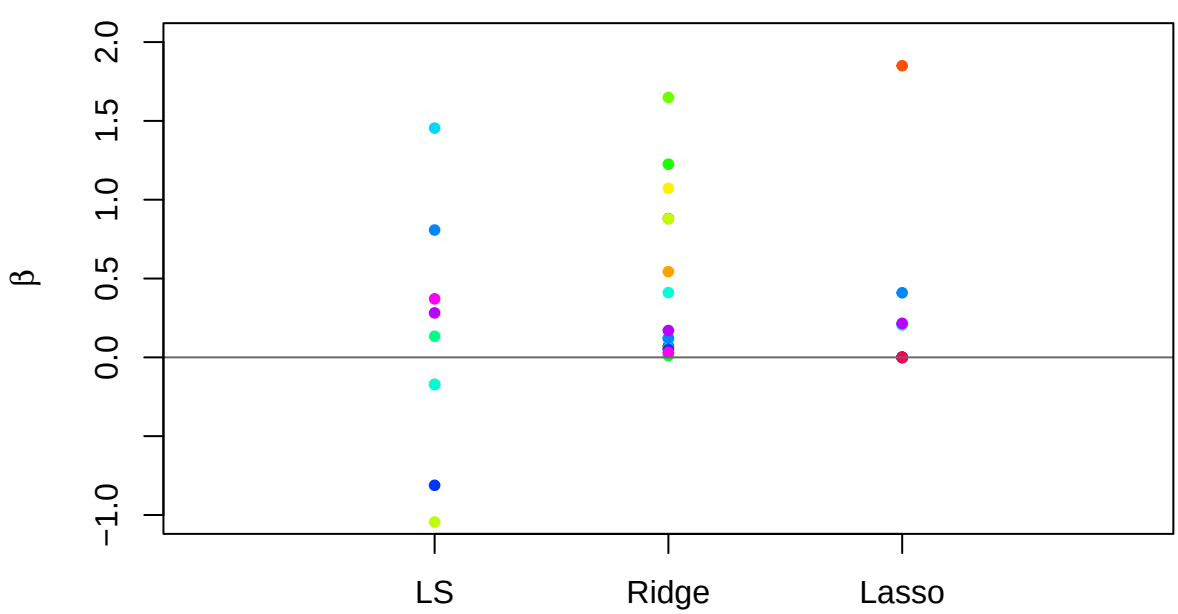
```r
# Compare coefficients estimates for LS, Ridge and Lasso

coef.ridge = predict(ridge.mod, s=bestlam.ridge, type = 'coefficients')[1:20,]
coef.lasso = predict(lasso.mod, s=bestlam.lasso, type = 'coefficients')[1:20,]
coef.ridge
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  13.69965470   0.07214526   0.88059238   0.54367980   1.07217873   0.87912033
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.64903804   1.22528239   0.01132160   0.05819360   0.41084586   0.11626187
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.12363705   0.05071118  22.84517646 -81.01744688   0.17009772   0.03096459
##       Errors   NewLeagueN
##  -1.42769526   8.96604057
```

```r
coef.lasso
```

```
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
##  24.6392478   0.0000000   1.8499430   0.0000000   0.0000000   0.0000000
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns
##   2.1961005   0.0000000   0.0000000   0.0000000   0.0000000   0.2058504
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
##   0.4095924   0.0000000   0.0000000 -99.9736534   0.2158902   0.0000000
##      Errors  NewLeagueN
##   0.0000000   0.0000000
```

```r
plot(rep(0, dim(x)[2]), coef(lm(y~x))[-1], col=rainbow(dim(x)[2]), pch=20,
     xlim=c(-1,3), ylim=c(-1,2), xlab='', ylab=expression(beta),
     axes=F)
points(rep(1, dim(x)[2]), coef.ridge[-1], col=rainbow(dim(x)[2]), pch=20)
points(rep(2, dim(x)[2]), coef.lasso[-1], col=rainbow(dim(x)[2]), pch=20)
abline(h=0, col='grey41', lty=1)
box()
axis(2)
axis(1, at=c(0,1,2), labels = c('LS', 'Ridge', 'Lasso'))
```



```r
# l2 norm
sqrt(sum((coef(lm(y~x))[-1])^2)) # LS
```

```
## [1] 135.4153
```

```r
sqrt(sum((coef.ridge[-1])^2))    # ridge
```

```
## [1] 84.70897
```

```r
# l1 norm
sum(abs(coef(lm(y~x))[-1])) # LS
```

```
## [1] 238.7295
```

```r
sum(abs(coef.lasso[-1]))    # lasso
```

```
## [1] 104.851
```

10.4 Logistic regression

```r
################################################################################
## Logistic Regression ##
# help(glm)

##  Pb2 of 23/07/2009  ##
# A store of an appliance chain sells two types of televisions: with 4:3
# screen and with 16:9 screen. The TV.txt file shows the number of
# televisions sold annually for both types of televisions from 1999
# to 2008. By introducing an appropriate logistic model and estimating
# the parameters with the maximum likelihood method:
# a) comment the residual deviance by comparing it with that of the model
#    without regressor.
# Assuming that the store is representative of the Italian situation:
# b) provide a pointwise estimate for the proportion of 16:9 televisions
#    sold in Italy in 2009;
# c) provide a pointwise estimate for the year in which sales of 16:9
#    televisions exceeded those of 4:3;
# d) provide a pointwise estimate for the year in which 16:9 televisions
#    will cover (or have covered) 99% of the Italian market.

TV = read.table('Lab10/TV.txt', header=T)
head(TV)
```

```
##       Anno         Tipo
## 1 2006.549   sedicinoni
## 2 2007.082   sedicinoni
## 3 2002.237   sedicinoni
## 4 2006.052   sedicinoni
## 5 2000.754 quattroterzi
## 6 2007.529   sedicinoni
```

```r
dim(TV)
```

```
## [1] 198   2
```

```r
# a) comment the residual deviance by comparing it with that of the model
#    without regressor.
fit = glm(factor(Tipo) ~ Anno, data=TV, family='binomial')
summary(fit)
```

```
##
## Call:
## glm(formula = factor(Tipo) ~ Anno, family = "binomial", data = TV)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5109  -0.4890   0.1591   0.3787   2.0030
##
```

```
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1808.2102   264.4591  -6.837 8.06e-12 ***
## Anno            0.9035     0.1321   6.838 8.01e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 235.86  on 197  degrees of freedom
## Residual deviance: 126.57  on 196  degrees of freedom
## AIC: 130.57
##
## Number of Fisher Scoring iterations: 6
```

```
cols = fun_colori(2)
plot(TV$Anno, as.numeric(factor(TV$Tipo))-1, col=ifelse(TV$Tipo=="sedicinoni",cols[1],cols[2]), pch=19)
lines(seq(1997, 2010, by=0.1),
      predict(fit, data.frame(Anno=seq(1997,2010,by=0.1)), type='response'))

# null model
Freq.tot = table(TV$Tipo)[2]/ (table(TV$Tipo)[1] + table(TV$Tipo)[2])
abline(h = Freq.tot, col='blue', lty=2)

# Assuming that the store is representative of the Italian situation:
# b) provide a pointwise estimate for the proportion of 16:9 televisions
#    sold in Italy in 2009;
predict(fit, data.frame(Anno=2009), type='response')
```

```
##         1
## 0.9990309
```

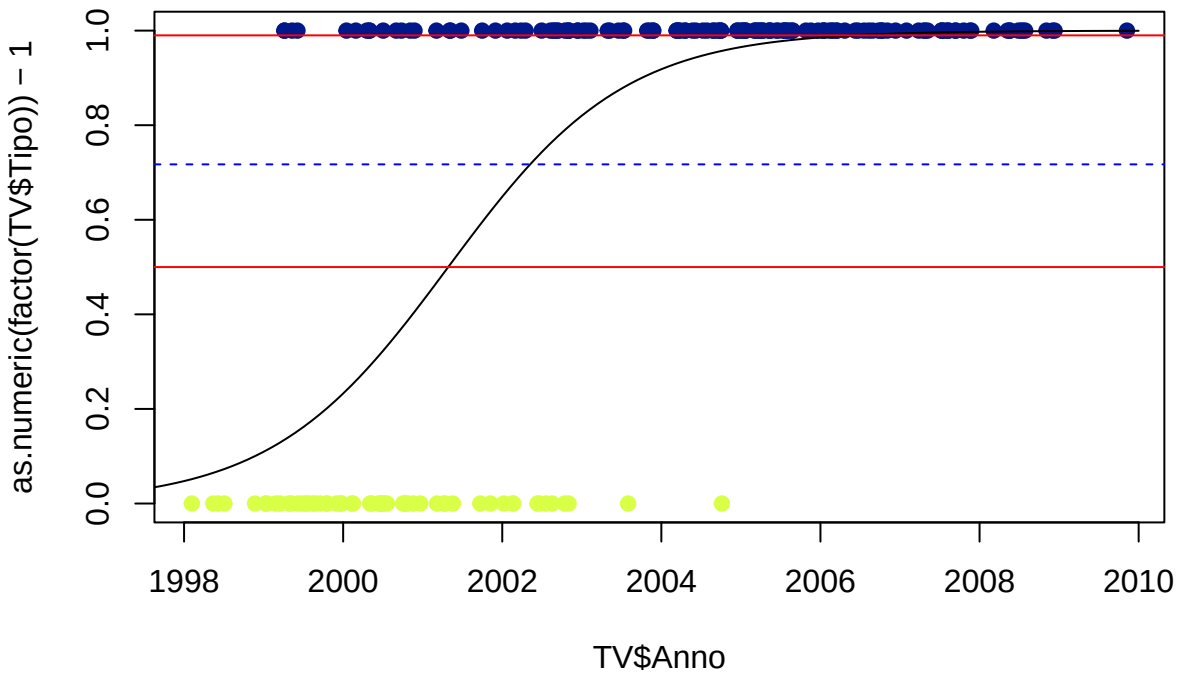
```
# c) provide a pointwise estimate for the year in which sales of 16:9
#    televisions exceeded those of 4:3;
(log(0.5/0.5) - coefficients(fit)[1]) / coefficients(fit)[2]
```

```
## (Intercept)
##    2001.321
```

```
abline(h=0.5, col='red')

# d) provide a pointwise estimate for the year in which 16:9 televisions
#    will cover (or have covered) 99% of the Italian market.
(log(0.99/0.01) - coefficients(fit)[1]) / coefficients(fit)[2]
```

```
## (Intercept)
##    2006.407
```

```
abline(h=0.99, col='red')
```



10.5 CARTs

10.5.1 Carseats dataset

```
#-----------------------------------------------------------------------------
##### Classification and regression trees
#####--------------------------

### Classification Trees: Carseats dataset

# help(Carseats)
dim(Carseats)
```

```
## [1] 400  11
```

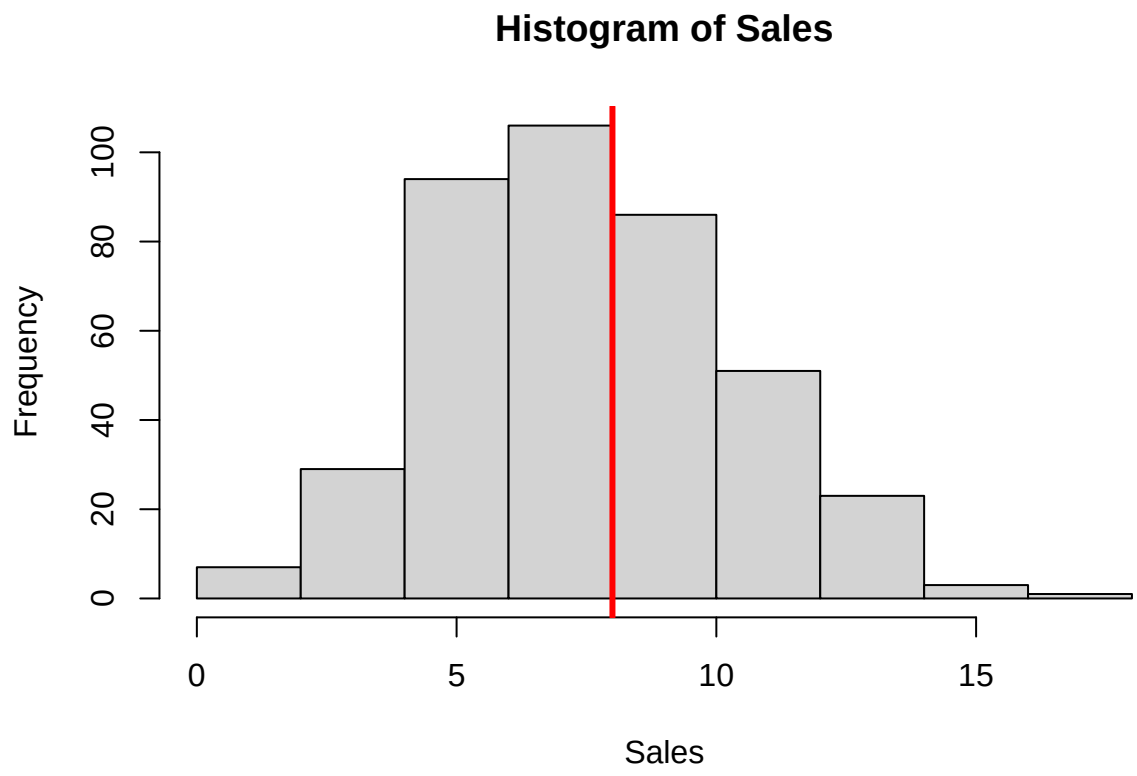
```
head(Carseats)
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 1  9.50       138     73          11        276   120       Bad  42        17
## 2 11.22       111     48          16        260    83      Good  65        10
## 3 10.06       113     35          10        269    80    Medium  59        12
## 4  7.40       117    100           4        466    97    Medium  55        14
## 5  4.15       141     64           3        340   128       Bad  38        13
## 6 10.81       124    113          13        501    72       Bad  78        16
##   Urban  US
## 1   Yes Yes
## 2   Yes Yes
## 3   Yes Yes
## 4   Yes Yes
## 5   Yes  No
## 6    No Yes
```

```
names(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

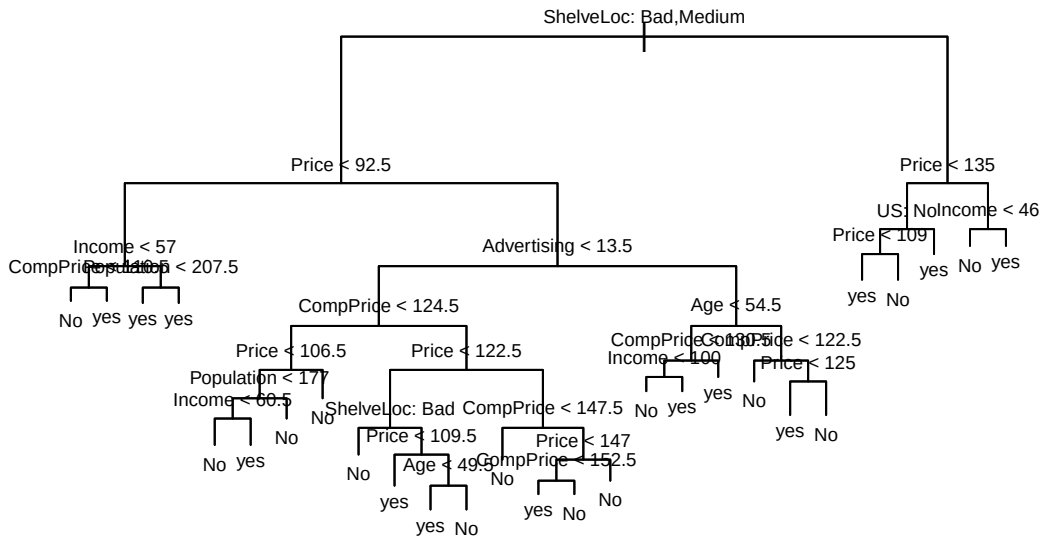
```r
attach(Carseats)
hist(Sales)
abline(v=8,lwd=3,col='red')
```




```r
High = ifelse(Sales<=8,"No","yes")
detach(Carseats)

Carseats = data.frame(Carseats,High)

library(tree)
# help(tree)

tree.carseats = tree(factor(High)~.-Sales,Carseats)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = factor(High) ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

```r
plot(tree.carseats)
text(tree.carseats,pretty=0,cex=0.6)
```




```
tree.carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930 yes ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481 yes ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050 yes ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300 yes ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742 yes ( 0.14286 0.85714 ) *
##                 349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830 yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 yes ( 0.20000 0.80000 )
```

```
##            44) CompPrice < 130.5 14  18.250 yes ( 0.35714 0.64286 )
##              88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##              89) Income > 100 5   0.000 yes ( 0.00000 1.00000 ) *
##            45) CompPrice > 130.5 11   0.000 yes ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##            46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##            47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##              94) Price < 125 5   0.000 yes ( 0.00000 1.00000 ) *
##              95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##     3) ShelveLoc: Good 85  90.330 yes ( 0.22353 0.77647 )
##       6) Price < 135 68  49.260 yes ( 0.11765 0.88235 )
##        12) US: No 17  22.070 yes ( 0.35294 0.64706 )
##          24) Price < 109 8   0.000 yes ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##        13) US: Yes 51  16.880 yes ( 0.03922 0.96078 ) *
##       7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##        14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##        15) Income > 46 11  15.160 yes ( 0.45455 0.54545 ) *
```

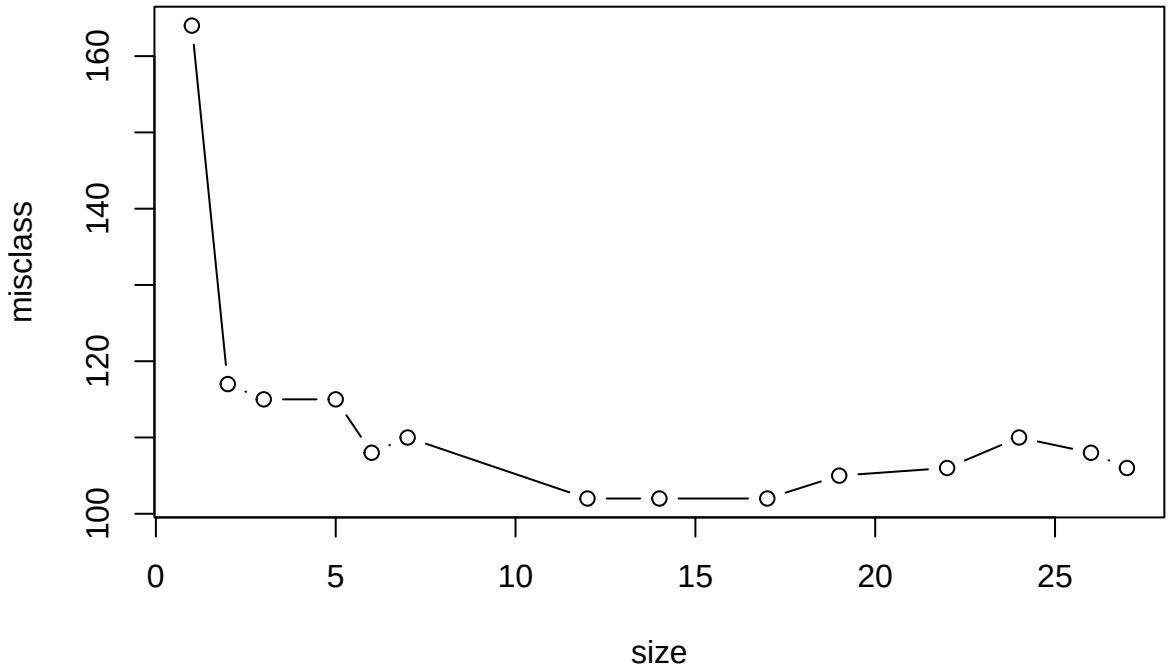
```r
# use cross validation to prune the tree optimally
# help(cv.tree)

set.seed(1)
cv.carseats = cv.tree(tree.carseats,FUN=prune.misclass)

names(cv.carseats)
```

```
## [1] "size"   "dev"    "k"      "method"
```

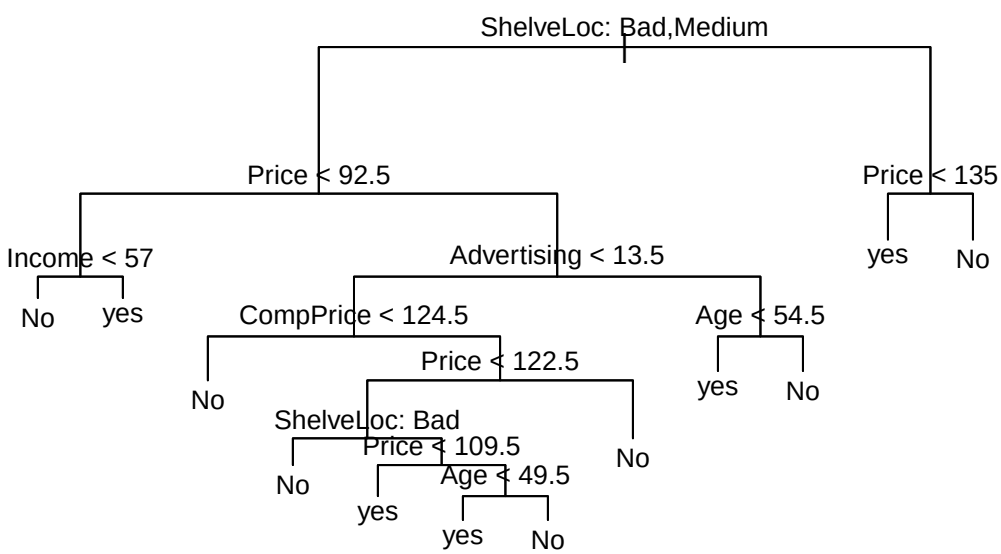
```r
cv.carseats
```

```
## $size
##  [1] 27 26 24 22 19 17 14 12  7  6  5  3  2  1
##
## $dev
##  [1] 106 108 110 106 105 102 102 102 110 108 115 115 117 164
##
## $k
##  [1]      -Inf  0.000000  0.500000  1.000000  1.333333  1.500000  1.666667
##  [8]  2.500000  3.800000  4.000000  5.000000  7.500000 18.000000 47.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(cv.carseats$size,cv.carseats$dev,type="b",xlab="size",ylab="misclass")
```




```
# help(prune.misclass)
prune.carseats = prune.misclass(tree.carseats,best=12)

plot(prune.carseats)
text(prune.carseats,pretty=0, cex=0.8)
```



10.5.2 Boston housing dataset

```
### Regression Trees: Boston housing dataset

# help(Boston)
dim(Boston)
```

```
## [1] 506  14
```

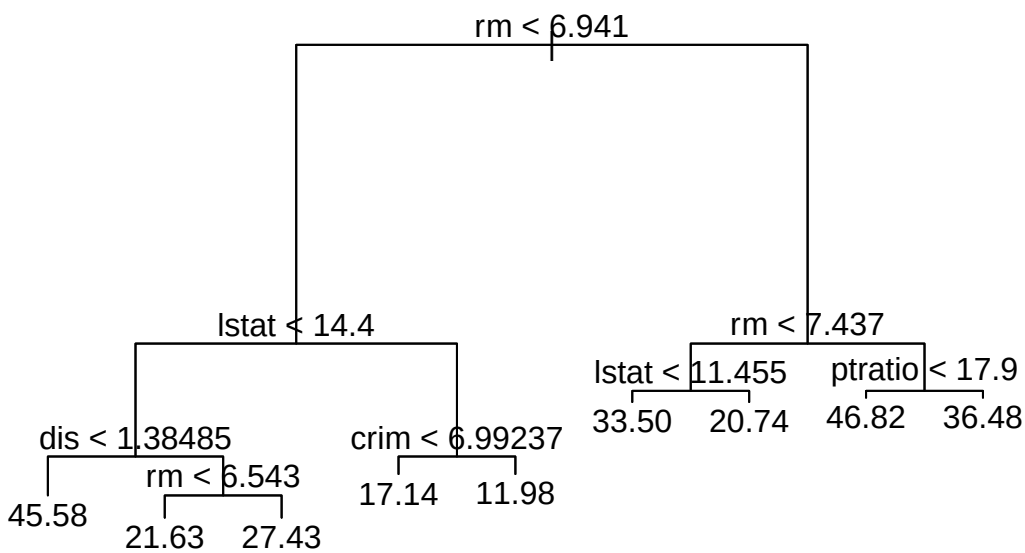
```
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

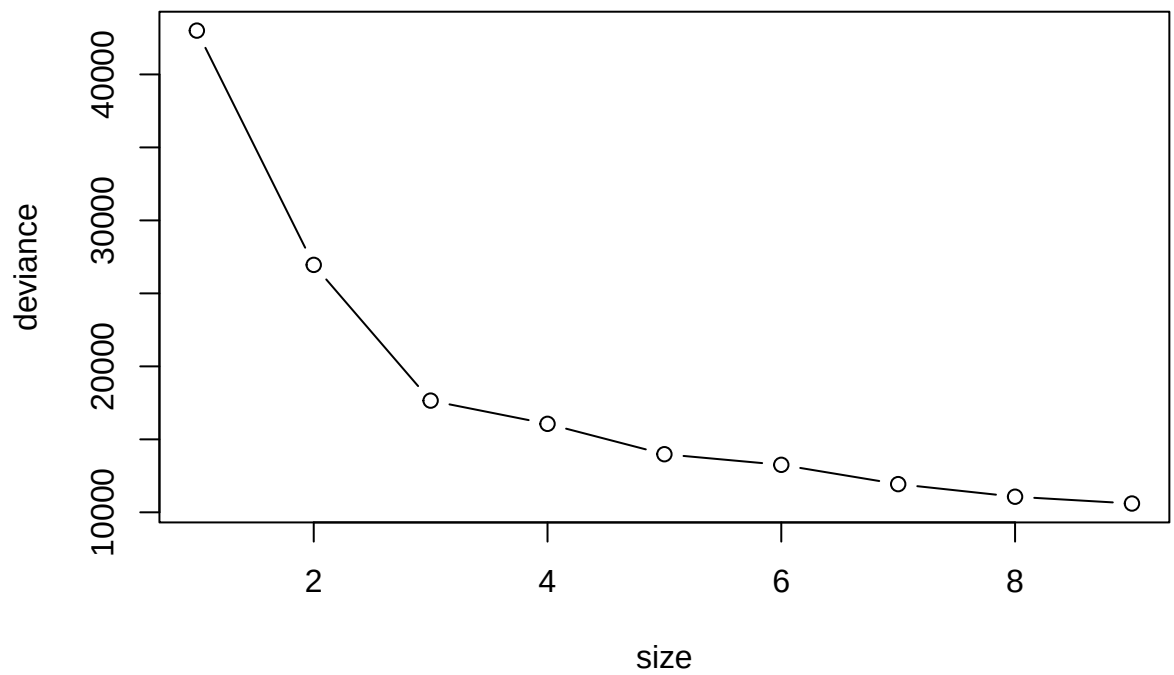
```
tree.boston = tree(medv~.,Boston)
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston)
## Variables actually used in tree construction:
## [1] "rm"      "lstat"   "dis"     "crim"    "ptratio"
## Number of terminal nodes:  9
## Residual mean deviance:  13.55 = 6734 / 497
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -17.68000  -2.23000   0.07026   0.00000   2.22100  16.50000
```

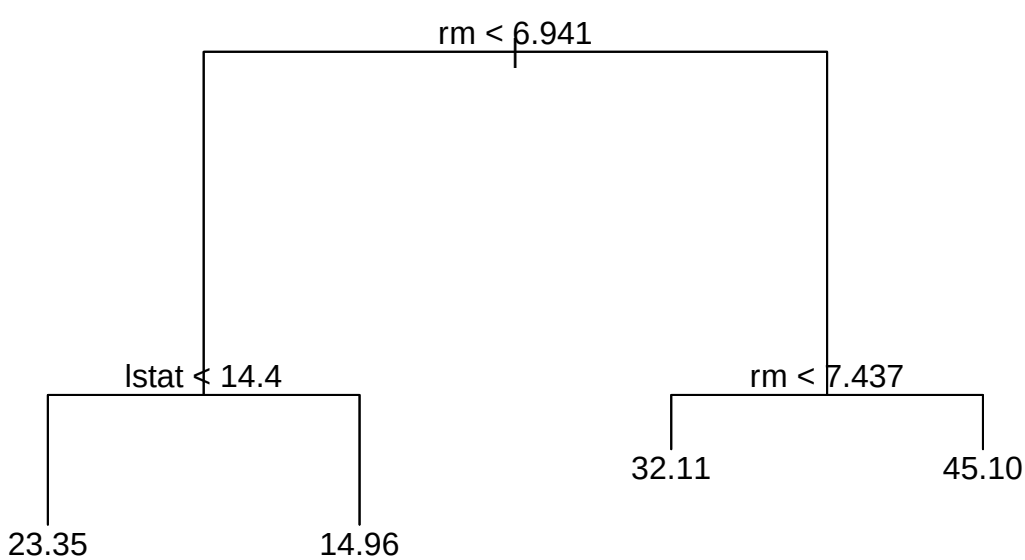
```
plot(tree.boston)
text(tree.boston,pretty=0)
```




```
cv.boston = cv.tree(tree.boston)

plot(cv.boston$size,cv.boston$dev,type='b',xlab='size',ylab='deviance')
```




```
prune.boston = prune.tree(tree.boston,best=4)

plot(prune.boston)
text(prune.boston,pretty=0)
```



10.5.3 Exam exercise

```r
#### Pb3 of 09/02/2022  ####
# The file wine.txt reports the data on the alcohol content in 179 bottles of wine.
# For the alcohol content consider a linear model, accounting for the sugar content
# of grapes, and for type of wine ('Red', 'Rose', 'White'):
#   alcoholg = b0,g + b1,g*sugar + eps,
# with eps ~N(0, sigma^2) and g the grouping structure induced by the type of wine.
# a) Estimate the parameters of the model ({b0,g, b1,g, sigma}). Verify the model assumptions,
#    reporting any plot you consider important.
# b) Perform two statistical tests - each at level 1% - to verify if
#     - there is a significant dependence of the mean alcohol content on the type of wine;
#     - there is a significant dependence of the mean alcohol content on the sugar content.
# c) Based on tests (b) or any other test deemed relevant, reduce the model and report
#    the updated model parameters.
# d) Build a prediction interval at 99% for a new bottle of red wine made with grapes
#    with 20 g of sugar.

data = read.table('Lab10/wine.txt')
head(data)
```

```
##   alcohol sugar type
## 1    16.4 23.35  Red
## 2    11.6 18.42  Red
## 3    12.5 17.82  Red
## 4    11.2 17.76  Red
## 5    16.6 23.55  Red
## 6    10.4 17.44  Red
```

```r
data$type = as.factor(data$type)

# a)
D1 = ifelse(data$type == 'Red', 1, 0)
D2 = ifelse(data$type == 'White', 1, 0)
fit = lm(alcohol ~ D1 + D2 + sugar + D1:sugar + D2:sugar, data = data)
summary(fit)
```

```
##
## Call:
## lm(formula = alcohol ~ D1 + D2 + sugar + D1:sugar + D2:sugar,
##     data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.61294 -0.44519  0.01684  0.48766  1.89144
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.71191    0.63180  -2.710  0.00741 **
## D1           0.45853    0.84885   0.540  0.58977
## D2           1.01148    0.92170   1.097  0.27399
## sugar        0.66214    0.03121  21.214  < 2e-16 ***
## D1:sugar     0.06514    0.04232   1.539  0.12560
## D2:sugar    -0.06589    0.04508  -1.462  0.14560
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6726 on 173 degrees of freedom
## Multiple R-squared:  0.9048, Adjusted R-squared:  0.9021
## F-statistic: 328.9 on 5 and 173 DF,  p-value: < 2.2e-16
```

```r
fit2 = lm(alcohol ~ type + sugar + type:sugar, data = data)
summary(fit2)
```

```
##
## Call:
## lm(formula = alcohol ~ type + sugar + type:sugar, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.61294 -0.44519  0.01684  0.48766  1.89144
##
```

```
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -1.25338    0.56690  -2.211  0.02835 *
## typeRose        -0.45853    0.84885  -0.540  0.58977
## typeWhite        0.55295    0.87848   0.629  0.52989
## sugar            0.72728    0.02858  25.446  < 2e-16 ***
## typeRose:sugar  -0.06514    0.04232  -1.539  0.12560
## typeWhite:sugar -0.13103    0.04330  -3.026  0.00285 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6726 on 173 degrees of freedom
## Multiple R-squared:  0.9048, Adjusted R-squared:  0.9021
## F-statistic: 328.9 on 5 and 173 DF,  p-value: < 2.2e-16
```

```
beta0 = c(coef(fit)[1],coef(fit)[1]+coef(fit)[2], coef(fit)[1]+coef(fit)[3])
beta1 = c(coef(fit)[4],coef(fit)[4]+coef(fit)[5], coef(fit)[4]+coef(fit)[6])
beta0
```

```
## (Intercept) (Intercept) (Intercept)
##  -1.7119121  -1.2533849  -0.7004341
```

```
beta1
```

```
##     sugar     sugar     sugar
## 0.6621416 0.7272803 0.5962484
```

```
shapiro.test(residuals(fit))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fit)
## W = 0.99455, p-value = 0.7562
```

```
# b)
library(car)
linearHypothesis(fit,
                 rbind(c(0,1,0,0,0,0),
                       c(0,0,1,0,0,0),
                       c(0,0,0,0,1,0),
                       c(0,0,0,0,0,1)),
                 c(0,0,0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## D1 = 0
## D2 = 0
## D1:sugar = 0
## D2:sugar = 0
##
## Model 1: restricted model
## Model 2: alcohol ~ D1 + D2 + sugar + D1:sugar + D2:sugar
##
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1    177 239.410
## 2    173  78.273  4    161.14 89.037 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
linearHypothesis(fit,
                 rbind(c(0,0,0,1,0,0),
                       c(0,0,0,0,1,0),
                       c(0,0,0,0,0,1)),
                 c(0,0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## sugar = 0
```

```
## D1:sugar = 0
## D2:sugar = 0
##
## Model 1: restricted model
## Model 2: alcohol ~ D1 + D2 + sugar + D1:sugar + D2:sugar
##
##   Res.Df    RSS Df Sum of Sq     F    Pr(>F)
## 1    176 726.94
## 2    173  78.27  3    648.67 477.9 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# c)
linearHypothesis(fit,
                 rbind(c(0,1,0,0,0,0),
                       c(0,0,1,0,0,0)),
                 c(0,0))
```

```
## Linear hypothesis test
##
## Hypothesis:
## D1 = 0
## D2 = 0
##
## Model 1: restricted model
## Model 2: alcohol ~ D1 + D2 + sugar + D1:sugar + D2:sugar
##
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    175 78.818
## 2    173 78.273  2   0.54512 0.6024 0.5486
```

```
fit2 = lm(alcohol~ sugar + D1:sugar + D2:sugar, data = data)
summary(fit2)
```

```
##
## Call:
## lm(formula = alcohol ~ sugar + D1:sugar + D2:sugar, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.68380 -0.44242  0.01759  0.48212  1.88956
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.243293   0.356392  -3.489 0.000614 ***
## sugar        0.639249   0.018018  35.477  < 2e-16 ***
## sugar:D1     0.087528   0.006207  14.101  < 2e-16 ***
## sugar:D2    -0.016915   0.006266  -2.699 0.007626 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6711 on 175 degrees of freedom
## Multiple R-squared:  0.9041, Adjusted R-squared:  0.9025
## F-statistic: 550.2 on 3 and 175 DF,  p-value: < 2.2e-16
```

```
c(coef(fit2)[1])
```

```
## (Intercept)
##   -1.243293
```

```
c(coef(fit2)[2],coef(fit2)[2]+coef(fit2)[3],coef(fit2)[2]+coef(fit2)[4])
```

```
##     sugar     sugar     sugar
## 0.6392491 0.7267769 0.6223343
```

```
# d)
predict(fit2, data.frame(sugar = 20, D1 = 1, D2=0),
        interval = 'prediction', level = 0.99)
```

```
##        fit      lwr      upr
## 1 13.29224 11.53151 15.05298
```

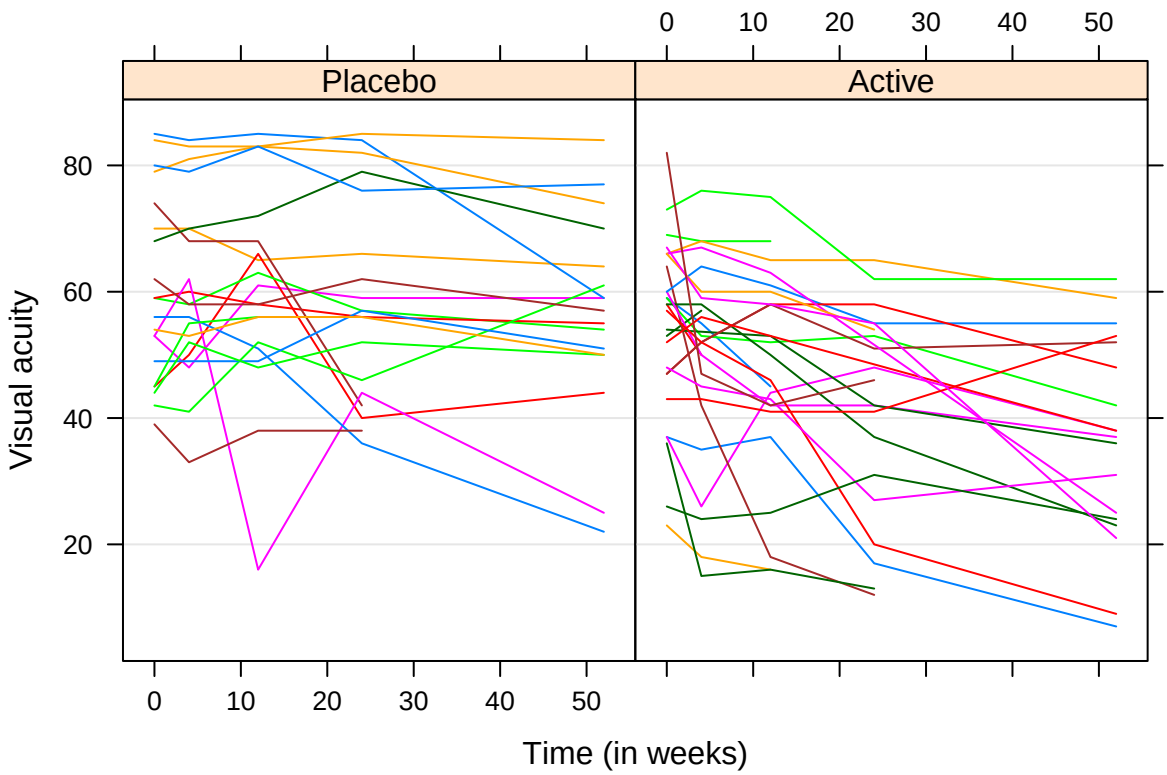
11 Lab 11 - LMMs

```
# General topics
# Section From LM to LMM
# Topics:
#   1. Linear Models with homoscedastic and independent errors
#   2. Linear Models with heteroscedastic and independent errors
#      2.1 VarIdent()
#      2.2 VarPower()
#   3. Linear Models with heteroscedastic and dependent errors
#!     for each individual in the time spans we impose:
#      3.1 CorCompSym() equal correlation
#      3.2 AR(1) decreasing correlation
#      3.3 general no contraints on the correlation
#
#
# Section LMM 1
# Topics:
#   LINEAR MIXED MODELS WITH HOMOSCEDASTIC RESIDUALS
#   1. Linear Models with random intercept (q=0)
#   2. Linear Models with random intercept + slope (q=1)
#      2.1 general structure of D
#      2.2 diagonal D
#   3. Interpretation of random effects and PVRE
#   4. Prediction
#   5. Diagnostic
#   6. Models comparison
#
#   SUPPLEMENTARY MATERIAL: LINEAR MIXED MODELS WITH HETEROSCEDASTIC RESIDUALS (VarPower())
#   1. Linear Models with random intercept (q=0)
#   2. Linear Models with random intercept + slope (q=1)
#      2.1 general structure of D
#      2.2 diagonal D
#
#
# Section LMM 2
# Topics:
#   - Linear Mixed Models with Random Intercept
#   - Linear Mixed Models with Random Intercept + Random Slope
#   - Inference
#   - Assessing Assumptions
#   - Prediction
```

11.1 From LM to LMM

```
# Topics:
#   1. Linear Models with homoscedastic and independent errors
#   2. Linear Models with heteroscedastic and independent errors
#      2.1 VarIdent()
#      2.2 VarPower()
#   3. Linear Models with heteroscedastic and dependent errors
#!  for each individual in the time spans we impose:
#      3.1 CorCompSym() equal correlation
#      3.2 AR(1) decreasing correlation
#      3.3 general no contraints on the correlation

library(nlmeU) ## --> for the dataset
library(nlme)  ## --> for models implementation

library(corrplot)
library(lattice)
library(plot.matrix)

data(armd) # Age-Related Macular Degeneration: dataset of interest
data(armd0) # Age-Related Macular Degeneration: dataset for visualization
# help(armd)

# The ARMD data arise from a randomized multi-center clinical trial comparing an
# experimental treatment (interferon-alpha)
# versus placebo for patients diagnosed with ARMD. Patients with ARMD progressively
# lose vision.
# The dataset contains information about 234 subjects, for which the visual level is
```

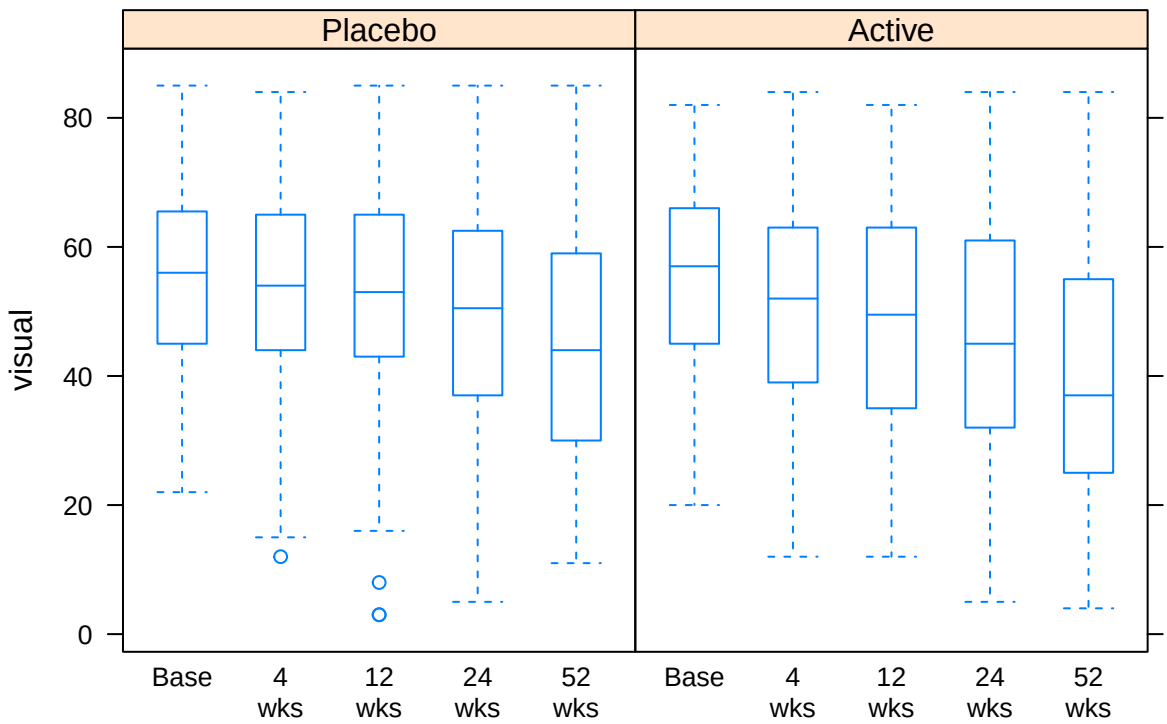
```r
# measured up to 4 times.
# We are mainly interested in the effect of treatment on the visual acuity measurements.

## The ARMD0 dataset contains the same information, but with an extra row for each
## patient relative
## to the measurement at time 0. ARMD0 contains 240 subjects.

## Visual-acuity profiles for selected patients --> we visualize same of the trends
armd0.subset = subset(armd0, as.numeric(subject) %in% seq(1, 240, 5)) # one each 5 patients
#! The dataset of the Ieva's slides example

xy1 = xyplot(visual ~ time | treat.f,
                 groups = subject,
                 data = armd0.subset,
                 type = "l", lty = 1)
update(xy1, xlab = "Time (in weeks)",
         ylab = "Visual acuity",
         grid = "h")
```




```r
## We observe a decreasing trend in time, on average, but patients have very
## different trends.
## Also, Active patients have on average lower values of the response.

#! Each line represent a subject observed at each time-registration instants
#! The trend of visual acuity measurment, our target, is not always the same
#! And eterogeneity: the lines spread a lot over time, so variability is not
#! const over time

## sample means across time and treatment
flst = list(armd$time.f, armd$treat.f)
tMn = tapply(armd$visual, flst, FUN = mean)
tMn
```

```
##        Placebo   Active
## 4wks  53.96581 50.91228
## 12wks 52.87179 48.67273
## 24wks 49.33036 45.46078
## 52wks 44.43810 39.10000
```

```r
## We confirm what we observe in the plot

## Box-plots for visual acuity by treatment and time
bw1 = bwplot(visual ~ time.f | treat.f,
        data = armd0)
xlims = c("Base", "4\nwks", "12\nwks", "24\nwks", "52\nwks")
update(bw1, xlim = xlims, pch = "|")
```




```r
#! We sew both cases a decreasing trend, and enlarging variability with time (a bit)
```

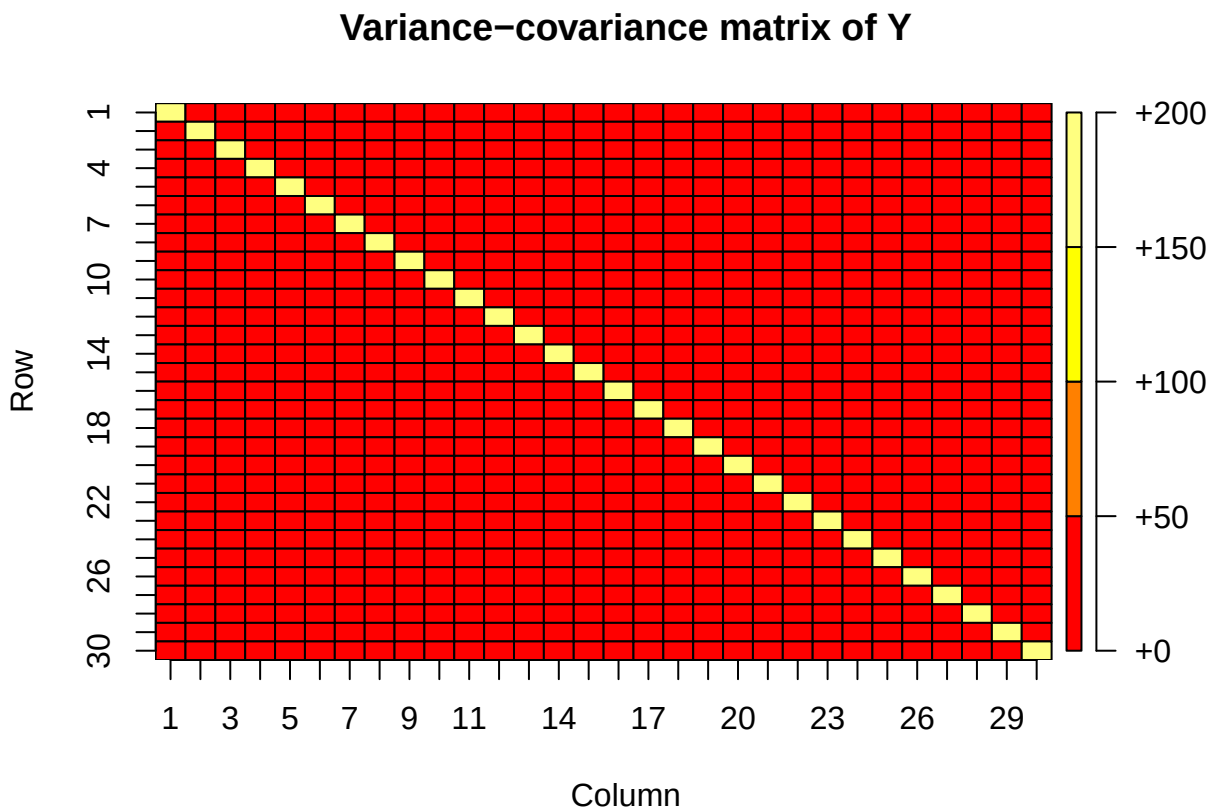
11.1.1 1. Linear Models with homoscedastic and independent errors

```r
################################################################################
# 1. Linear Models with homogeneous and independent errors: we start by considering
#    all observations as independent, with homogeneous variance

# LM: VISUAL_it = b_0*t + b1*VISUAL_0i + b_2*t*TREAT_i + e_it
#! b_0*t so that at each time we get a different intercept

# b_0t, b_1, and b_2t denote the timepoint-specific intercept,
# baseline visual acuity effect, and timepoint-specific treatment effect.
# Thus, the model assumes a time-dependent treatment effect,
# with the time variable being treated as a factor.

# To obtain timepoint-specific intercepts at 4,12,24 and 52 weeks,
# the overall intercept is removed from the model by specifying the -1 term.

lm1.form = lm(visual ~ -1 + visual0 + time.f + treat.f:time.f, data = armd )
summary(lm1.form)
```

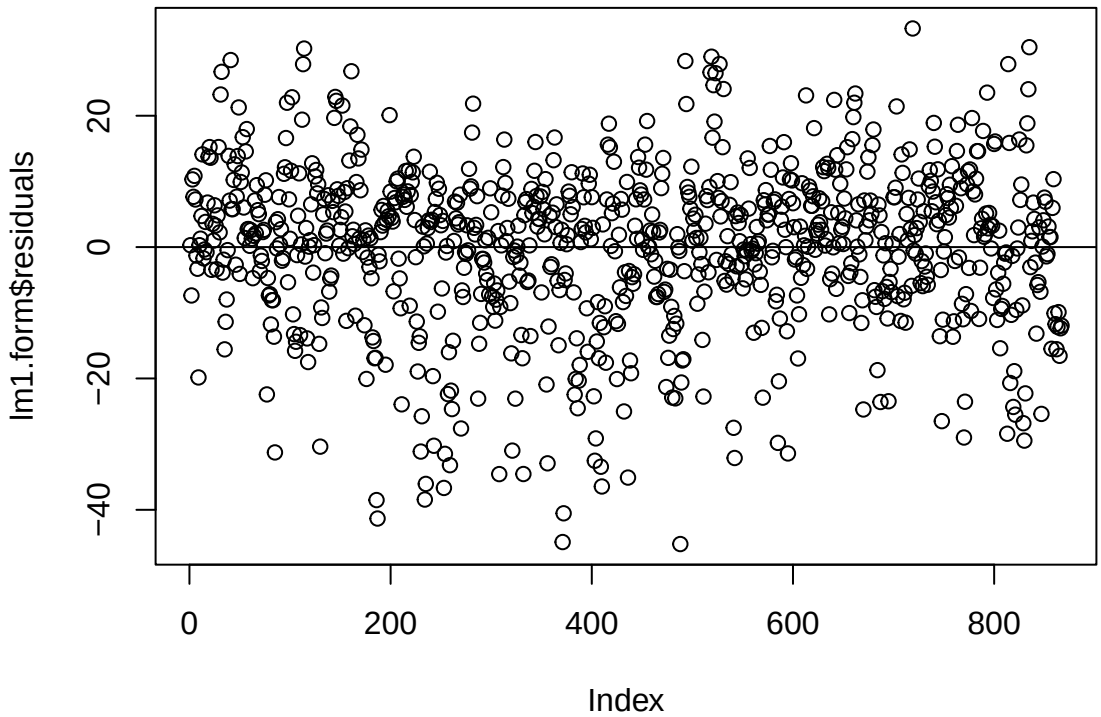
```
##
## Call:
## lm(formula = visual ~ -1 + visual0 + time.f + treat.f:time.f,
##     data = armd)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -45.210  -6.459   1.532   7.512  33.283
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## visual0                    0.83037    0.02842  29.213  < 2e-16 ***
## time.f4wks                 8.07531    1.94341   4.155 3.58e-05 ***
## time.f12wks                7.08066    1.94066   3.649  0.00028 ***
## time.f24wks                3.63022    1.95316   1.859  0.06342 .
## time.f52wks               -1.74643    1.98952  -0.878  0.38029
## time.f4wks:treat.fActive  -2.35278    1.62894  -1.444  0.14900
## time.f12wks:treat.fActive -3.70852    1.64378  -2.256  0.02432 *
## time.f24wks:treat.fActive -3.44915    1.69399  -2.036  0.04205 *
## time.f52wks:treat.fActive -4.47345    1.77811  -2.516  0.01206 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.38 on 858 degrees of freedom
## Multiple R-squared:  0.9432, Adjusted R-squared:  0.9426
## F-statistic:  1583 on 9 and 858 DF,  p-value: < 2.2e-16
```

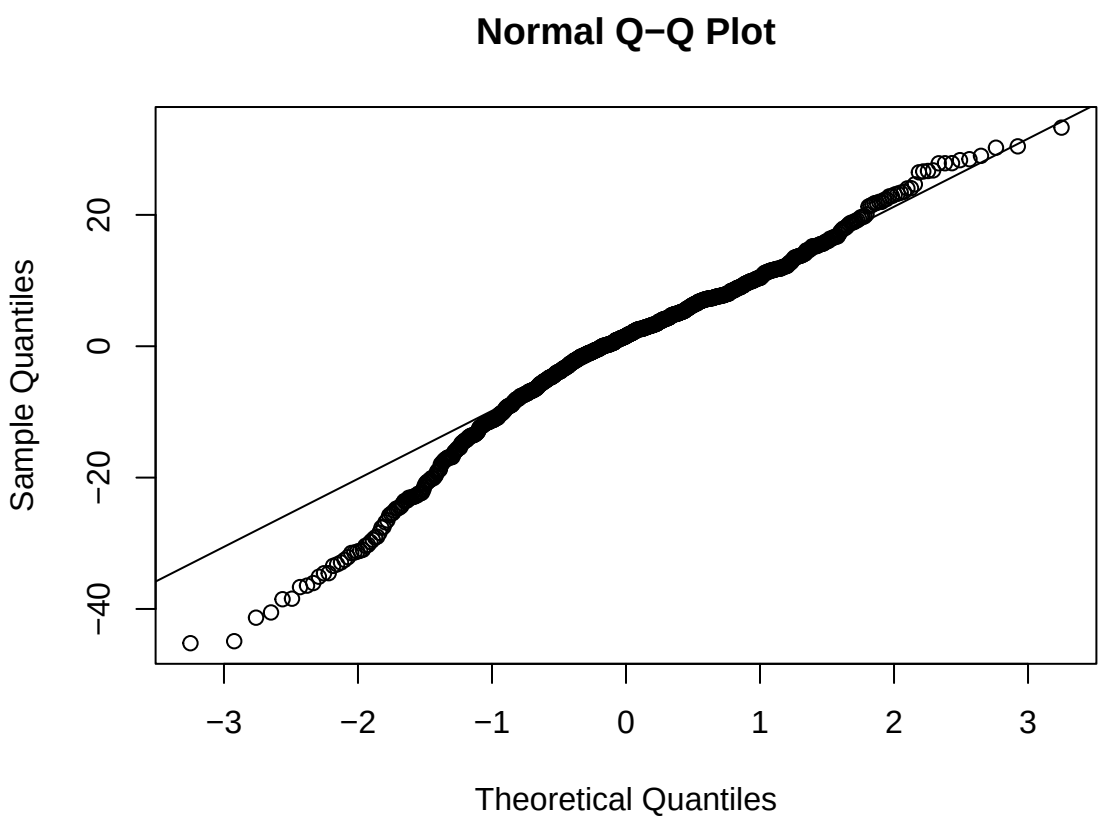
```r
#! time.f<K>wks have decreasing coeffs (8,7,3,-1) as the effect will decrease with time
#! R^2 adj is good, but we now have to check the assumptions


# variance-covariance matrix of Y  --> it is a diagonal matrix with a value of 12.38^2
par(mar = c(4,4,4,4))
#! 12.38 get from the summary, it was residual std error
plot(diag(x=12.38^2,nrow=30, ncol=30), main='Variance-covariance matrix of Y')
```




```r
## residual analysis
plot(lm1.form$residuals)
#! they seem quite homoscedastic, the trend is sparse around zero
abline(h=0)
```



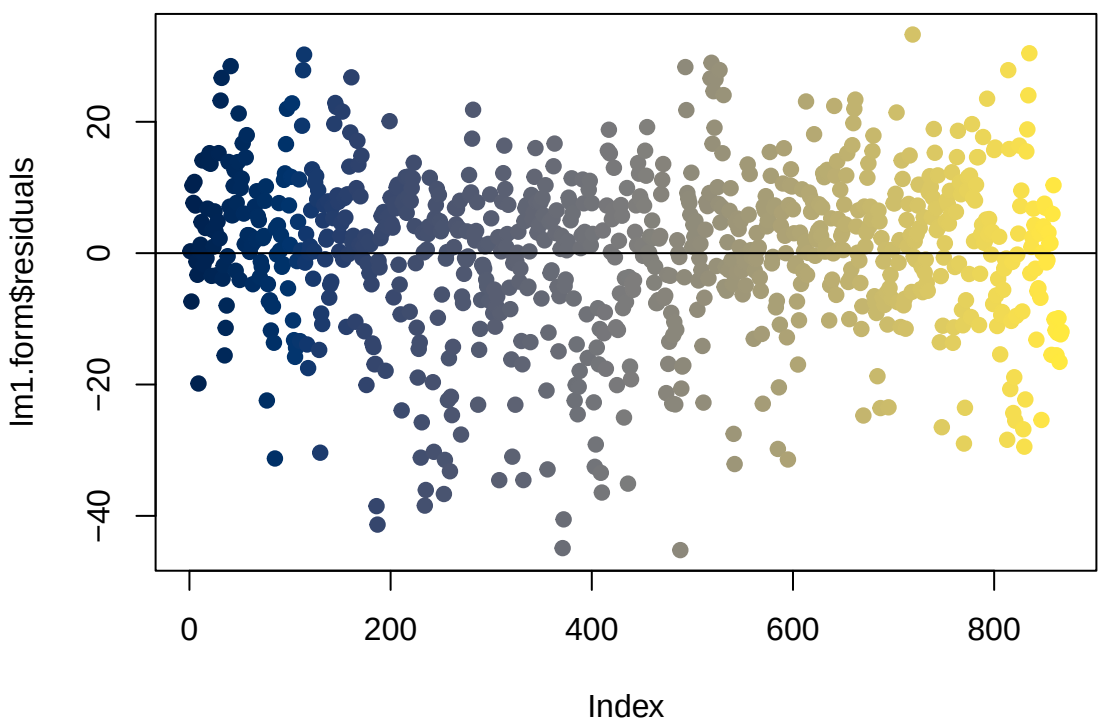

```
qqnorm(lm1.form$residuals)
qqline(lm1.form$residuals)
```



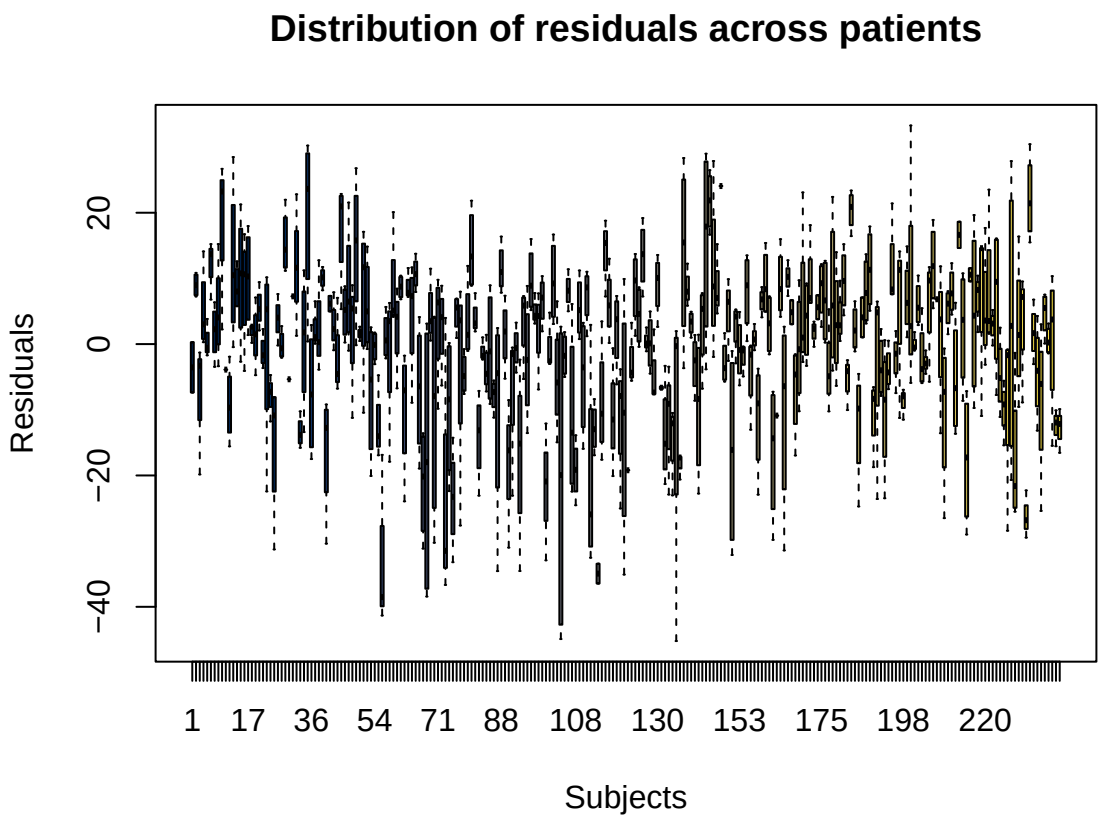
```r
shapiro.test(lm1.form$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  lm1.form$residuals
## W = 0.96922, p-value = 1.453e-12
```

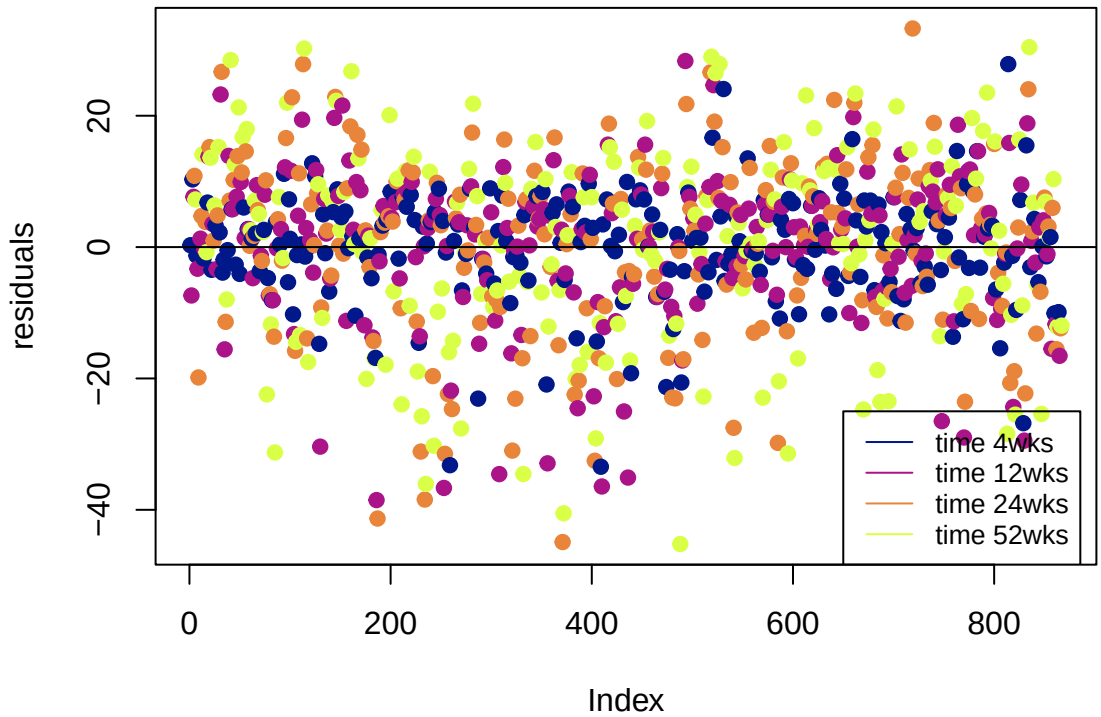
```r
#! we reject the hp of gaussianity (for errors)


## But we know that observations are not independent and that the variance of the
## visual measurements increases in time

## let's color the residuals relative to different patients
# colori =rainbow(length(unique(armd$subject)))
colori = hcl.colors(length(unique(armd$subject)), palette="cividis")

num_sub = table(armd$subject)
colori2 = rep(colori, num_sub)
plot(lm1.form$residuals, col=colori2, pch=19)
abline(h=0)   ## --> not very informative
```



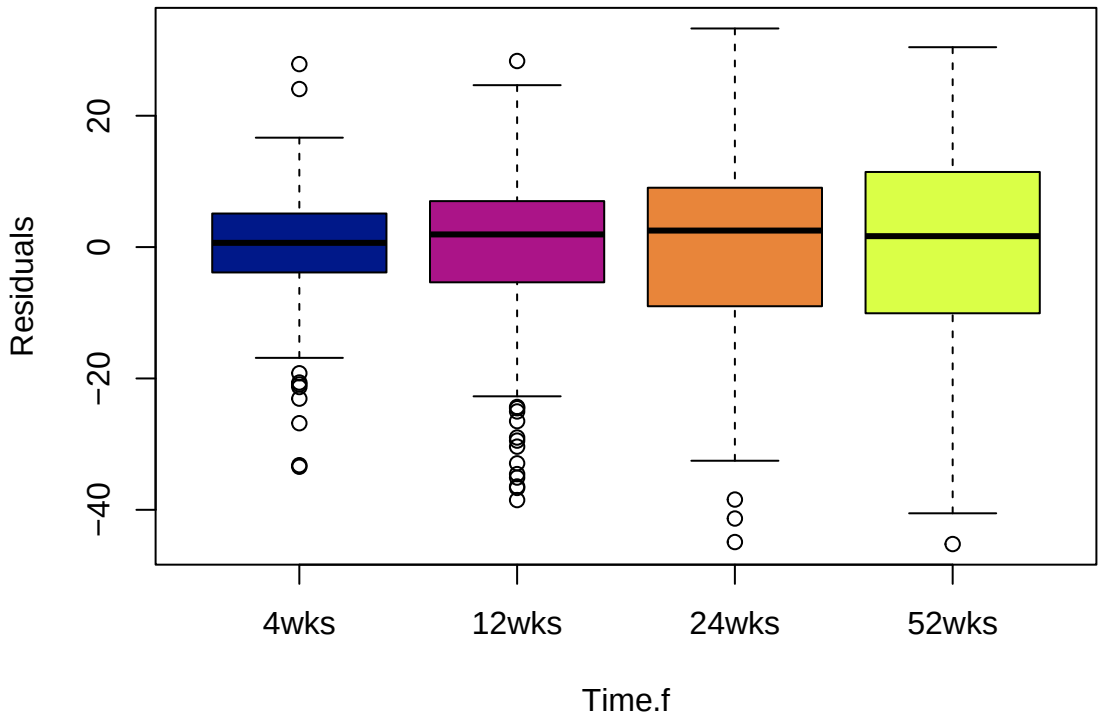

```r
boxplot(lm1.form$residuals ~ armd$subject, col=colori,
        xlab='Subjects', ylab='Residuals',
        main ='Distribution of residuals across patients')
```




```
## --> informative!

## let's color the residuals relative to different time instants
set.seed(1)
colori = colori_fun(4)
colori2 = colori[armd$tp] # associate to each one of the 4 time instants a color
plot(lm1.form$residuals, col=colori2, ylab='residuals', pch=19)
abline(h=0)
legend(650, -25, legend=c("time 4wks", "time 12wks", "time 24wks", "time 52wks"),
       col=colori, lty=1, cex=0.8)
```




```
## Note: we observe that red points are the closest to 0, purple ones are the farthest
## We expect the residuals to be heterogeneous across different time instants observations
#! So we see trends wrt subjects and time instants

boxplot(lm1.form$residuals ~ armd$time.f, col=colori,
        xlab='Time.f', ylab='Residuals')
```




```
## -> the variance of th observations increases in time



# The model does not take into account the correlation
# between the visual acuity observations obtained from the same subject.
# It also does not take into account the heterogeneous variability
# present at different time points. Thus, it should not be used as a basis for inference.
```

11.1.2 2. Linear Models with heteroscedastic and independent errors

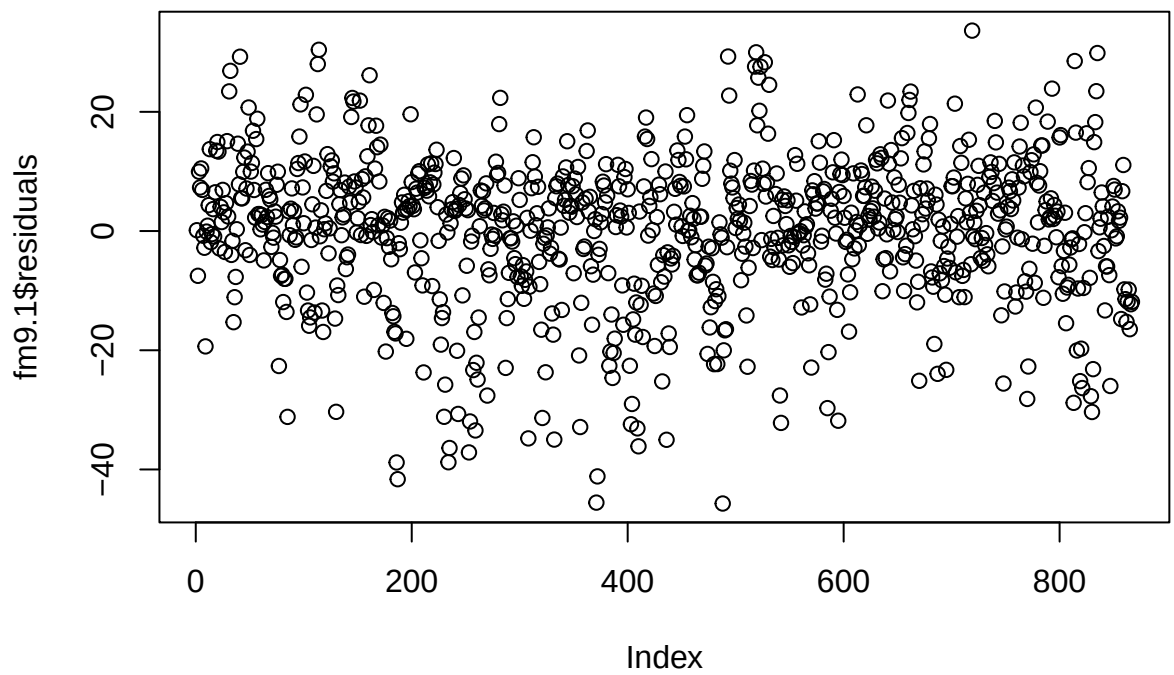
```
################################################################################
## 2. Linear models with heteroscedastic and independent errors

## We know that variance increases in time --> we model the variance as a function
## of time
## We have different possibilities

## gls() function allows the inclusion of dependency and heteroscedasticity
```

```
## 2.1 Option 1: VarIdent()
fm9.1 = gls(visual ~ -1 + visual0 + time.f + treat.f:time.f,  # the same as before
            weights = varIdent(form = ~1|time.f), # Var. function; <delta, stratum>-group
            data = armd)

#! varIdent case gives a structure like
#!            / 1                    \
#!  sigma^2 * |   delta1             |
#!            |         delta2       |
#!            \               delta3 /
#! ie we let the different stratum (here the four time istants) to take different values
#! those blocks may also have different dimension, but square
summary(fm9.1)
```

11.1.2.1 2.1 VarIdent()

```
## Generalized least squares fit by REML
##   Model: visual ~ -1 + visual0 + time.f + treat.f:time.f
##   Data: armd
##        AIC      BIC    logLik
##   6740.294 6802.104 -3357.147
##
## Variance function:
##  Structure: Different standard deviations per stratum
##  Formula: ~1 | time.f
##  Parameter estimates:
##     4wks    12wks    24wks    52wks
## 1.000000 1.397600 1.664321 1.880852
##
## Coefficients:
##                              Value Std.Error  t-value p-value
## visual0                   0.862987  0.025444 33.91714  0.0000
## time.f4wks                6.272900  1.599432  3.92195  0.0001
## time.f12wks               5.282146  1.761645  2.99842  0.0028
## time.f24wks               1.835279  1.908354  0.96171  0.3365
## time.f52wks              -3.560392  2.071846 -1.71846  0.0861
## time.f4wks:treat.fActive -2.325253  1.085147 -2.14280  0.0324
## time.f12wks:treat.fActive -3.689255  1.530278 -2.41084  0.0161
## time.f24wks:treat.fActive -3.432641  1.877967 -1.82785  0.0679
## time.f52wks:treat.fActive -4.439493  2.227562 -1.99298  0.0466
##
##  Correlation:
##                          visul0 tm.f4w tm.f12 tm.f24 tm.f52 t.4:.A t.12:.
## time.f4wks               -0.879
## time.f12wks              -0.796  0.700
## time.f24wks              -0.734  0.645  0.584
## time.f52wks              -0.683  0.601  0.544  0.501
## time.f4wks:treat.fActive  0.020 -0.352 -0.016 -0.015 -0.014
## time.f12wks:treat.fActive  0.010 -0.009 -0.429 -0.007 -0.007  0.000
## time.f24wks:treat.fActive  0.007 -0.006 -0.005 -0.474 -0.005  0.000  0.000
## time.f52wks:treat.fActive  0.012 -0.010 -0.009 -0.009 -0.504  0.000  0.000
##                          t.24:.
## time.f4wks
## time.f12wks
## time.f24wks
## time.f52wks
## time.f4wks:treat.fActive
## time.f12wks:treat.fActive
## time.f24wks:treat.fActive
## time.f52wks:treat.fActive  0.000
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -4.0577911 -0.5342854  0.1387429  0.6498811  3.4597584
##
## Residual standard error: 8.244094
## Degrees of freedom: 867 total; 858 residual
```

```r
#! Parameter estimates:
#!   4wks    12wks    24wks    52wks
#!   1.000000 1.397600 1.664321 1.880852
#! correctly values are increasing as we expect, from the boxplot for example
#! And now we can do more precisely inference, since the residuals have different
#! variances according to the time-interval units to which they refer

plot(fm9.1$residuals)
```



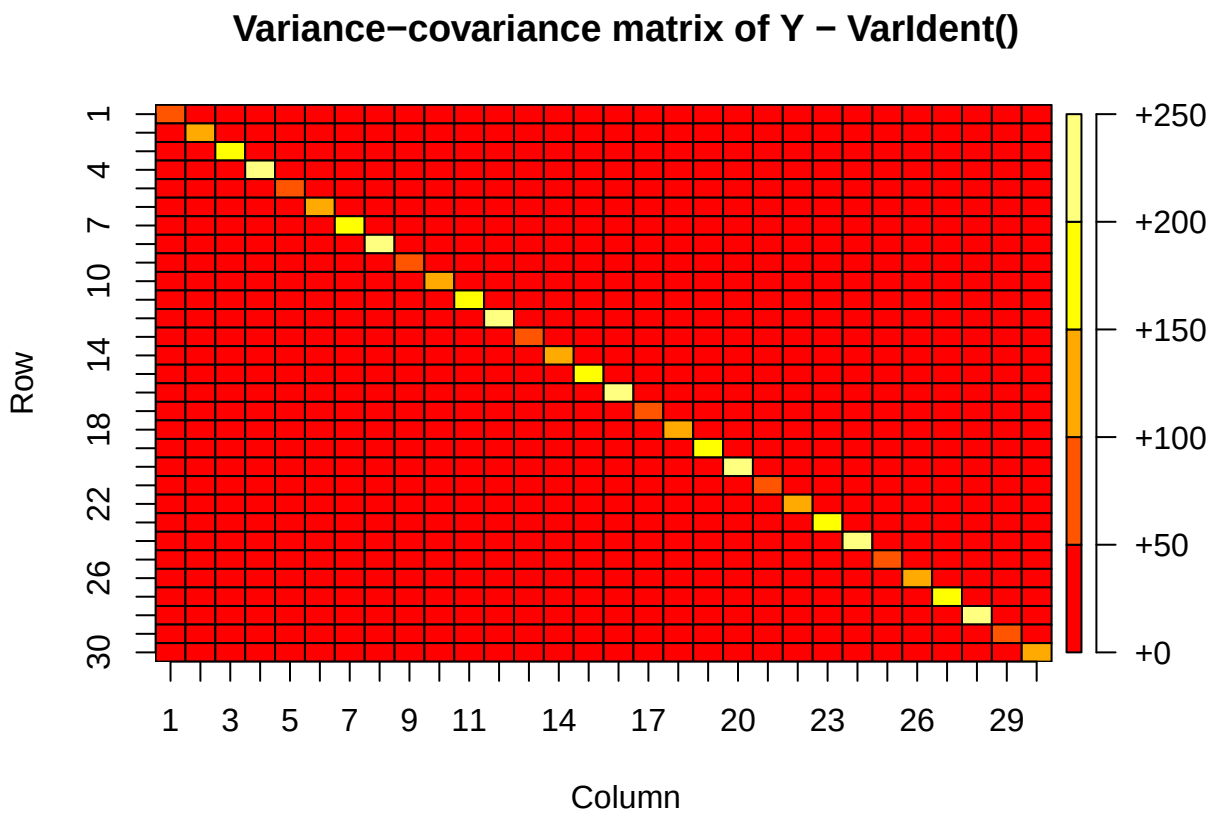

```
fm9.1$modelStruct$varStruct
```

```
## Variance function structure of class varIdent representing
##     4wks    12wks    24wks    52wks
## 1.000000 1.397600 1.664321 1.880852
```

```
intervals(fm9.1, which = "var-cov")  ## 95% CI
```

```
## Approximate 95% confidence intervals
##
##  Variance function:
##          lower     est.    upper
## 12wks 1.226867 1.397600 1.592091
## 24wks 1.457597 1.664321 1.900364
## 52wks 1.640890 1.880852 2.155906
##
##  Residual standard error:
##    lower     est.    upper
## 7.519025 8.244094 9.039081
```

```
# Visualization of Variance-covariance matrix of Y (first 30 observations)
par(mar = c(4,4,4,4))
plot(diag(x=c(1.000000^2*8.244094^2, 1.397600^2*8.244094^2,
              1.664321^2*8.244094^2, 1.880852^2*8.244094^2),
          nrow=30, ncol=30),
     main='Variance-covariance matrix of Y - VarIdent()')
```




```
## To formally test the hypothesis that the variances are timepoint specific,
## we apply the anova() function. The LR test tests the null hypothesis of homoscedasticity.

## The anova() function will take the model objects as arguments, and return an
## ANOVA testing
## whether the more complex model is significantly better at capturing the data
## than the simpler model.

anova(fm9.1, lm1.form)  ## lm1.form is_contained_in fm9.1
```

```
##          Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## fm9.1        1 13 6740.294 6802.104 -3357.147
## lm1.form     2 10 6821.620 6869.166 -3400.810 1 vs 2 87.32618  <.0001
```

```
#! we can do this since one model is contained in the other
#! new model vs old model, and the new one is confirmed to be an improvement

#! How do we compare models quantitatively? using an F-test, comparing the ratio
#! of the sum of their residuals. That for both traditional models.
#! Here we can use an anova instead, as we are comparing different type of models.
```

```
## 2.2 Option 2: VarPower()
## Now that we know the variance is increasing in time, we try a more parsimonious model
#! as before we had to estimate three parameters, the deltas
fm9.2 = update(fm9.1, weights = varPower(form = ~time)) # Var. function; <delta, v_it>-group

#! varPower case gives sigma^2_t = time_t^delta
#! So it gives a structure as
#!            / time1^(2*delta)                        \
#!  sigma^2 * |                time2^(2*delta)         |
#!            |                                ecc     |
#!            \                                    ecc /
#! so we have just one parameter to estimate (where times are 4,12,24,52)
#! it works with same time instants for all the patients

summary(fm9.2)
```

11.1.2.2 2.2 VarPower()

```
## Generalized least squares fit by REML
##   Model: visual ~ -1 + visual0 + time.f + treat.f:time.f
##   Data: armd
##        AIC      BIC    logLik
##   6738.126 6790.427 -3358.063
##
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
##  Parameter estimates:
##     power
## 0.2519332
##
## Coefficients:
##                              Value Std.Error  t-value p-value
## visual0                   0.862917  0.025564 33.75523  0.0000
## time.f4wks                6.276734  1.615396  3.88557  0.0001
## time.f12wks               5.285972  1.747721  3.02449  0.0026
## time.f24wks               1.839097  1.886872  0.97468  0.3300
## time.f52wks              -3.556533  2.123940 -1.67450  0.0944
## time.f4wks:treat.fActive  -2.325312  1.115198 -2.08511  0.0374
## time.f12wks:treat.fActive -3.689296  1.484085 -2.48591  0.0131
## time.f24wks:treat.fActive -3.432676  1.821213 -1.88483  0.0598
## time.f52wks:treat.fActive -4.439566  2.322625 -1.91144  0.0563
##
##  Correlation:
##                           visul0 tm.f4w tm.f12 tm.f24 tm.f52 t.4:.A t.12:.
## time.f4wks                -0.875
## time.f12wks               -0.807  0.705
## time.f24wks               -0.746  0.652  0.601
## time.f52wks               -0.669  0.585  0.540  0.499
## time.f4wks:treat.fActive   0.019 -0.357 -0.016 -0.014 -0.013
## time.f12wks:treat.fActive  0.010 -0.009 -0.420 -0.008 -0.007  0.000
## time.f24wks:treat.fActive  0.007 -0.006 -0.006 -0.465 -0.005  0.000  0.000
## time.f52wks:treat.fActive  0.011 -0.010 -0.009 -0.009 -0.512  0.000  0.000
##                           t.24:.
## time.f4wks
## time.f12wks
## time.f24wks
## time.f52wks
## time.f4wks:treat.fActive
## time.f12wks:treat.fActive
## time.f24wks:treat.fActive
## time.f52wks:treat.fActive  0.000
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -3.9483516 -0.5311949  0.1331253  0.6502082  3.3663349
##
## Residual standard error: 5.974906
## Degrees of freedom: 867 total; 858 residual
```

```r
#! Parameter estimates:
#!   power
#!   0.2519332
#! delta is increasing as we expected

fm9.2$modelStruct$varStruct
```

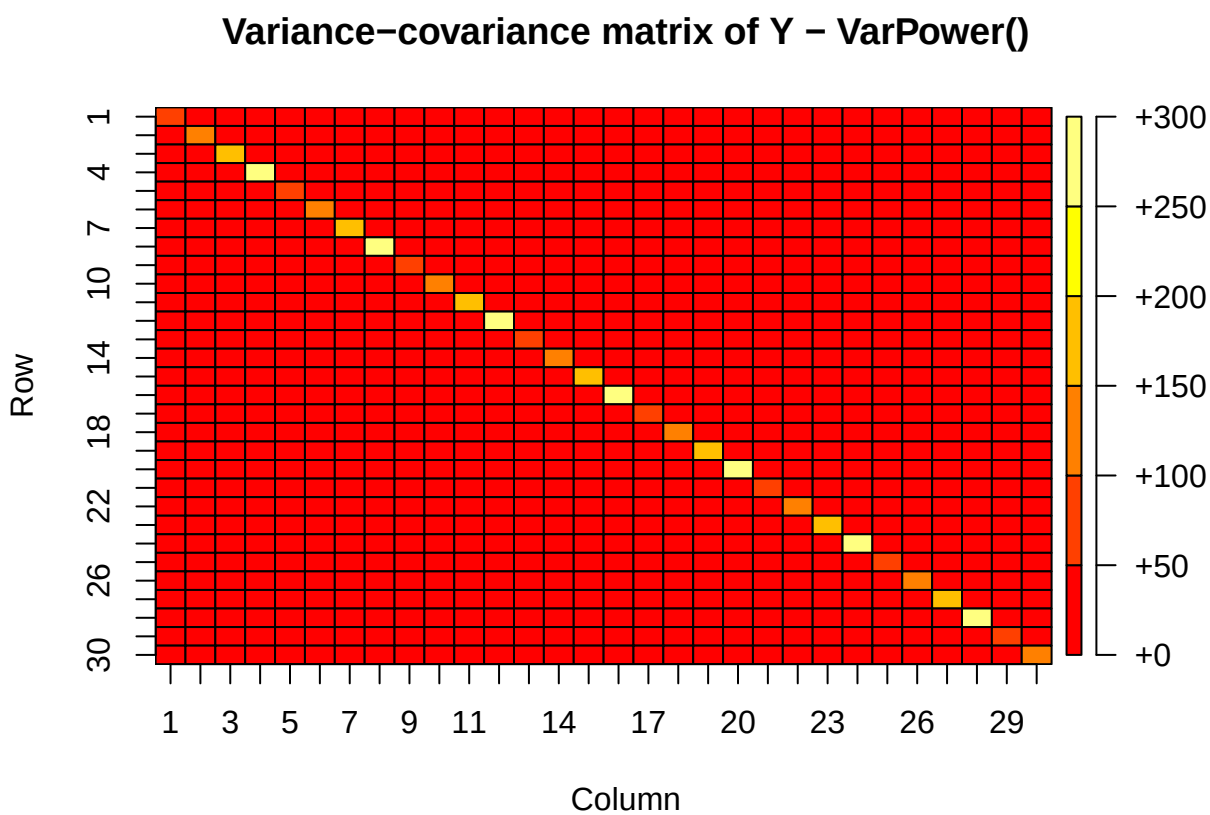
```
## Variance function structure of class varPower representing
##     power
## 0.2519332
```

```r
intervals(fm9.2, which = "var-cov")
```

```
## Approximate 95% confidence intervals
##
##  Variance function:
##           lower      est.     upper
## power 0.1997298 0.2519332 0.3041365
##
##  Residual standard error:
```

```
##    lower     est.    upper
## 5.151231 5.974906 6.930286
```

```
#! CIs, we see we dont contain zero

# Visualization of Variance-covariance matrix of Y (first 30 observations)
par(mar = c(4,4,4,4))
plot(diag(x=c(4^(2*0.2519332)*5.974906^2, 12^(2*0.2519332)*5.974906^2,
              24^(2*0.2519332)*5.974906^2, 52^(2*0.2519332)*5.974906^2), nrow=30, ncol=30),
     main='Variance-covariance matrix of Y - VarPower()')
```




```
# Test of the variance structure: power of time vs. timepoint-specific variances
anova(fm9.2, fm9.1)
```

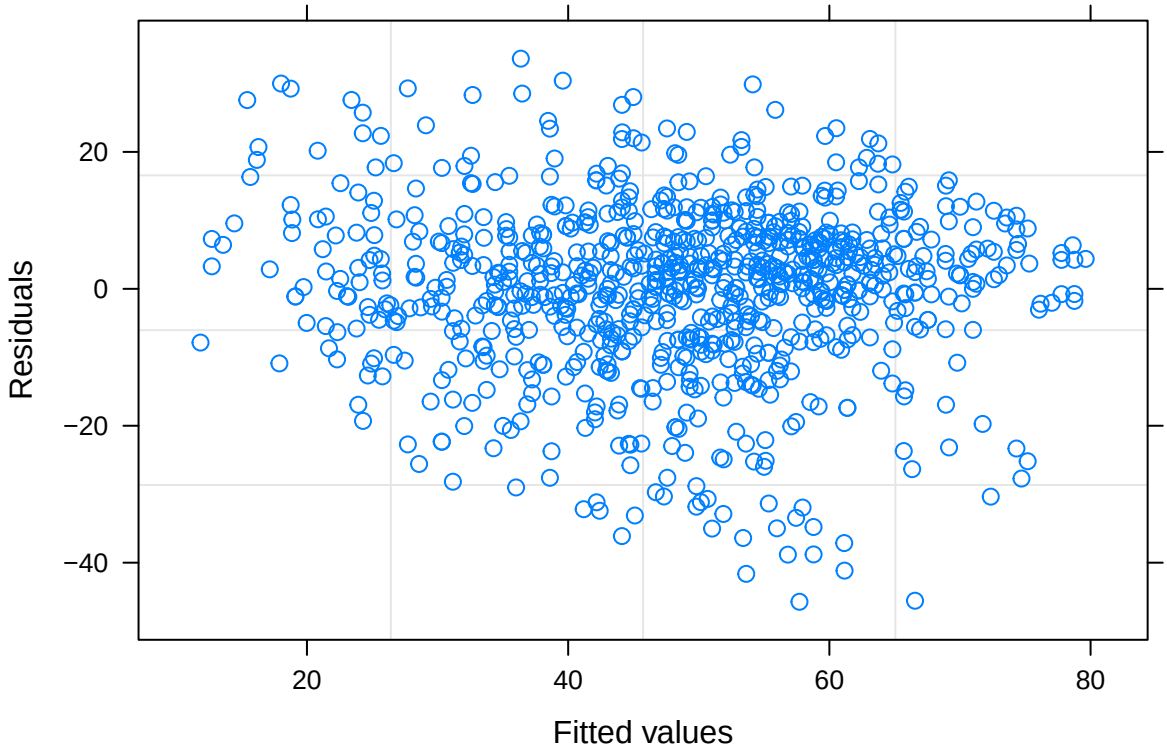
```
##       Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## fm9.2     1 11 6738.126 6790.427 -3358.063
## fm9.1     2 13 6740.294 6802.104 -3357.147 1 vs 2 1.831985  0.4001
```

```
#! anova doable since this is again a subset of the previous
#! low pvalue, we reject a model to better than the other,
#! so are both equivalent, and we select the simplest one

AIC(fm9.2, fm9.1)  # --> fm9.2 is better in terms of AIC and parsimony!
```

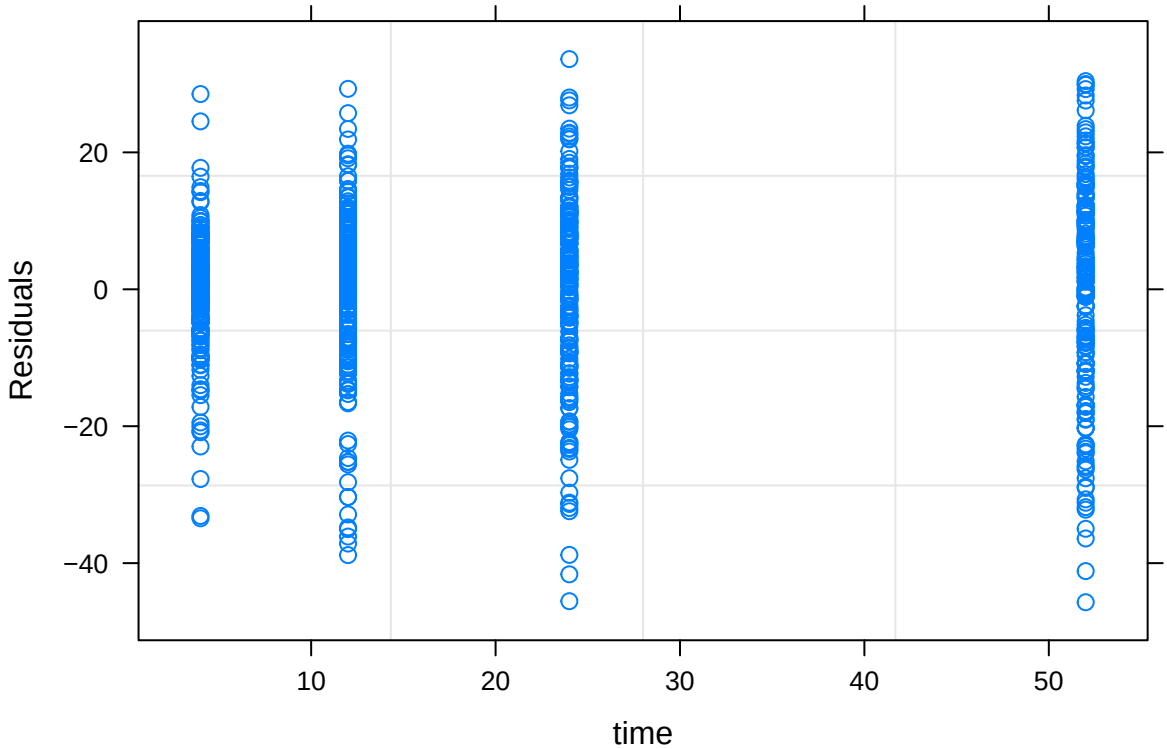
```
##       df      AIC
## fm9.2 11 6738.126
## fm9.1 13 6740.294
```

```
## Residual analysis --> we assess the fit of the model using residual plots.

## raw residuals
plot(fm9.2, resid(., type = "response") ~ fitted(.)) # Raw vs. fitted
```

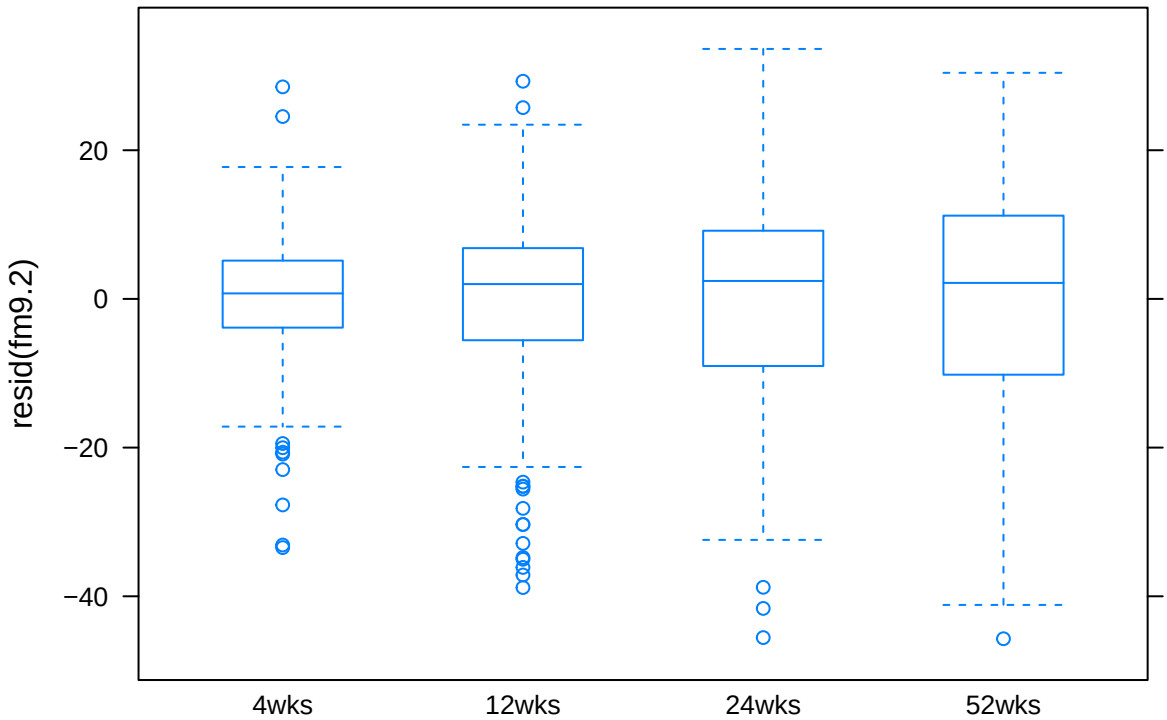
11.1.2.3 Residual analysis



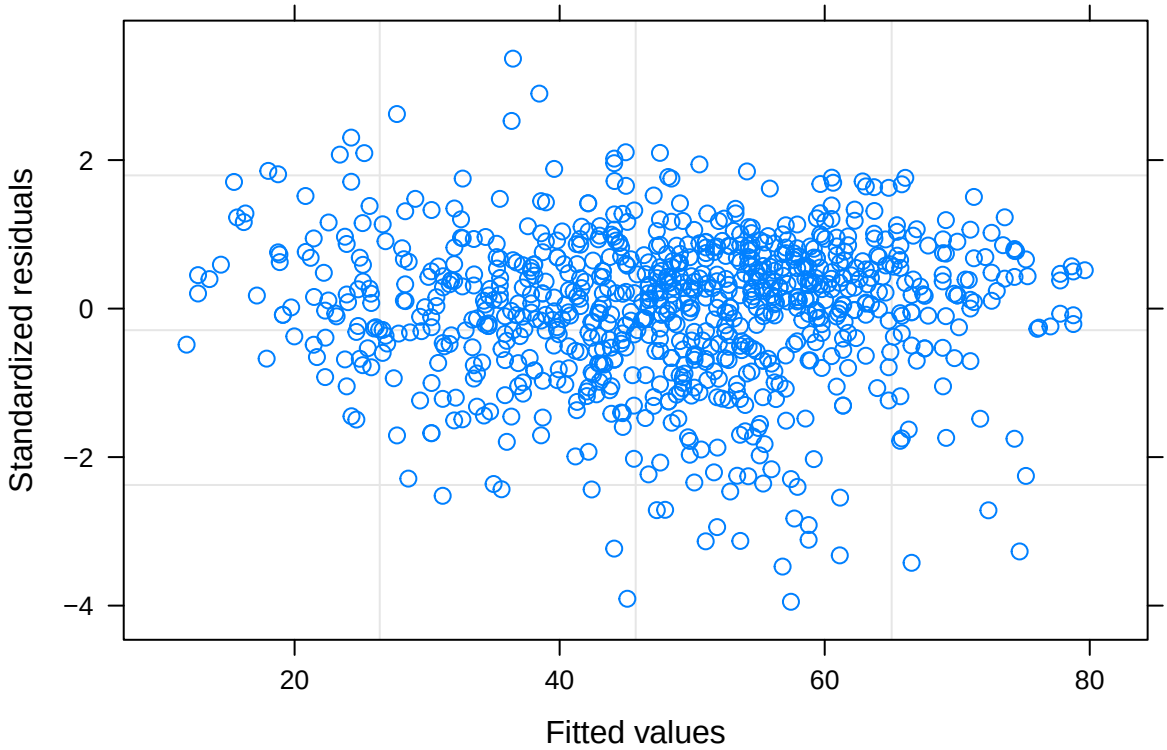

```r
# We observe an asymmetric pattern, with large positive (negative) residuals present
# mainly for small (large) fitted values. But it can be a consequence of the fact
# that raw residuals are intrinsically heteroscedastic and correlated.

plot(fm9.2, resid(., type = "response") ~ time) # Raw vs. time (not shown)
```



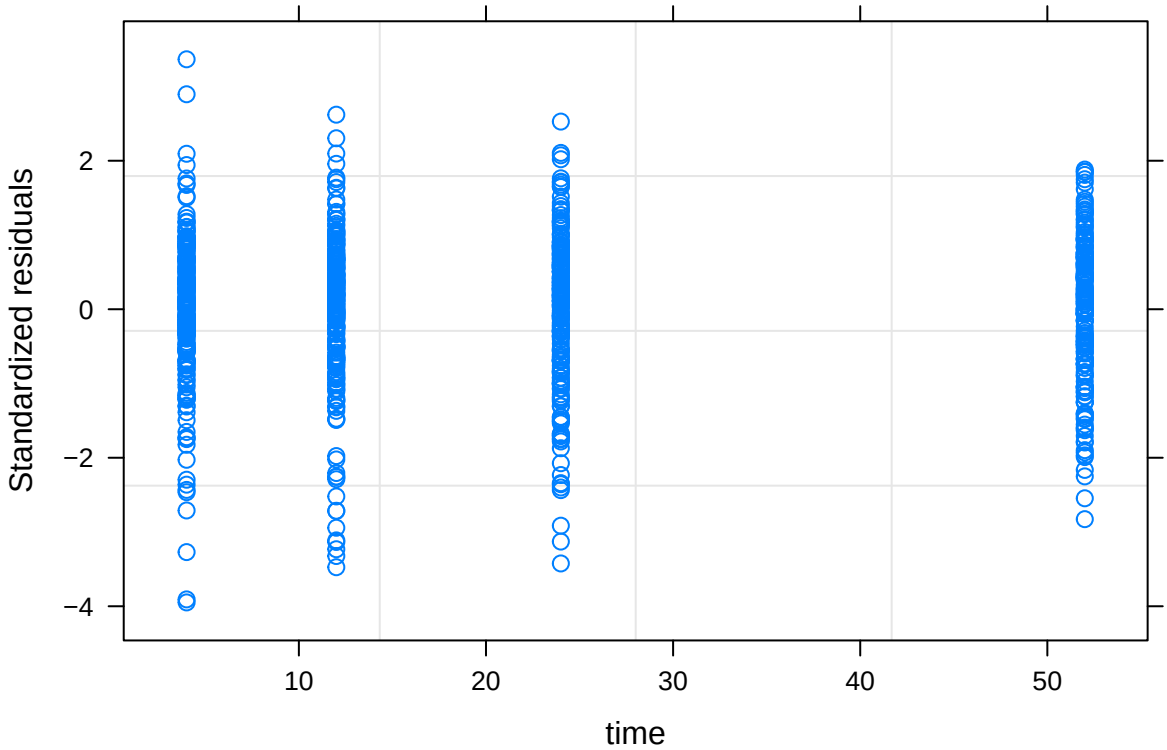
```r
bwplot(resid(fm9.2) ~ time.f, pch = "|", data = armd)
```



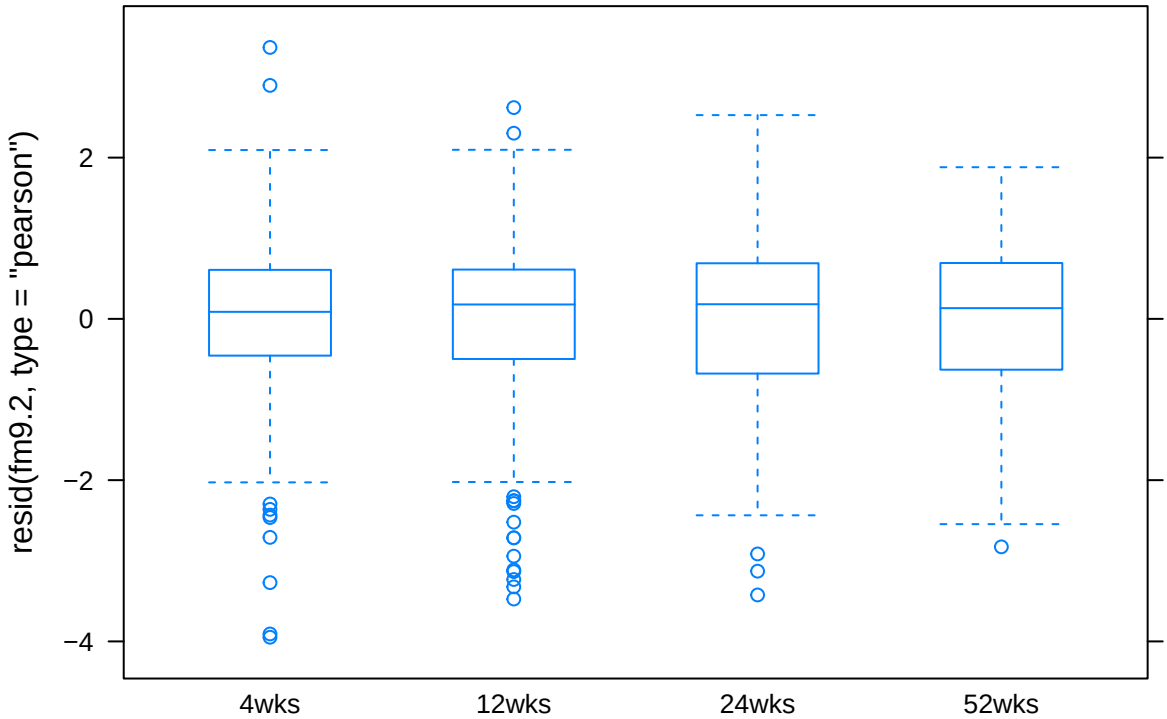

```r
# The boxand-whiskers plots clearly show an increasing variance of the residuals.

## Pearson residuals
## Pearson residuals are obtained from the raw residuals by dividing the latter by an
## estimate of the appropriate residual standard deviation, so they should be more
## homoscedastic

plot(fm9.2, resid(., type = "pearson" ) ~ fitted(.)) # Pearson vs. fitted
```




```
plot(fm9.2,resid(., type = "pearson") ~ time)
```



```r
bwplot( resid(fm9.2, type = "pearson") ~ time.f, # Pearson vs. time.f
        pch = "|", data = armd)
```



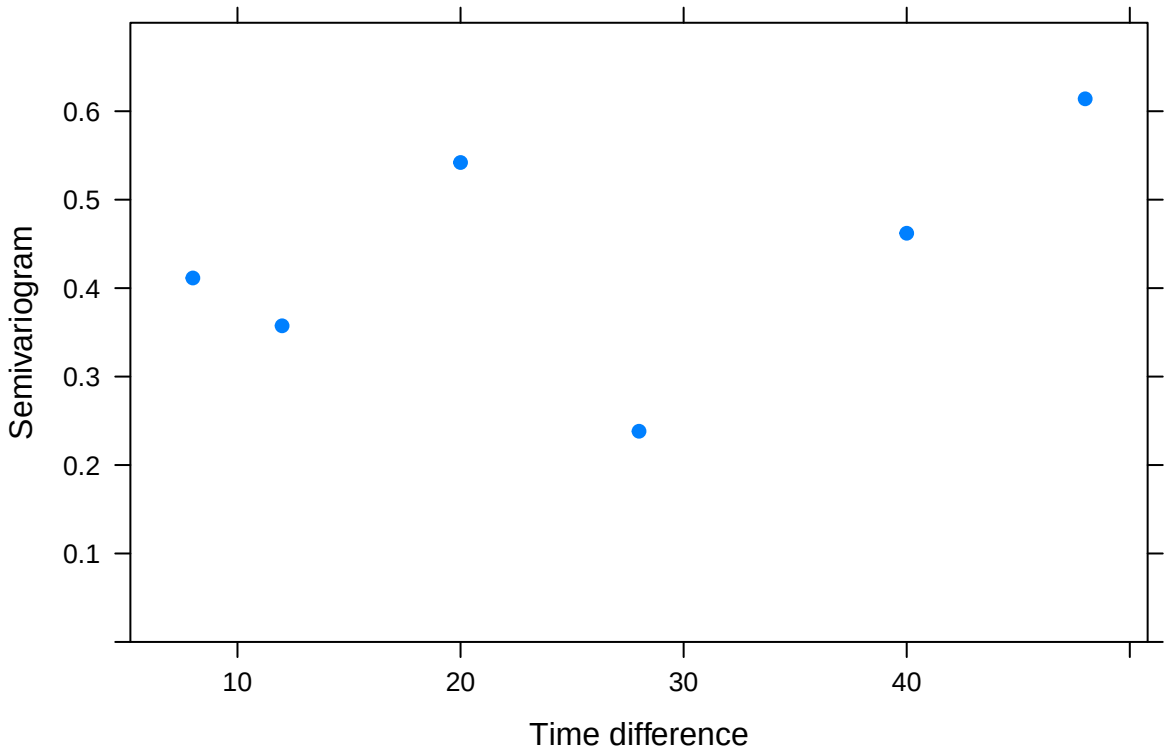

```r
## this plot illustrate the effect of scaling: the variance of the residuals is
## virtually constant.

#! Residuals are still the same, but we know the variance each of them has according to the
#! week time. So after the correction by scaling correctly we see better plots
```

11.1.3 3. Linear Models with heteroscedastic and dependent errors

```r
################################################################################
## 3. Linear models with heteroscedastic and dependent errors

## We now modify the model, so that the visual acuity measurements,
## obtained for the same individual, are allowed to be correlated.

## We can estimate the semivariogram to calculate correlation coefficients between Pearson
## residuals for every pair of timepoints, separately.

# The semivariogram function can be defined as the complement of the correlation function.
#! low semivariogram equiv high correlation

## Variogram per time difference
Vg1 = Variogram(fm9.2, form = ~ time | subject)
Vg1
```

```
##      variog dist n.pairs
## 1 0.4114380    8     224
## 2 0.3573865   12     214
## 3 0.5420185   20     212
## 4 0.2381944   28     190
## 5 0.4620258   40     194
## 6 0.6140054   48     193
```

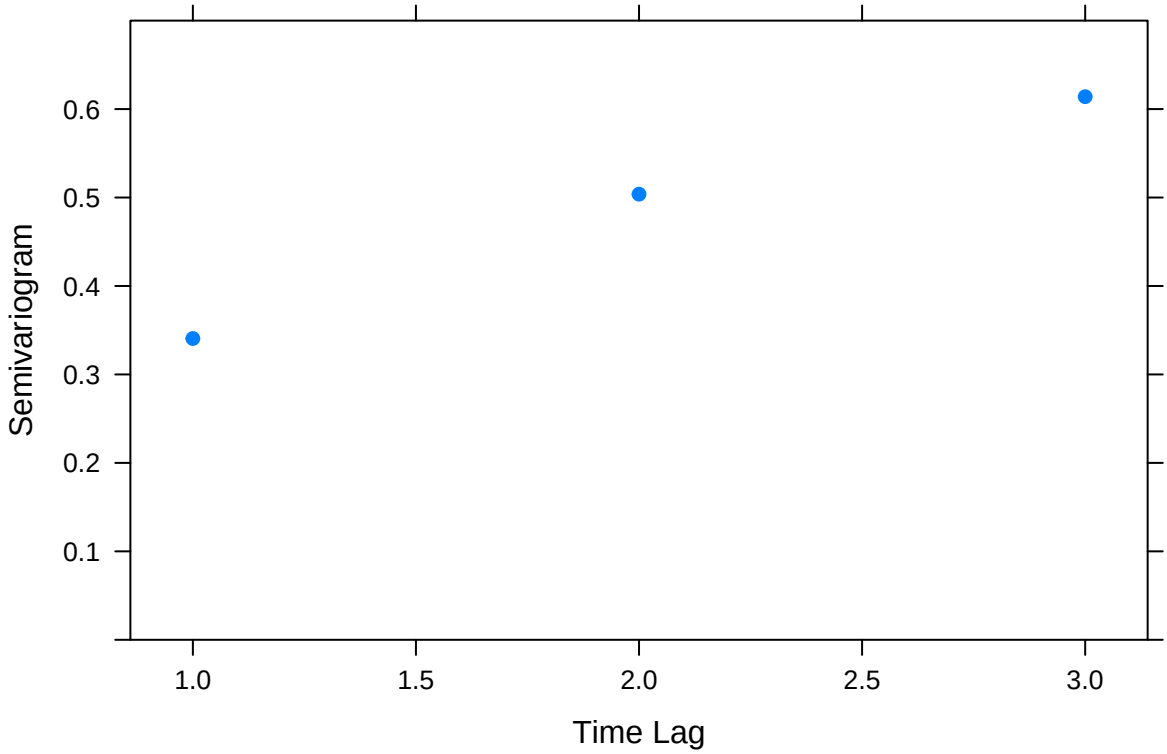
```r
plot(Vg1, smooth = FALSE, xlab = "Time difference",ylim=c(0,0.7), pch=19)
```




```r
#! strange points, then increasing semivariogram
#! So obs will be high correlated if are close in time (as we can expect)

## Variogram per time lag
Vg2 = Variogram(fm9.2, form = ~tp | subject)
Vg2
```

```
##      variog dist n.pairs
## 1 0.3406047    1     628
## 2 0.5037954    2     406
## 3 0.6140054    3     193
```

```r
plot(Vg2, smooth = FALSE, xlab = "Time Lag",ylim=c(0,0.7), pch=19)
```




```
## From these two plots we see that correlation decreases with time lag/difference
## Therefore, a  correlation structure like, eg, a compound symmetry, will most
## likely not fit the data well.
## A more appropriate structure might be, eg, an autoregressive process of order 1 AR(1).

## Nevertheless, for illustrative purposes, we consider a model with a compound symmetry
## correlation structure.
```

```
## 3.1 Correlation 1: CorCompSym()
lm1.form = formula(visual ~ -1 + visual0 + time.f + treat.f:time.f )
fm12.1 = gls(lm1.form, weights = varPower(form = ~time),
        correlation = corCompSymm(form = ~1|subject),
        data = armd)
#! eps_i_VEC ~ Normal(0_VEC, sigma^2 * R_i)
#! R_i = Lambda_i * C_i * Lambda_i
#! - Lambda built with varPower structure (time^delta) -> to model variance
#! - C built if corCompSym structure -> to model correlation

summary(fm12.1)
```

11.1.3.1 3.1 CorCompSym() equal correlation

```
## Generalized least squares fit by REML
##   Model: lm1.form
##   Data: armd
##       AIC      BIC   logLik
##   6456.92 6513.975 -3216.46
##
## Correlation Structure: Compound symmetry
##  Formula: ~1 | subject
##  Parameter estimate(s):
##       Rho
## 0.5732584
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
```

```
##  Parameter estimates:
##     power
## 0.2598167
##
## Coefficients:
##                                Value Std.Error   t-value p-value
## visual0                     0.922747 0.0368207 25.060552  0.0000
## time.f4wks                  2.978881 2.1841555  1.363859  0.1730
## time.f12wks                 1.939807 2.2915716  0.846497  0.3975
## time.f24wks                -1.775009 2.4031989 -0.738603  0.4604
## time.f52wks                -6.981161 2.5808300 -2.705006  0.0070
## time.f4wks:treat.fActive  -2.259712 1.1258703 -2.007080  0.0451
## time.f12wks:treat.fActive -3.588143 1.5073004 -2.380510  0.0175
## time.f24wks:treat.fActive -2.902815 1.8368847 -1.580292  0.1144
## time.f52wks:treat.fActive -5.325264 2.3082963 -2.307011  0.0213
##
##  Correlation:
##                           visul0 tm.f4w tm.f12 tm.f24 tm.f52 t.4:.A t.12:.
## time.f4wks                -0.932
## time.f12wks               -0.888  0.923
## time.f24wks               -0.847  0.898  0.890
## time.f52wks               -0.788  0.858  0.857  0.848
## time.f4wks:treat.fActive   0.027 -0.280 -0.208 -0.233 -0.261
## time.f12wks:treat.fActive  0.018 -0.161 -0.337 -0.225 -0.253  0.565
## time.f24wks:treat.fActive  0.016 -0.157 -0.195 -0.383 -0.249  0.556  0.559
## time.f52wks:treat.fActive  0.012 -0.149 -0.186 -0.212 -0.434  0.541  0.543
##                           t.24:.
## time.f4wks
## time.f12wks
## time.f24wks
## time.f52wks
## time.f4wks:treat.fActive
## time.f12wks:treat.fActive
## time.f24wks:treat.fActive
## time.f52wks:treat.fActive  0.539
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -3.9567690 -0.4972270  0.1395999  0.6310208  3.4664509
##
## Residual standard error: 5.981515
## Degrees of freedom: 867 total; 858 residual
```

```r
intervals(fm12.1, which = "var-cov")
```

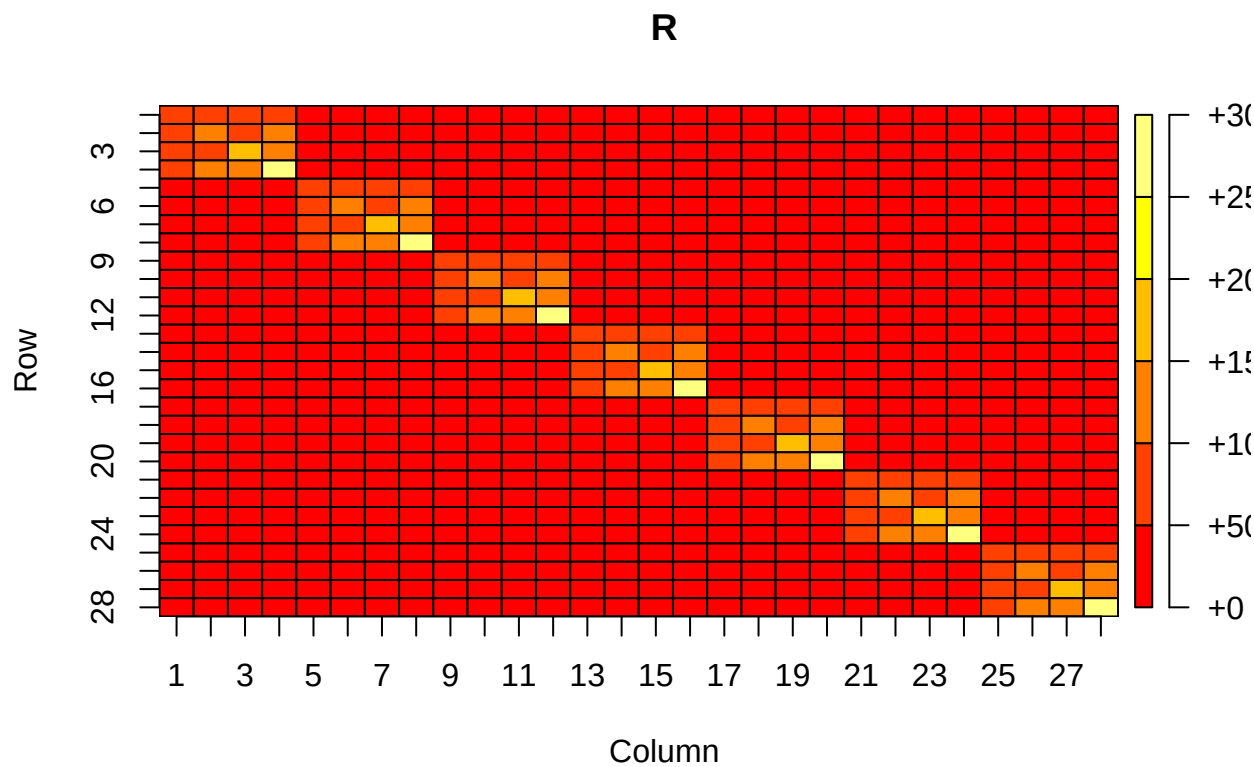
```
## Approximate 95% confidence intervals
##
##  Correlation structure:
##         lower      est.     upper
## Rho 0.5044681 0.5732584 0.6366409
##
##  Variance function:
##           lower      est.     upper
## power 0.2158106 0.2598167 0.3038228
##
##  Residual standard error:
##    lower     est.    upper
## 5.235662 5.981515 6.833620
```

```r
# With the estimates of rho, sigma and delta we can estimate the var-cov matrix

# The marginal variance-covariance structure
fm12.1vcov = getVarCov(fm12.1, individual = "2")  #estimate of R_i, eg on unit i=2
nms = c("4wks", "12wks", "24wks", "52wks")
dnms = list(nms, nms) # Dimnames created
dimnames(fm12.1vcov) = dnms # Dimnames assigned
print(fm12.1vcov)
```

```
## Marginal variance covariance matrix
##         4wks   12wks   24wks   52wks
## 4wks  73.531  56.077  67.143  82.081
## 12wks 56.077 130.140  89.323 109.200
## 24wks 67.143  89.323 186.560 130.740
```

```
## 52wks 82.081 109.200 130.740 278.810
##   Standard Deviations: 8.575 11.408 13.659 16.698
```

```
## on the diagonal we have (5.981515^2)*TIME^(2*0.2598167)
## out of the diagonal we have (5.981515^2)*TIME_1^(0.2598167)*TIME_2^(0.2598167)*rho

## Visualization of the marginal variance-covariance matrix of Y
R_i = rbind(c( 73.531 , 56.077 , 67.143 , 82.081),
            c(56.077, 130.140 , 89.323, 109.200),
            c(67.143,  89.323, 186.560, 130.740),
            c(82.081, 109.200, 130.740, 278.810))
#! diagonal obtained by the time^delta
#! out of diagonal obtained by correlation stuff

R = matrix(0, nrow=28, ncol=28)
for(i in 0:6){
        R[(i*4+1):(i*4+4),(i*4+1):(i*4+4)] = R_i
}
plot(R)
```




```
print(cov2cor(fm12.1vcov), corr = TRUE, stdevs = FALSE)  ## Estimate of C_i (correlation matrix)
```

```
## Marginal correlation matrix
##          4wks   12wks   24wks   52wks
## 4wks  1.00000 0.57326 0.57326 0.57326
## 12wks 0.57326 1.00000 0.57326 0.57326
## 24wks 0.57326 0.57326 1.00000 0.57326
## 52wks 0.57326 0.57326 0.57326 1.00000
```

```
## Test of independence vs. compound-symmetry correlation structure
anova(fm9.2, fm12.1) # M9.2 C M12.1
```

```
##        Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## fm9.2      1 11 6738.126 6790.427 -3358.063
## fm12.1     2 12 6456.920 6513.975 -3216.460 1 vs 2 283.2067  <.0001
```

```r
# The result of the LR test is clearly statistically significant, indicating
# the importance of the adjustment for the correlation in modeling the data
```

```r
## 3.2 Correlation 2: AR(1)
#! 1    rho  rho^2 rho^3
#! rho   1   rho   rho^2
#! rho^2 rho 1     rho
#! rho^3 rho^3 rho 1
#! structure now of the cov matrix

fm12.2 = update(fm9.2,
                correlation = corAR1(form = ~tp|subject),
                data = armd)
summary(fm12.2)
```

11.1.3.2 3.2 AR(1) decreasing correlation

```
## Generalized least squares fit by REML
##   Model: visual ~ -1 + visual0 + time.f + treat.f:time.f
##   Data: armd
##        AIC      BIC    logLik
##   6396.911 6453.966 -3186.455
##
## Correlation Structure: ARMA(1,0)
##  Formula: ~tp | subject
##  Parameter estimate(s):
##      Phi1
## 0.6573069
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
##  Parameter estimates:
##     power
## 0.2311874
##
## Coefficients:
##                              Value Std.Error   t-value p-value
## visual0                   0.882255 0.0373833 23.600255  0.0000
## time.f4wks                5.225502 2.2192976  2.354575  0.0188
## time.f12wks               4.173269 2.3145900  1.803027  0.0717
## time.f24wks               0.506153 2.4080635  0.210191  0.8336
## time.f52wks              -4.848195 2.5648488 -1.890246  0.0591
## time.f4wks:treat.fActive  -2.277328 1.1497227 -1.980763  0.0479
## time.f12wks:treat.fActive -3.601515 1.4894624 -2.417996  0.0158
## time.f24wks:treat.fActive -3.125211 1.7817577 -1.754005  0.0798
## time.f52wks:treat.fActive -4.744491 2.2102239 -2.146611  0.0321
##
##  Correlation:
##                          visul0 tm.f4w tm.f12 tm.f24 tm.f52 t.4:.A t.12:.
## time.f4wks               -0.931
## time.f12wks              -0.893  0.939
## time.f24wks              -0.858  0.878  0.916
## time.f52wks              -0.807  0.810  0.831  0.885
## time.f4wks:treat.fActive  0.027 -0.281 -0.231 -0.176 -0.135
## time.f12wks:treat.fActive  0.019 -0.184 -0.332 -0.250 -0.188  0.649
## time.f24wks:treat.fActive  0.016 -0.122 -0.217 -0.371 -0.274  0.419  0.645
## time.f52wks:treat.fActive  0.014 -0.081 -0.142 -0.237 -0.417  0.265  0.408
##                          t.24:.
## time.f4wks
## time.f12wks
## time.f24wks
## time.f52wks
## time.f4wks:treat.fActive
## time.f12wks:treat.fActive
## time.f24wks:treat.fActive
## time.f52wks:treat.fActive  0.626
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -3.8420789 -0.5181778  0.1473721  0.6473900  3.2994278
##
```

```
## Residual standard error: 6.356295
## Degrees of freedom: 867 total; 858 residual
```

```
intervals(fm12.2, which = "var-cov")
```

```
## Approximate 95% confidence intervals
##
##  Correlation structure:
##          lower      est.     upper
## Phi1 0.6039751 0.6573069 0.7047785
##
##  Variance function:
##           lower      est.     upper
## power 0.1831995 0.2311874 0.2791752
##
##  Residual standard error:
##    lower     est.    upper
## 5.503595 6.356295 7.341109
```

```
# The marginal variance-covariance structure
fm12.2vcov = getVarCov(fm12.2, individual = "2")  #Estimate of R_i, e.g. i=2
dimnames(fm12.2vcov) = dnms
fm12.2vcov
```

```
## Marginal variance covariance matrix
##         4wks   12wks   24wks   52wks
## 4wks  76.698  64.992  50.144  39.411
## 12wks 64.992 127.470  98.346  77.296
## 24wks 50.144  98.346 175.620 138.030
## 52wks 39.411  77.296 138.030 251.100
##   Standard Deviations: 8.7578 11.29 13.252 15.846
```

```
## on the diagonal we have (6.356295^2)*TIME^(2*0.2311874)
## out of the diagonal we have (6.35629^2)*4^(0.2311874)*12^(0.2311874)*0.6573069
##                             (6.35629^2)*4^(0.2311874)*24^(0.2311874)*0.6573069^2...


fm12.2cor = cov2cor(fm12.2vcov)  #Estimate of C_i
print(fm12.2cor, digits = 2,
        corr = TRUE, stdevs = FALSE)
```

```
## Marginal correlation matrix
##       4wks 12wks 24wks 52wks
## 4wks  1.00  0.66  0.43  0.28
## 12wks 0.66  1.00  0.66  0.43
## 24wks 0.43  0.66  1.00  0.66
## 52wks 0.28  0.43  0.66  1.00
```

```
#! Now we have different variances and also different covariances among units of the same group
#! The structure is indeed 1 rho rho^2 rho^3

# Compound-symmetry vs. autoregressive correlation (but now are nonnested models)
#! but at least we can compare AIC or BIC for both (lower is better)
anova(fm12.1, fm12.2)
```

```
##        Model df      AIC      BIC    logLik
## fm12.1     1 12 6456.920 6513.975 -3216.460
## fm12.2     2 12 6396.911 6453.966 -3186.455
```

```
## we prefer AR(1)
```

```
## 3.3 Correlation 3: general correlation structure
#! Now the structure is that the correlation are all different
#! 1     rho12 rho13 rho14
#! rho12 1     rho23 rho24
#! ecc
#! with all rho_ij different
fm12.3 = update(fm12.2, correlation = corSymm(form = ~tp|subject),
                ## the variance function is still VarPower()
                data = armd)
summary(fm12.3)
```

11.1.3.3 3.3 general no contraints on the correlation

```
## Generalized least squares fit by REML
##   Model: visual ~ -1 + visual0 + time.f + treat.f:time.f
##   Data: armd
##       AIC      BIC  logLik
##   6387.2 6468.028 -3176.6
##
## Correlation Structure: General
##  Formula: ~tp | subject
##  Parameter estimate(s):
##  Correlation:
##   1     2     3
## 2 0.582
## 3 0.448 0.651
## 4 0.301 0.531 0.766
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
##  Parameter estimates:
##     power
## 0.2712624
##
## Coefficients:
##                              Value Std.Error   t-value p-value
## visual0                   0.891363  0.036160 24.650550  0.0000
## time.f4wks                4.730782  2.142862  2.207693  0.0275
## time.f12wks               3.673184  2.252921  1.630409  0.1034
## time.f24wks               0.002312  2.367054  0.000977  0.9992
## time.f52wks              -5.332287  2.555316 -2.086743  0.0372
## time.f4wks:treat.fActive -2.290808  1.097764 -2.086794  0.0372
## time.f12wks:treat.fActive -3.602429  1.487225 -2.422248  0.0156
## time.f24wks:treat.fActive -3.127500  1.824666 -1.714013  0.0869
## time.f52wks:treat.fActive -4.947308  2.310109 -2.141590  0.0325
##
##  Correlation:
##                           visul0 tm.f4w tm.f12 tm.f24 tm.f52 t.4:.A t.12:.
## time.f4wks                -0.933
## time.f12wks               -0.887  0.924
## time.f24wks               -0.844  0.873  0.908
## time.f52wks               -0.783  0.795  0.843  0.910
## time.f4wks:treat.fActive   0.027 -0.278 -0.212 -0.189 -0.148
## time.f12wks:treat.fActive  0.018 -0.163 -0.339 -0.257 -0.239  0.574
## time.f24wks:treat.fActive  0.016 -0.125 -0.220 -0.385 -0.336  0.435  0.639
## time.f52wks:treat.fActive  0.013 -0.084 -0.175 -0.287 -0.438  0.284  0.508
##                           t.24:.
## time.f4wks
## time.f12wks
## time.f24wks
## time.f52wks
## time.f4wks:treat.fActive
## time.f12wks:treat.fActive
## time.f24wks:treat.fActive
## time.f52wks:treat.fActive  0.739
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -4.0328913 -0.5168285  0.1465522  0.6401960  3.4785338
##
## Residual standard error: 5.737927
## Degrees of freedom: 867 total; 858 residual
```

```
intervals(fm12.3, # 95% CIs for rho, delta, sigma
          which = "var-cov")
```

```
## Approximate 95% confidence intervals
##
##  Correlation structure:
##              lower      est.     upper
## cor(1,2) 0.4896328 0.5820577 0.6615479
## cor(1,3) 0.3324015 0.4481986 0.5506836
## cor(1,4) 0.1518214 0.3006238 0.4361013
## cor(2,3) 0.5711740 0.6512205 0.7189975
## cor(2,4) 0.4192969 0.5309634 0.6268031
## cor(3,4) 0.6984709 0.7657795 0.8196585
```

```
##
##  Variance function:
##           lower      est.     upper
## power 0.2190782 0.2712624 0.3234466
##
##  Residual standard error:
##    lower     est.    upper
## 4.953914 5.737927 6.646019
```

```
#! All the rho_ij seem to be different from 0 at 95% confidence

fm12.3vcov = getVarCov(fm12.3, individual = "2")  ## Estimate of R_i (italic)
dimnames(fm12.3vcov) = dnms
fm12.3vcov   # fm12.3vcov[1,1] = sigma_2 * Lambda[1]^2 = 5.737927^2*(4^0.2712624)^2
```

```
## Marginal variance covariance matrix
##         4wks   12wks   24wks   52wks
## 4wks  69.846  54.769  50.897  42.105
## 12wks 54.769 126.760  99.627 100.190
## 24wks 50.897  99.627 184.630 174.380
## 52wks 42.105 100.190 174.380 280.860
##   Standard Deviations: 8.3574 11.259 13.588 16.759
```

```
fm12.3cor = cov2cor(fm12.3vcov)    ## Estimate of C_i
print(fm12.3cor, corr = TRUE, stdevs = FALSE)
```

```
## Marginal correlation matrix
##          4wks   12wks   24wks   52wks
## 4wks  1.00000 0.58206 0.44820 0.30062
## 12wks 0.58206 1.00000 0.65122 0.53096
## 24wks 0.44820 0.65122 1.00000 0.76578
## 52wks 0.30062 0.53096 0.76578 1.00000
```

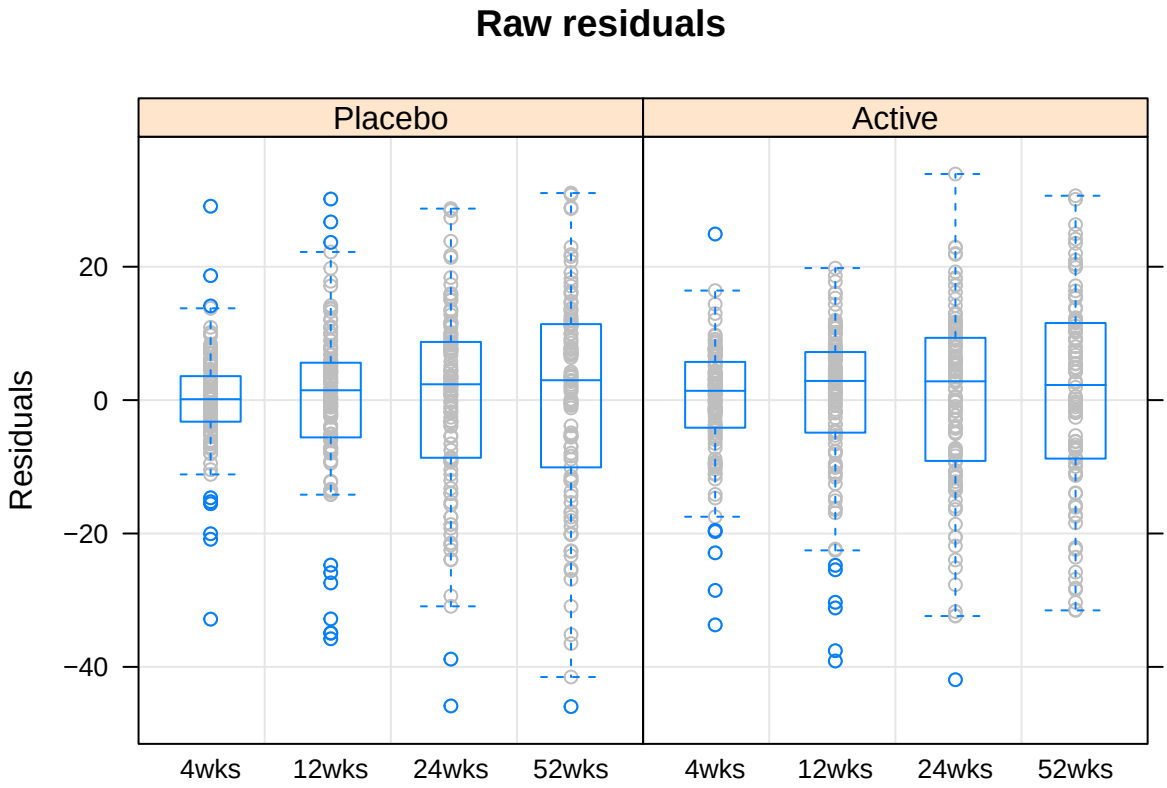
```
## Autoregressive of order 1 vs. a general correlation structure
anova(fm12.2, fm12.3) # M12.2 C M12.3 --> we prefer M12.3
```

```
##        Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## fm12.2     1 12 6396.911 6453.966 -3186.455
## fm12.3     2 17 6387.200 6468.028 -3176.600 1 vs 2 19.71104  0.0014
```

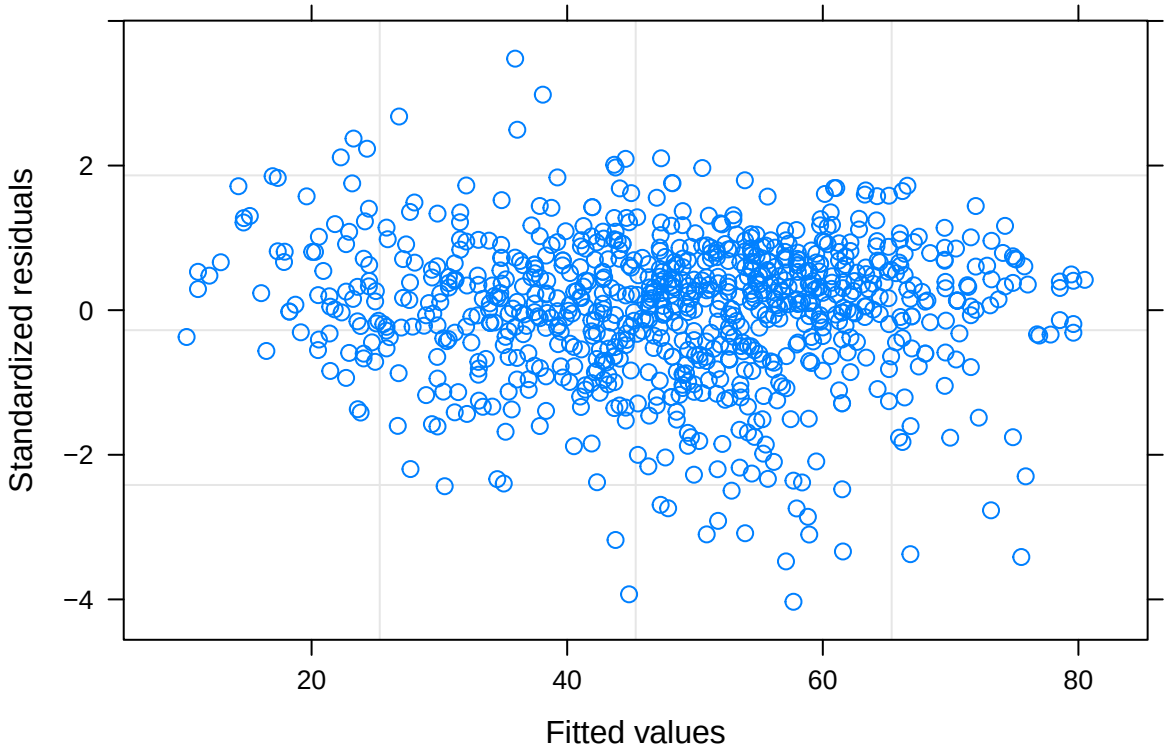
```
#! ie the most complex one seems also the best one. We have lots of parameters
#! to estimate, but the fit is better so it is worth it
```

```
## Model-Fit Diagnostics

# (a) Plots (and boxplots) of raw residuals
panel.bwxplot0 = function(x,y, subscripts, ...){
                       panel.grid(h = -1)
                       panel.stripplot(x, y, col = "grey", ...)
                       panel.bwplot(x, y, pch = "|", ...)
                       }
bwplot(resid(fm12.3) ~ time.f | treat.f,
         panel = panel.bwxplot0,
         ylab = "Residuals", data = armd, main="Raw residuals")
```

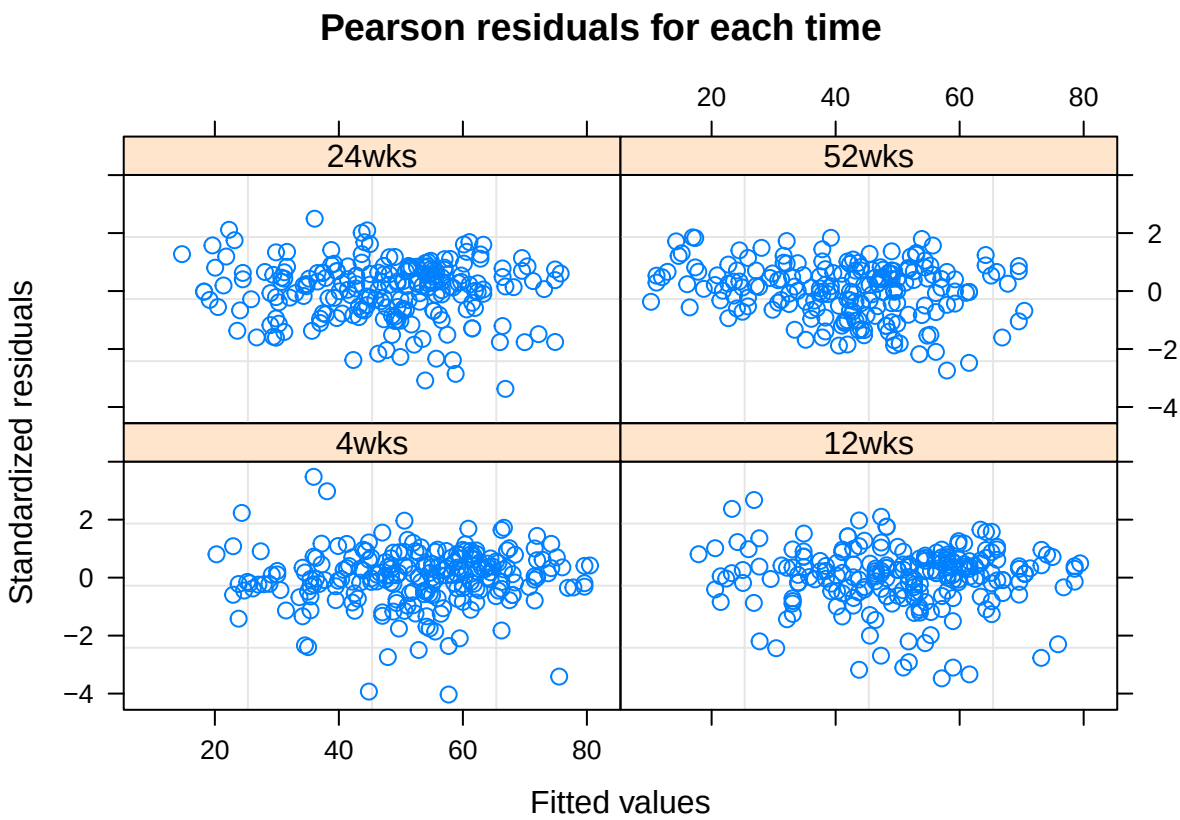
11.1.3.4 Model-Fit Diagnostics



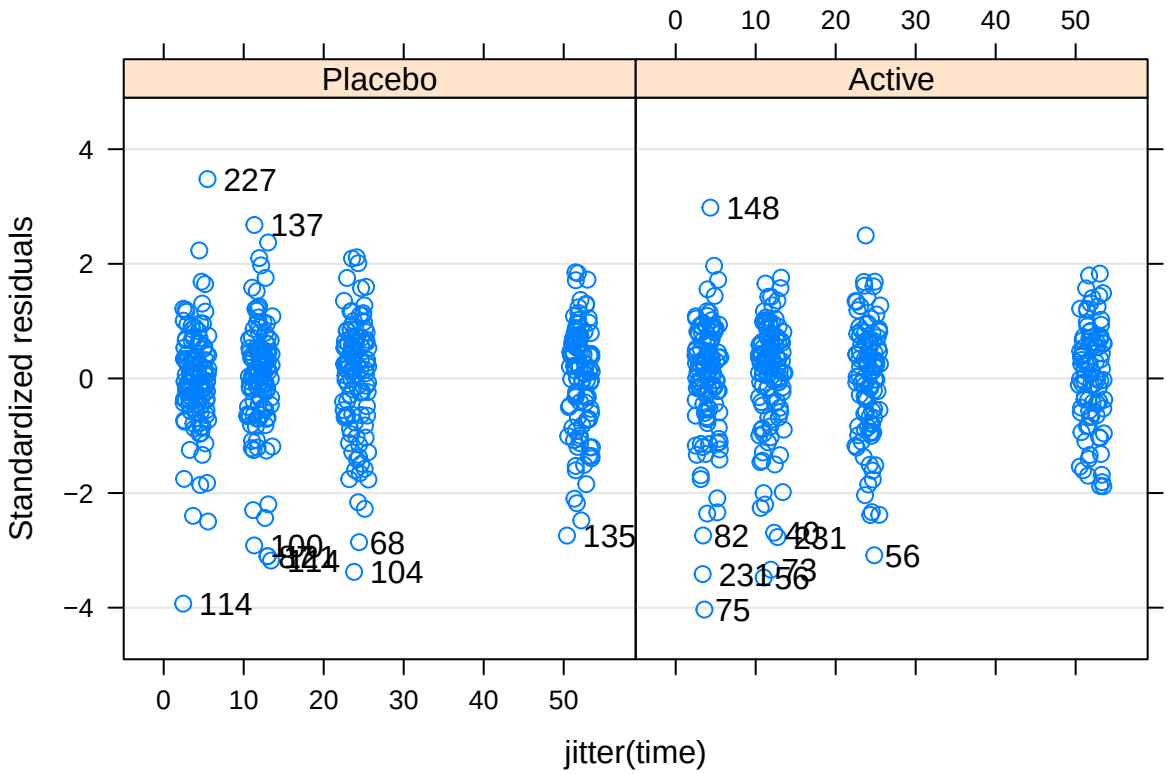

```
# The box-and-whiskers plots clearly show an increasing variance of the residuals with timepoint.
# This reflects the heteroscedasticity.


# (b) Plots of Pearson residuals vs. fitted values
# Pearson residuals are obtained from the raw residuals by dividing the latter by an
# estimate of the appropriate residual standard deviation, so they should be more homoscedastic

plot(fm12.3)
```




```
# Due to the correlation of the residuals corresponding to the measurements obtained
# for the same patient at different timepoints, the plot reveals a pattern, with a few
# large, positive residuals in the upper-left part and a few negative ones in the lower-right part.

## We therefore decide to visualize the residuals for each time instants
plot(fm12.3,
       resid(., type = "p") ~ fitted(.) | time.f, main="Pearson residuals for each time")
```




```
stdres.plot =
        plot(fm12.3, resid(., type = "p") ~ jitter(time) | treat.f,
                id = 0.01, adj = c(-0.3, 0.5 ), grid = FALSE)
plot(update(stdres.plot, # Fig. 12.4
               xlim = c(-5,59), ylim = c(-4.9, 4.9), grid = "h"))
```



```
# The four scatterplots show a somewhat more balanced pattern.
```

11.2 LMM 1 - Age-Related Macular Degeneration

```
library(nlmeU)
library(corrplot)
library(nlme)
library(lattice)
library(plot.matrix)
library(lme4)
library(insight)

# Topics:
#   LINEAR MIXED MODELS WITH HOMOSCEDASTIC RESIDUALS
#   1. Linear Models with random intercept (q=0)
#   2. Linear Models with random intercept + slope (q=1)
#      2.1 general structure of D
#      2.2 diagonal D
#   3. Interpretation of random effects and PVRE
#   4. Prediction
#   5. Diagnostic
#   6. Models comparison
#
#   SUPPLEMENTARY MATERIAL: LINEAR MIXED MODELS WITH HETEROSCEDASTIC RESIDUALS (VarPower())
#   1. Linear Models with random intercept (q=0)
#   2. Linear Models with random intercept + slope (q=1)
#      2.1 general structure of D
#      2.2 diagonal D

#! Y = Z*beta + W*b +eps
#! Z design matrix for the fixed effects
#! W design matrix for the random effects (so still here we have an intercept, and
#! any subset of covariates or their transformation)

data(armd) # Age-Related Macular Degeneration


## Linear Mixed-Effects models (LMM)
## In the LMM approach, the hierarchical structure of the data is directly addressed,
## with random effects that describe the contribution of the variability at different levels
## of the hierarchy to the total variability of the observations.

## Two main R packages:
## 1. 'lme4' --> it does not handle heteroscedastic residuals but it has a lot of "accessories"
## 2. 'nlme' --> it handles heteroscedastic residuals but it has less "accessories"

## We will use lmer() function in lme4 package for LMM models with homogeneous residuals and
## lme() function in nlme package for LMM models with heteroscedastic residuals

##########################################################
#### LINEAR MIXED MODELS WITH HOMOSCEDASTIC RESIDUALS ####
############ lme4 package --> lmer() function ############
##########################################################
```

###1. Linear Models with random intercept (q=0)

```
##############################################################################
# Model 1. Random intercept, homoscedastic residuals

# We now treat time as a numeric variable (as in the last script)
fm16.1mer = lmer(visual ~ visual0 + time * treat.f + (1|subject),
                 # to say "a random intercept that dependes ^^ on the subject
                 data = armd)

#! VAM = visual acuity measurement
#! VAM_ij = beta0 + beta1*VAMAtTime0_ij + beta2*time_ij +
#!           beta3*treat_ij + beta4*time_ij*treat_ij +
#!           + b_i + eps_ij
#! beta are not i dependent
#! b is i dependent (b_i) as we want a random effect to account group i
#! And so matrix W will be just the intercept, a vector of 1
#! We start from the hierarchical structure we see to get how to set the model
```

```r
#
#! So that even if we have individuals with the same covariates (same age, wheight, ecc)
#! our prediction will change according the group in they are. So a nice correction

summary(fm16.1mer)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: visual ~ visual0 + time * treat.f + (1 | subject)
##    Data: armd
##
## REML criterion at convergence: 6578
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.1875 -0.3969  0.0320  0.5514  2.9513
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept) 80.61    8.978
##  Residual             74.43    8.628
## Number of obs: 867, groups:  subject, 234
##
## Fixed effects:
##                    Estimate Std. Error t value
## (Intercept)         9.28808    2.68189   3.463
## visual0             0.82644    0.04467  18.502
## time               -0.21222    0.02293  -9.255
## treat.fActive      -2.42200    1.49997  -1.615
## time:treat.fActive -0.04959    0.03356  -1.478
##
## Correlation of Fixed Effects:
##             (Intr) visul0 time   trt.fA
## visual0     -0.920
## time        -0.185 -0.003
## treat.fActv -0.295  0.022  0.335
## tm:trt.fAct  0.126  0.002 -0.683 -0.476
```

```r
confint(fm16.1mer,oldNames=TRUE)
```

```
## Computing profile confidence intervals ...
```

```
##                        2.5 %      97.5 %
## .sig01              7.9291593 10.01110528
## .sigma              8.1598181  9.11262421
## (Intercept)         4.0454125 14.53187448
## visual0             0.7390912  0.91376265
## time               -0.2571424 -0.16726552
## treat.fActive      -5.3533430  0.50956354
## time:treat.fActive -0.1153109  0.01624458
```

```r
## Var-Cov matrix of fixed-effects
#! The usual ones we get in LMs
vcovb = vcov(fm16.1mer)
vcovb
```

```
## 5 x 5 Matrix of class "dpoMatrix"
##                    (Intercept)       visual0          time treat.fActive
## (Intercept)         7.19252735 -1.102120e-01 -1.135793e-02  -1.186575416
## visual0            -0.11021200  1.995141e-03 -2.950087e-06   0.001487695
## time               -0.01135793 -2.950087e-06  5.257619e-04   0.011518692
## treat.fActive      -1.18657542  1.487695e-03  1.151869e-02   2.249899973
## time:treat.fActive  0.01137481  2.644510e-06 -5.257615e-04  -0.023948166
##                    time:treat.fActive
## (Intercept)              1.137481e-02
## visual0                  2.644510e-06
## time                    -5.257615e-04
## treat.fActive           -2.394817e-02
## time:treat.fActive       1.126389e-03
```

```r
corb = cov2cor(vcovb)
nms = abbreviate(names(fixef(fm16.1mer)), 5)
rownames(corb) = nms
corb
```

```
## 5 x 5 Matrix of class "dpoMatrix"
##                    (Intercept)      visual0         time treat.fActive
## (Intercept)          1.0000000 -0.920028511 -0.184698688   -0.29496669
## visual0             -0.9200285  1.000000000 -0.002880404    0.02220472
## time                -0.1846987 -0.002880404  1.000000000    0.33490915
## treat.fActive       -0.2949667  0.022204722  0.334909145    1.00000000
## time:treat.fActive   0.1263744  0.001764064 -0.683203483   -0.47571466
##                    time:treat.fActive
## (Intercept)               0.126374434
## visual0                   0.001764064
## time                     -0.683203483
## treat.fActive            -0.475714662
## time:treat.fActive        1.000000000
```

```
## Var-Cov matrix of random-effects and errors
vc = VarCorr(fm16.1mer)
print(vc, comp = c("Variance", "Std.Dev."))
```

```
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept) 80.608   8.9782
##  Residual             74.434   8.6275
```

```
sigma2_eps = as.numeric(get_variance_residual(fm16.1mer))
sigma2_eps
```

```
## [1] 74.43401
```

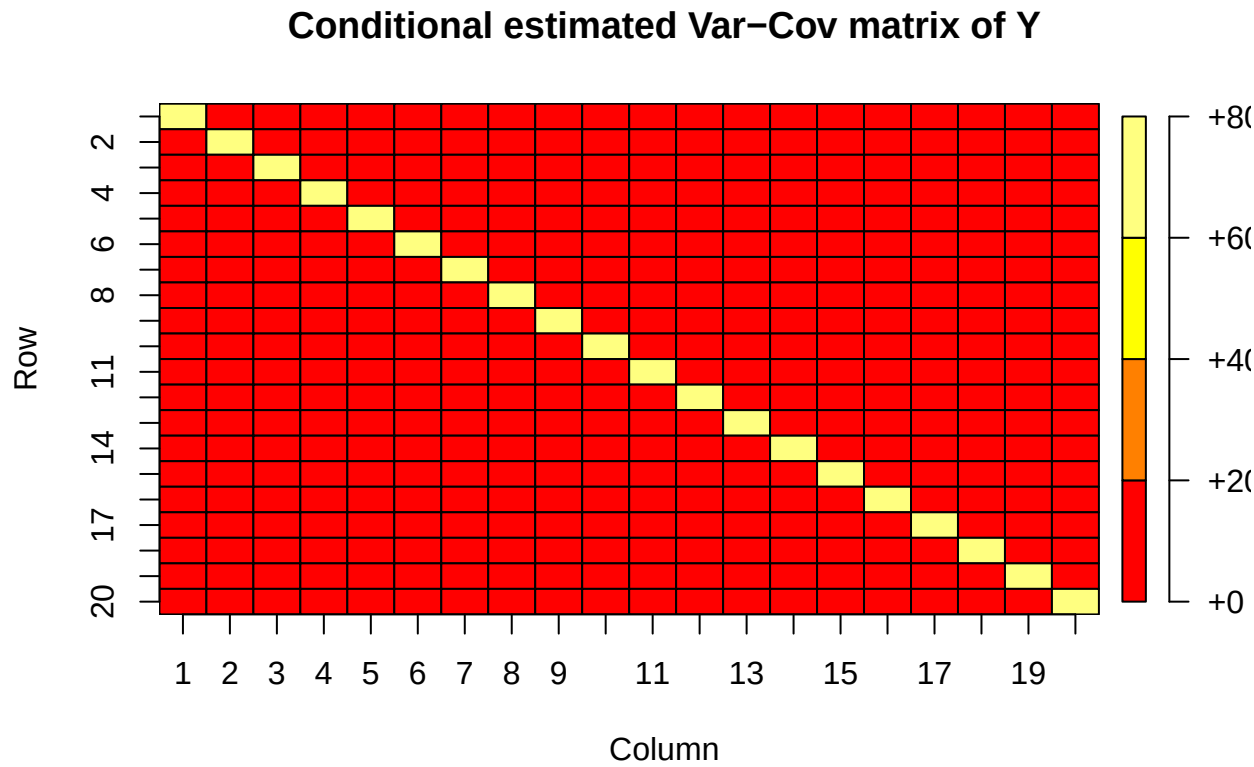
```
sigma2_b = as.numeric(get_variance_random(fm16.1mer))
sigma2_b
```

```
## [1] 80.60828
```

```
## Let's compute the conditional and marginal var-cov matrix of Y
sgma = summary(fm16.1mer)$sigma

A = getME(fm16.1mer, "A") # A  --> N x n, A represents the D (not italic)
I.n = Diagonal(ncol(A)) # IN  --> n x n

## the conditional variance-covariance matrix of Y (diagonal matrix)
SigmaErr = sgma^2 * (I.n)
SigmaErr[3:6, 3:6]  ## visualization of individual 2
```

```
## 4 x 4 diagonal matrix of class "ddiMatrix"
##      [,1]     [,2]     [,3]     [,4]
## [1,] 74.43401        .        .        .
## [2,]        . 74.43401        .        .
## [3,]        .        . 74.43401        .
## [4,]        .        .        . 74.43401
```

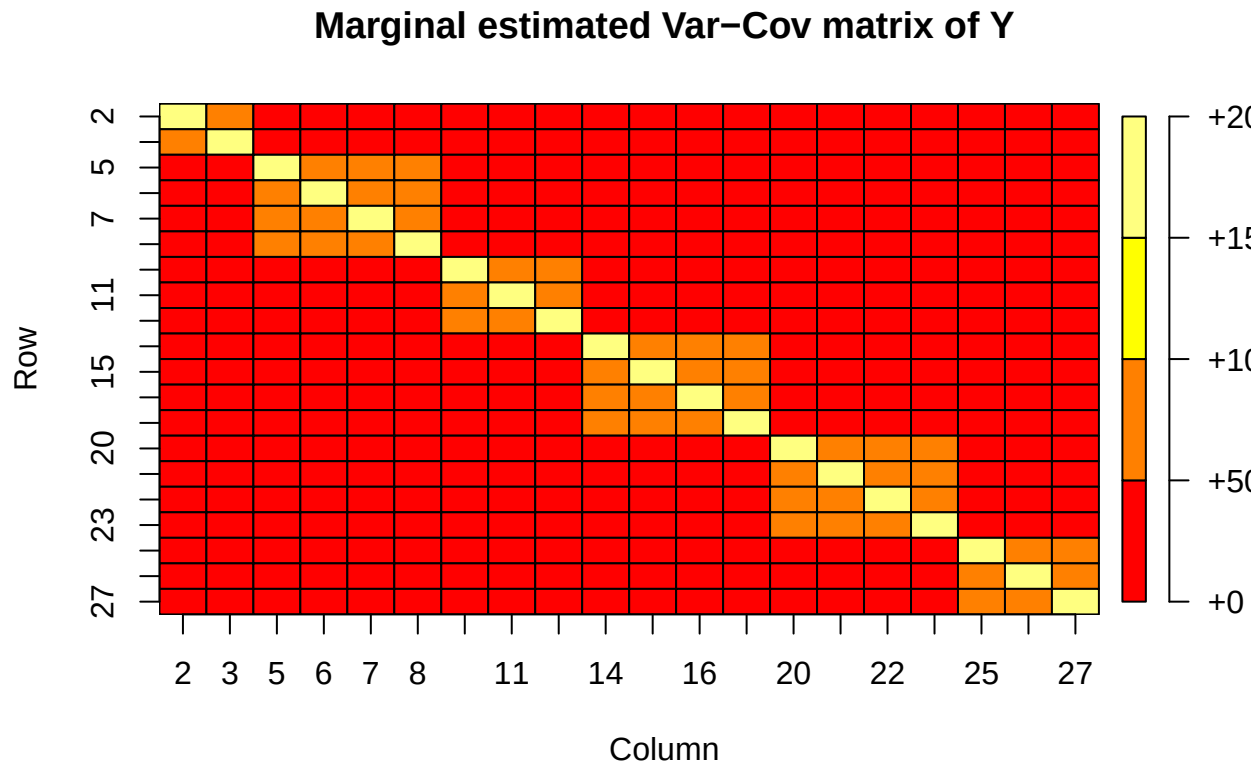
```
# Conditioned to the random effects b_i, we observe the var-cov of the errors
# that are independent and homoscedastic

## we visualize the first 20 rows/columns of the matrix
plot(as.matrix(SigmaErr[1:20,1:20]), main = 'Conditional estimated Var-Cov matrix of Y')
```




```
## the marginal variance-covariance matrix of Y (block-diagonal matrix)
V = sgma^2 * (I.n + crossprod(A))
# V = s^2*(I_N+A*A) --> s^2*(I_N) is the error
# part, s^2*(A*A) is the random effect part

V[3:6, 3:6]  #-> V is a block-diagional matrix, the marginal var-cov matrix
```

```
## 4 x 4 sparse Matrix of class "dsCMatrix"
##           5         6         7         8
## 5 155.04229  80.60828  80.60828  80.60828
## 6  80.60828 155.04229  80.60828  80.60828
## 7  80.60828  80.60828 155.04229  80.60828
## 8  80.60828  80.60828  80.60828 155.04229
```

```
# visualization of the first 20 rows/columns
plot(as.matrix(V[1:20,1:20]), main = 'Marginal estimated Var-Cov matrix of Y')
```




```
#! where now we see how measurements of the same subjects are correlated


# Another way to interpret the variance output is to note percentage of the subject
# variance out  of the total, ie the Percentage of Variance explained by the
# Random Effect (PVRE).
# This is also called the intraclass correlation (ICC), because it is also an estimate
# of the within cluster correlation.
```

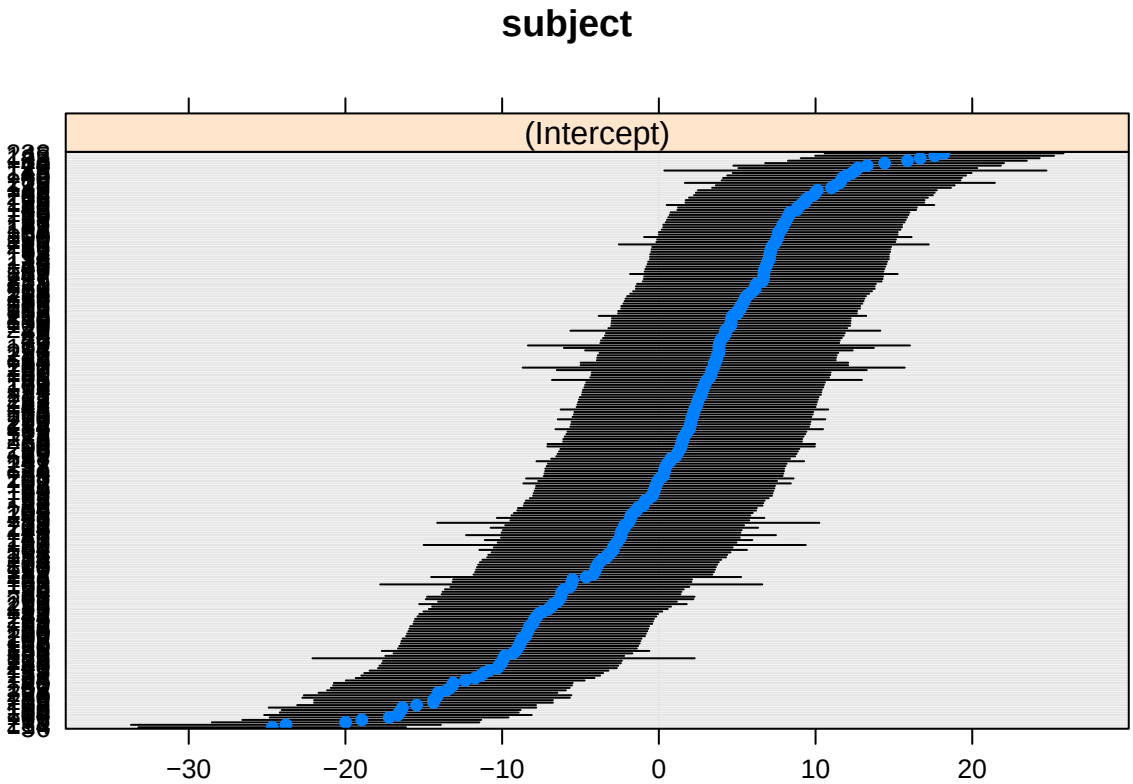
```
#! PVRE high -> the hierarchy/grouping is explaining a lot our response/target variable
#! (after having considered the contribution of the fixed effects part)
PVRE = sigma2_b/(sigma2_b+sigma2_eps)
PVRE # 51% is very high!
```

11.2.0.0.1 PVRE

```
## [1] 0.5199116
```

```
## visualization of the random intercepts with their 95% confidence intervals
# Random effects: b_0i for i=1,...,234s
dotplot(ranef(fm16.1mer, condVar=T))
```

```
## $subject
```




```
#! On y we have all the patients
#! low b_i -> their VAM is lower wrt the average of the population
#! high b_i -> their VAM is higher wrt the average of the population
#! If CI does not contain zero -> that effect is surely significant

# The dotplot shows the point and interval estimates for the random effects,
# ordering them and highlighting which are significantly different from the mean (0)
```

```
# Prediction
#-------------
# Let's now examine standard predictions vs. subject-specific predictions.
# As with most R models, we can use the predict function on the model object.

# Prediction from regression model
lm1 = lm(visual ~ -1 + visual0 + time.f + treat.f:time.f, data = armd )
#! -1 removes the intercept (so we get a model passing through the origin)
summary(lm1)
```

11.2.0.0.2 Prediction

```
##
## Call:
## lm(formula = visual ~ -1 + visual0 + time.f + treat.f:time.f,
##     data = armd)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -45.210  -6.459   1.532   7.512  33.283
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## visual0                   0.83037    0.02842  29.213  < 2e-16 ***
## time.f4wks                8.07531    1.94341   4.155 3.58e-05 ***
## time.f12wks               7.08066    1.94066   3.649  0.00028 ***
## time.f24wks               3.63022    1.95316   1.859  0.06342 .
## time.f52wks              -1.74643    1.98952  -0.878  0.38029
```

```
## time.f4wks:treat.fActive  -2.35278    1.62894  -1.444  0.14900
## time.f12wks:treat.fActive -3.70852    1.64378  -2.256  0.02432 *
## time.f24wks:treat.fActive -3.44915    1.69399  -2.036  0.04205 *
## time.f52wks:treat.fActive -4.47345    1.77811  -2.516  0.01206 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.38 on 858 degrees of freedom
## Multiple R-squared:  0.9432, Adjusted R-squared:  0.9426
## F-statistic:  1583 on 9 and 858 DF,  p-value: < 2.2e-16
```

```
predict_lm = predict(lm1)
head(predict_lm)
```

```
##        2        3        5        6        7        8
## 54.71451 52.36411 59.69675 57.34635 54.15527 47.75433
```

```
# Prediction from mixed model on the training set:
# 1) Without random effects ->  re.form=NA
predict_no_re = predict(fm16.1mer, re.form=NA)
head(predict_no_re) # (almost) same predictions
```

```
##        2        3        5        6        7        8
## 54.57880 52.48435 59.53744 57.44299 54.30131 46.97073
```

```
#! Similar to the ones obtained with the classical lm

# 2) With random effects
predict_re = predict(fm16.1mer)
head(predict_re)
```

```
##        2        3        5        6        7        8
## 52.16273 50.06828 67.00129 64.90684 61.76516 54.43457
```

```
#! Now the values are quite different


# Prediction from mixed model on a test observation from a subject present in the training set:
test.data= data.frame(subject= '234', treat.f='Active', visual0= 63, time = 12)

# 1) Without random effects ->  re.form=NA
predict_no_re = predict(fm16.1mer, newdata = test.data, re.form=NA)
predict_no_re
```

```
##        1
## 55.79011
```

```
# (9.28808 + 0.82644*test.data$visual0 -0.21222*test.data$time -2.42200  -0.04959*test.data$time )

# 2) With random effects
predict_re = predict(fm16.1mer, newdata = test.data)
predict_re
```

```
##        1
## 52.45544
```

```
# (9.28808 + 0.82644*test.data$visual0 -0.21222*test.data$time -2.42200  -0.04959*test.data$time -3.33466872 )

# where -3.33466872 comes from the random intercept vector and corresponds to the subject 234
re = ranef(fm16.1mer)[[1]]
re[row.names(re)==test.data$subject,]
```

```
## [1] -3.334669
```

```
# Prediction from mixed model on a test observation from a subject not present in the training set:
test.data= data.frame(subject= '400', treat.f='Active', visual0= 63, time = 12)

# 1) Without random effects ->  re.form=NA
predict_no_re = predict(fm16.1mer, newdata = test.data, re.form=NA)
predict_no_re # the same as before
```

```
##        1
## 55.79011
```

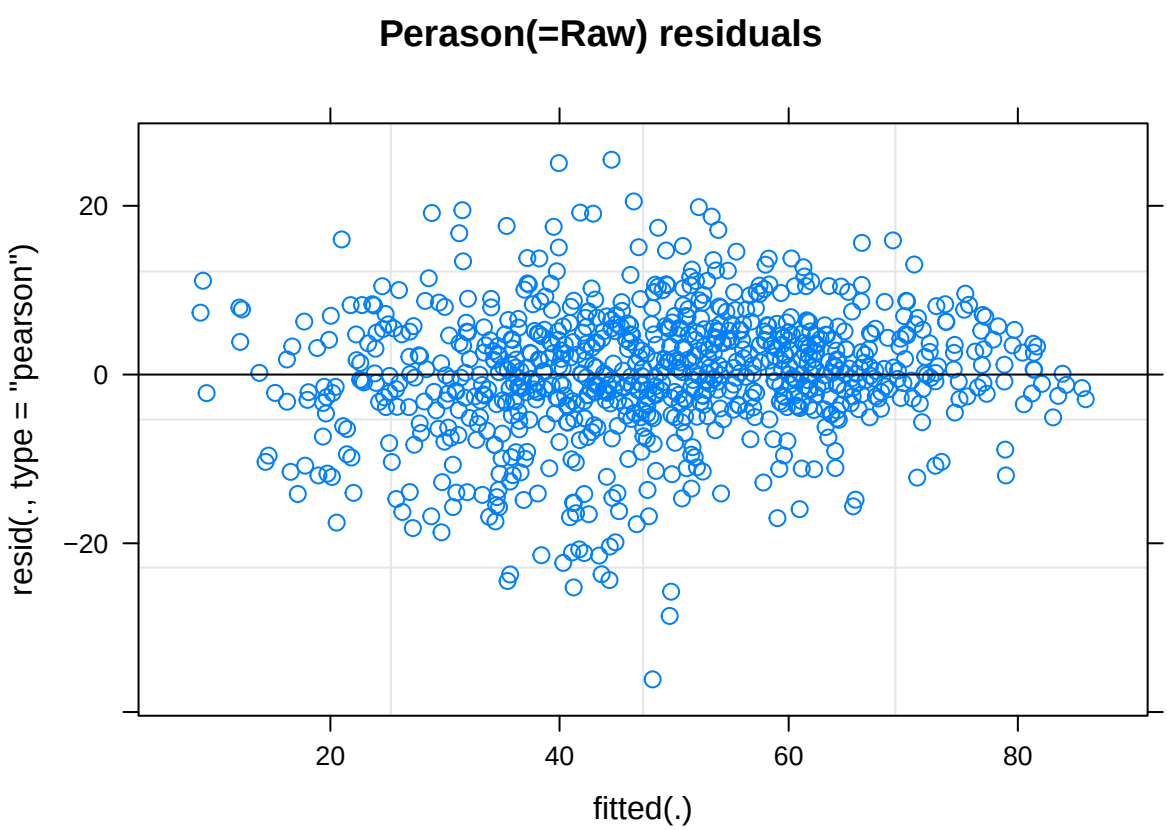
```
# 2) With random effects
# predict_re = predict(fm16.1mer, newdata = test.data)
# it does not recognize the subject --> allow.new.levels = T

#! error as R does not know his random effect value, being that data not into the training set
predict_re = predict(fm16.1mer, newdata=test.data, allow.new.levels = T)
predict_re # the same as before, it uses the average of the random intercept, ie 0
```

```
##        1
## 55.79011
```

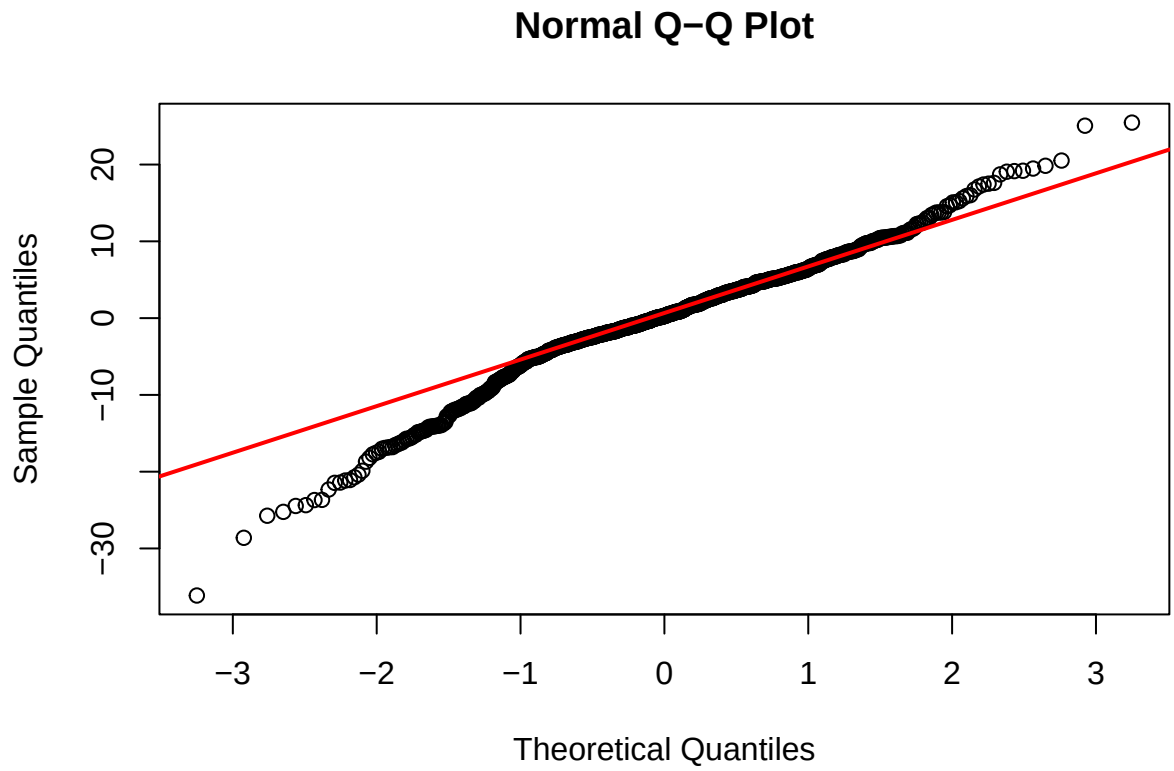
```
#! There is no standard way to do prediction with random effects, we can not consider them,
#! or set them to the mean, or estimate a value considering the distribution b_i~N(0,simga^2)
```

```
# Diagnostic plots
#--------------------
# 1) Assessing Assumption on the within-group errors

plot(fm16.1mer, main="Perason(=Raw) residuals")  ## Pearson and raw residuals are the same now
```

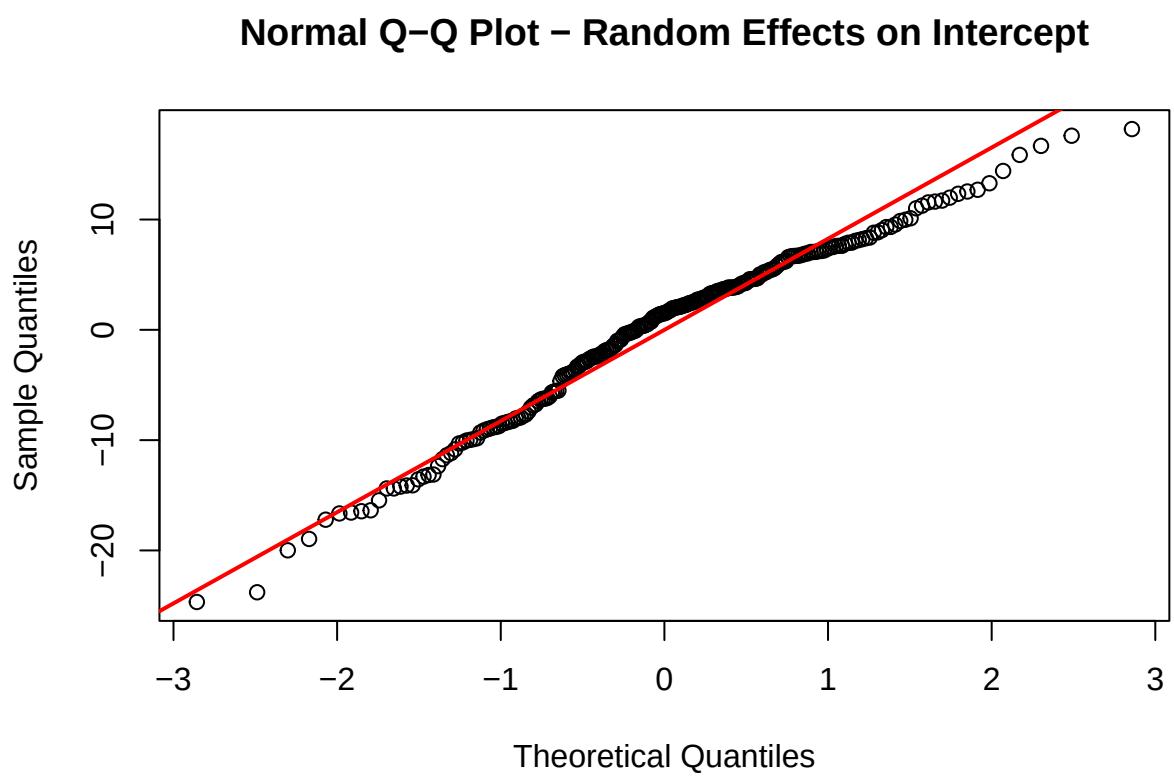
11.2.0.0.3 Diagnostic plots




```
qqnorm(resid(fm16.1mer))
qqline(resid(fm16.1mer), col='red', lwd=2)
```




```r
# 2) Assessing Assumption on the Random Effects

qqnorm(unlist(ranef(fm16.1mer)$subject),
       main='Normal Q-Q Plot - Random Effects on Intercept')
qqline(unlist(ranef(fm16.1mer)$subject), col='red', lwd=2)
```



11.2.1 2. Linear Models with random intercept + slope (q=1)

```
################################################################################
## Model 2: random intercept + slope and homoscedastic residuals
## Model 2.1: general D
#! VAM_it = beta0 + beta1*VAMTime0_it + beta1*time_it + beta3*Treat_it + beta4*time_it*Treat_it
#!          + b0_i + b1_i*time_it + eps_it
#! b_i = ( b0_i, b1_i )' ~ N( (0,0), [sigma^2_b0  cov        ]  )
#!                                   [ cov        sigma^2_b1 ]
#!
#! When we add a random effect wrt a variable, that variable must also appear in the classical
#! form of the linear model (if i understand it nicely)

fm16.2mer = lmer(visual ~ visual0 + time * treat.f + (1+time|subject),
                 data = armd, control=lmerControl(optimizer="bobyqa",
                                                  optCtrl=list(maxfun=2e5)))

#! Having now increased the number of parameters we should take more care about our data:
#! we need lots of them, with groups possibly balanced, ecc

summary(fm16.2mer)
```

11.2.1.1 2.1 general structure of D

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: visual ~ visual0 + time * treat.f + (1 + time | subject)
##    Data: armd
## Control: lmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 2e+05))
##
## REML criterion at convergence: 6435.8
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9304 -0.3400  0.0419  0.4619  2.9862
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  subject  (Intercept) 49.93696 7.067
##           time         0.07453 0.273    0.14
##  Residual             45.48494 6.744
## Number of obs: 867, groups:  subject, 234
##
## Fixed effects:
##                    Estimate Std. Error t value
## (Intercept)         4.76853    2.31955   2.056
## visual0             0.90890    0.03919  23.193
## time               -0.21529    0.03166  -6.799
## treat.fActive      -2.29010    1.18354  -1.935
## time:treat.fActive -0.05637    0.04622  -1.220
##
## Correlation of Fixed Effects:
##             (Intr) visul0 time   trt.fA
## visual0     -0.934
## time        -0.073  0.003
## treat.fActv -0.274  0.026  0.138
## tm:trt.fAct  0.050 -0.002 -0.685 -0.207
```

```
confint(fm16.2mer,oldNames=TRUE)
```

```
## Computing profile confidence intervals ...
```

```
##                          2.5 %      97.5 %
## .sig01              5.90348435  8.14119055
## .sig02             -0.09142188  0.41816520
## .sig03              0.22917085  0.31639258
## .sigma              6.30520479  7.24640136
## (Intercept)         0.10993051  9.44284079
## visual0             0.82966680  0.98794928
## time               -0.27736361 -0.15324335
## treat.fActive      -4.60672722  0.02361798
## time:treat.fActive -0.14707000  0.03414851
```

```r
#! the sig02 CI contains zero, so we can maybe rerun the model trying with a diagonal matrix,
#! as it does not seem that we have non-zero values as covariances

vcovb = vcov(fm16.2mer)
vcovb
```

```
## 5 x 5 Matrix of class "dpoMatrix"
##                     (Intercept)       visual0          time treat.fActive
## (Intercept)         5.380317103 -8.491453e-02 -5.362932e-03  -0.751801288
## visual0            -0.084914531  1.535776e-03  3.840435e-06   0.001202526
## time               -0.005362932  3.840435e-06  1.002525e-03   0.005153598
## treat.fActive      -0.751801288  1.202526e-03  5.153598e-03   1.400771957
## time:treat.fActive  0.005326322 -3.178301e-06 -1.002524e-03  -0.011341847
##                    time:treat.fActive
## (Intercept)              5.326322e-03
## visual0                 -3.178301e-06
## time                    -1.002524e-03
## treat.fActive           -1.134185e-02
## time:treat.fActive       2.135873e-03
```

```r
corb = cov2cor(vcovb)
nms = abbreviate(names(fixef(fm16.2mer)), 5)
rownames(corb) = nms
corb
```

```
## 5 x 5 Matrix of class "dpoMatrix"
##                    (Intercept)      visual0         time treat.fActive
## (Intercept)         1.00000000 -0.934144528 -0.073021494   -0.27385168
## visual0            -0.93414453  1.000000000  0.003095058    0.02592666
## time               -0.07302149  0.003095058  1.000000000    0.13752421
## treat.fActive      -0.27385168  0.025926661  0.137524213    1.00000000
## time:treat.fActive  0.04968621 -0.001754863 -0.685108399   -0.20735399
##                    time:treat.fActive
## (Intercept)               0.049686208
## visual0                  -0.001754863
## time                     -0.685108399
## treat.fActive            -0.207353989
## time:treat.fActive        1.000000000
```

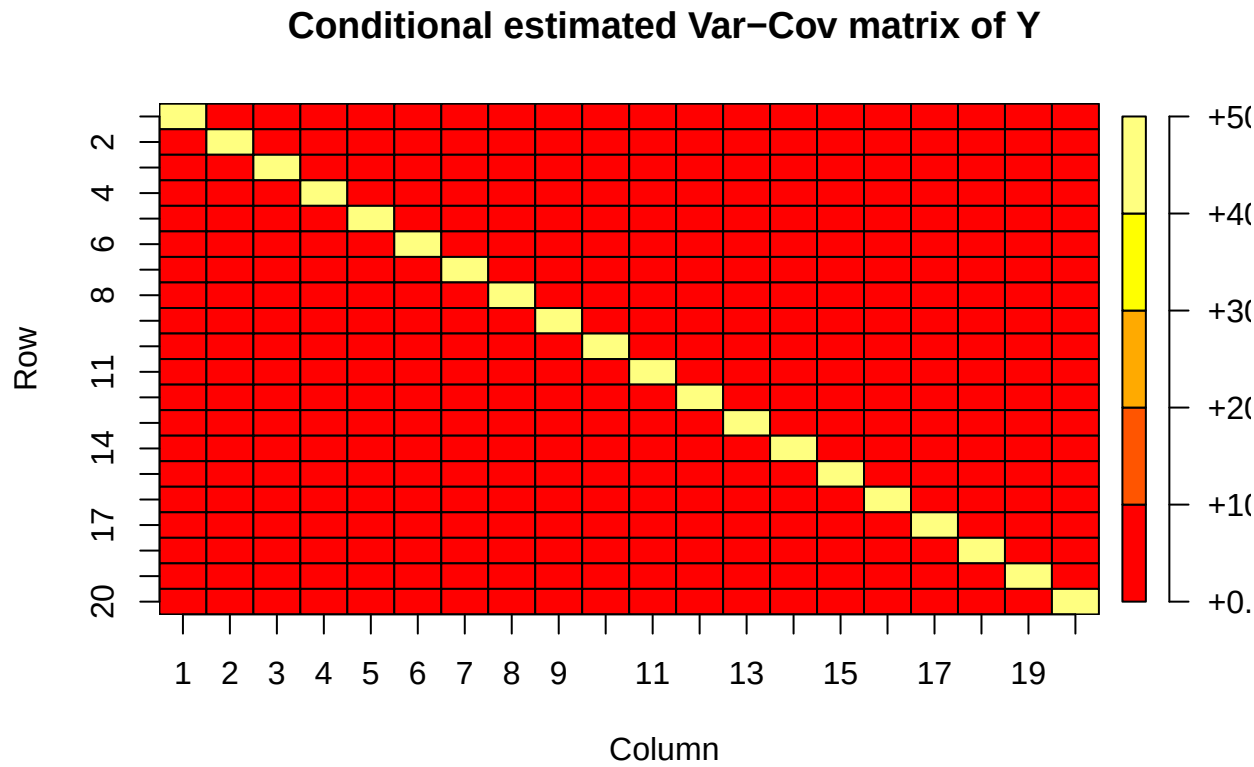
```r
## Var-Cov matrix of random-effects and errors
vc = VarCorr(fm16.2mer)
print(vc, comp = c("Variance", "Std.Dev."))
```

```
##  Groups   Name        Variance  Std.Dev. Corr
##  subject  (Intercept) 49.936960 7.0666
##           time         0.074528 0.2730   0.143
##  Residual             45.484944 6.7443
```

```r
## Let's compute the conditional and marginal var-cov matrix of Y
sgma = summary(fm16.2mer)$sigma

A = getME(fm16.2mer, "A") # A : N*2 x n
I.n = Diagonal(ncol(A)) # IN: n x n

## the conditional variance-covariance matrix of Y (diagonal matrix)
SigmaErr = sgma^2 * (I.n)
SigmaErr[3:6, 3:6]  ## visualization of individual 2
```

```
## 4 x 4 diagonal matrix of class "ddiMatrix"
##      [,1]     [,2]     [,3]     [,4]
## [1,] 45.48494        .        .        .
## [2,]        . 45.48494        .        .
## [3,]        .        . 45.48494        .
## [4,]        .        .        . 45.48494
```

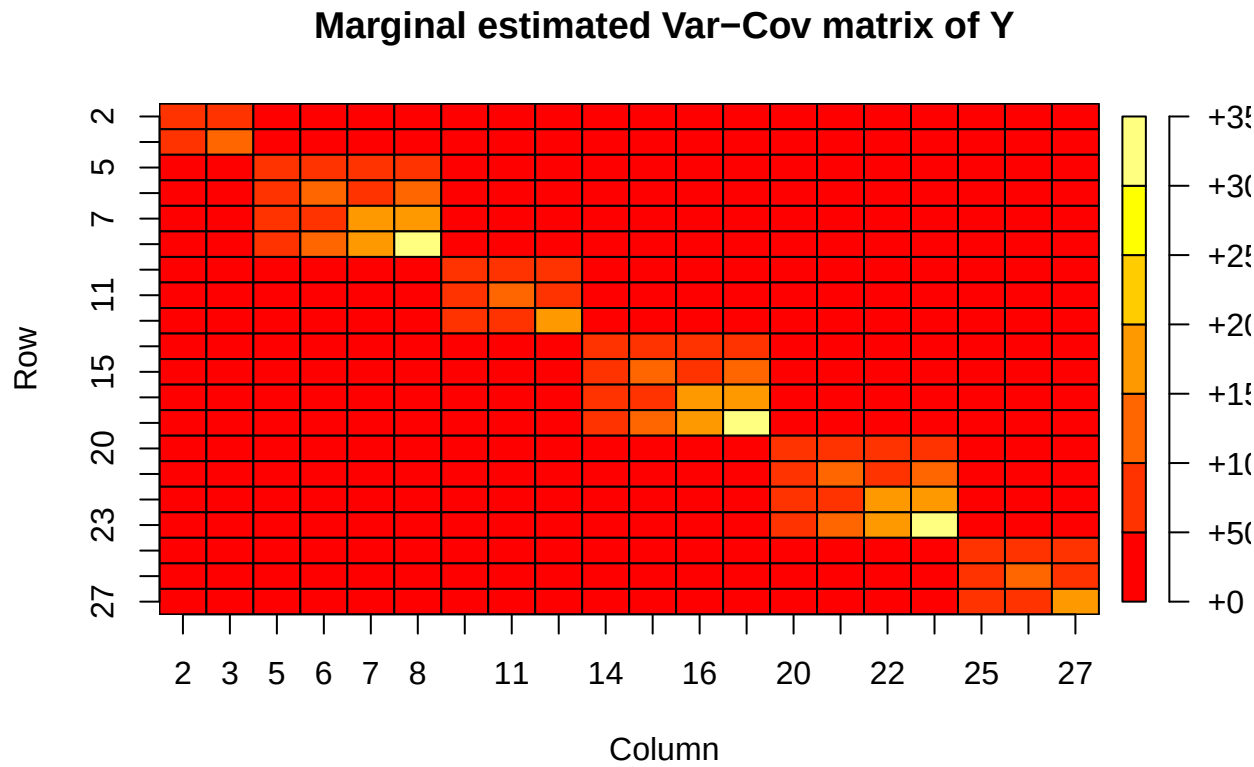
```r
# Conditioned to the random effects b_i, we observe the var-cov of the errors
# that are independent and homoscedastic

plot(as.matrix(SigmaErr[1:20,1:20]), main = 'Conditional estimated Var-Cov matrix of Y')
```




```r
## the marginal variance-covariance matrix of Y (block-diagonal matrix)
V = sgma^2 * (I.n + crossprod(A)) # V = s^2*(I_N+A*A)
V[3:6, 3:6]  #-> V is a block-diagional matrix, the marginal var-cov matrix
```

```
## 4 x 4 sparse Matrix of class "dsCMatrix"
##          5         6         7         8
## 5 98.82437  57.93435  64.82674  80.90896
## 6 57.93435 112.78403  81.34619 114.12275
## 7 64.82674  81.34619 151.61030 163.94343
## 8 80.90896 114.12275 163.94343 325.67663
```

```r
# visualization of the first 20 rows/columns
plot(as.matrix(V[1:20,1:20]), main = 'Marginal estimated Var-Cov matrix of Y')
```




```
# PVRE
#--------------------
#! how the formula update: when we have just the intercept:
#! PVRE = sgma^2_b / (sgma^2_b + sgma^2_eps)
#
#! when we have more random effects:
#! sgma^2_bvec = var( b0_i + b1_i*time_it ) =
#!             = sgma^2_b0 + sgma^2_b1*time_it^2 + 2*time_it*cov(b0,b1)
#! But having there time^2 maybe we can get high PVRE just because we have high values of time
#! and the formula is still the same with that updated sgma^2_b as the above one
#! PVRE = sgma^2_b / (sgma^2_b + sgma^2_eps)
#
# In this case the variance of random sigma2_R effects represents the mean random
# effect variance of the model and is given by
# sigma2_b = Var(b0,b1) = sigma2_b0 + 2Cov(b0,b1)*mean(w) + sigma2_b1*mean(w^2)
# See equation (10) in Johnson (2014), Methods in Ecology and Evolution, 5(9), 944-946.
sigma2_eps = as.numeric(get_variance_residual(fm16.2mer))
sigma2_eps
```

11.2.1.1.1 PVRE

```
## [1] 45.48494
```

```
sigma2_b = as.numeric(get_variance_random(fm16.2mer))
# 49.933917 + 0.074552*mean(armd$time^2) +2*0.143*7.06639*0.27304*mean(armd$time)
sigma2_b
```

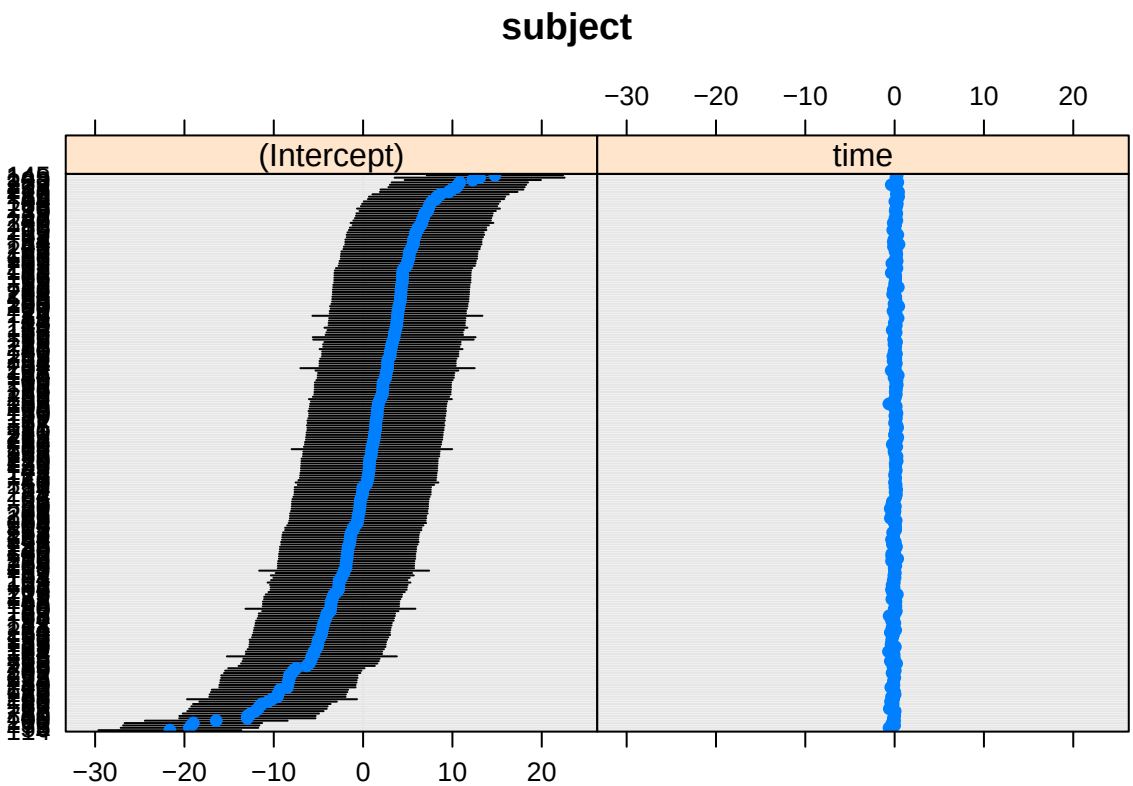
```
## [1] 121.0456
```

```
PVRE = sigma2_b/(sigma2_b+sigma2_eps)
PVRE # 72% is very high!
```

```
## [1] 0.7268672
```

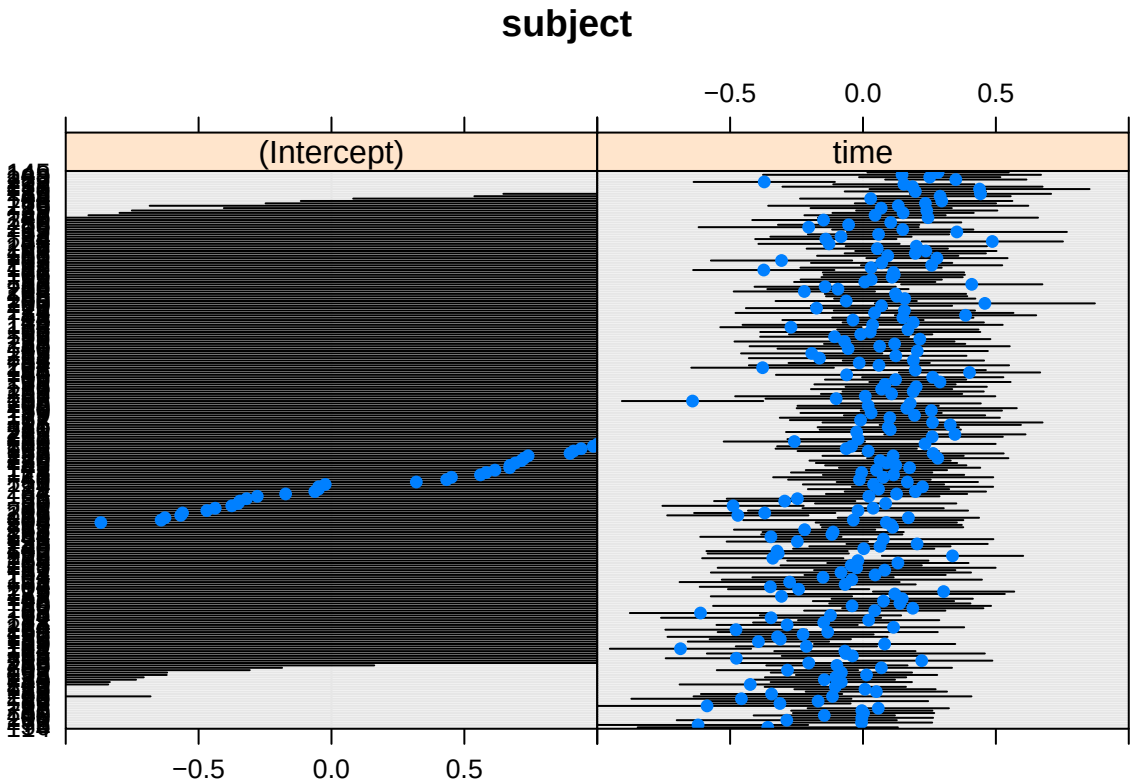
```r
## visualization of the random intercepts with their 95% confidence intervals
# Random effects: b_0i, b_1i for i=1,...,234
dotplot(ranef(fm16.2mer, condVar=T))
```

```
## $subject
```




```r
#! sgma^2 for b0 is high, we have variability as seen from the plot
#! sgma^2 for b1 is low, but we can zoom and see also there the variability
dotplot(ranef(fm16.2mer, condVar=T),xlim=c(-1,1))
```

```
## $subject
```




```
#! so also there we have variability
```

```
# Comparing models
#------------------
# The anova function, when given two or more arguments representing fitted models,
# produces likelihood ratio tests comparing the models.
anova(fm16.1mer, fm16.2mer)
```

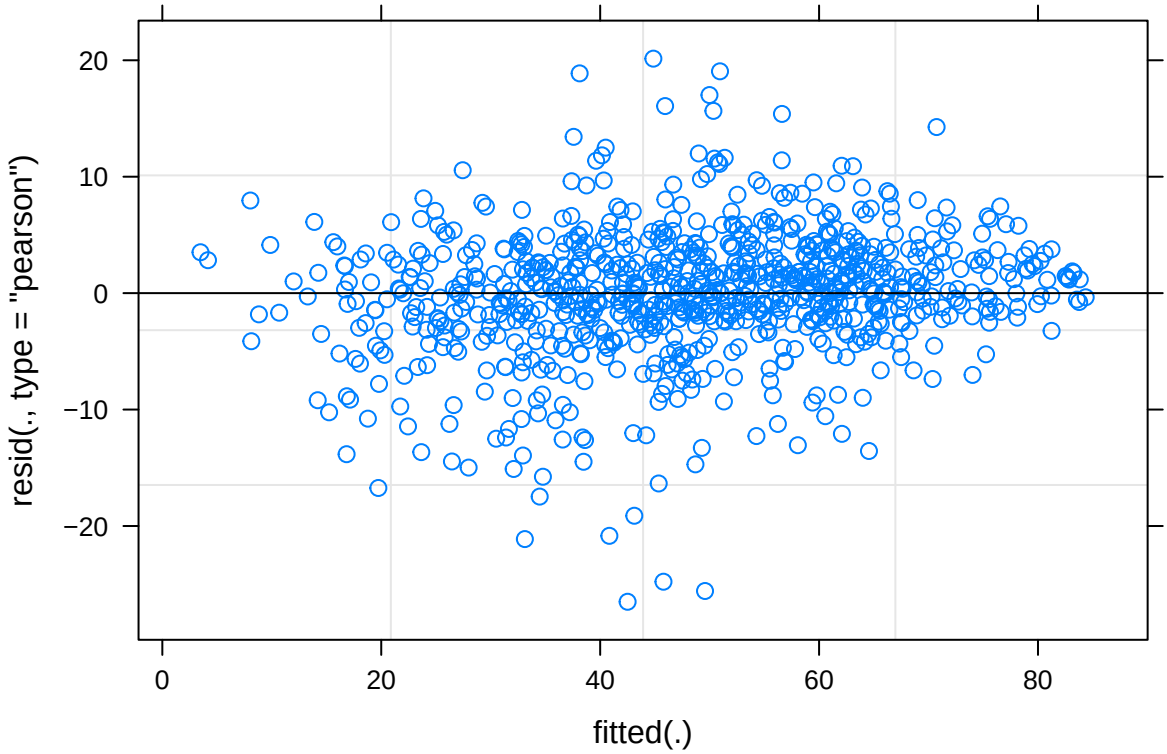
11.2.1.1.2 Comparing models

```
## refitting model(s) with ML (instead of REML)

## Data: armd
## Models:
## fm16.1mer: visual ~ visual0 + time * treat.f + (1 | subject)
## fm16.2mer: visual ~ visual0 + time * treat.f + (1 + time | subject)
##           npar    AIC    BIC  logLik deviance  Chisq Df Pr(>Chisq)
## fm16.1mer    7 6579.7 6613.0 -3282.8   6565.7
## fm16.2mer    9 6442.0 6484.9 -3212.0   6424.0 141.66  2  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

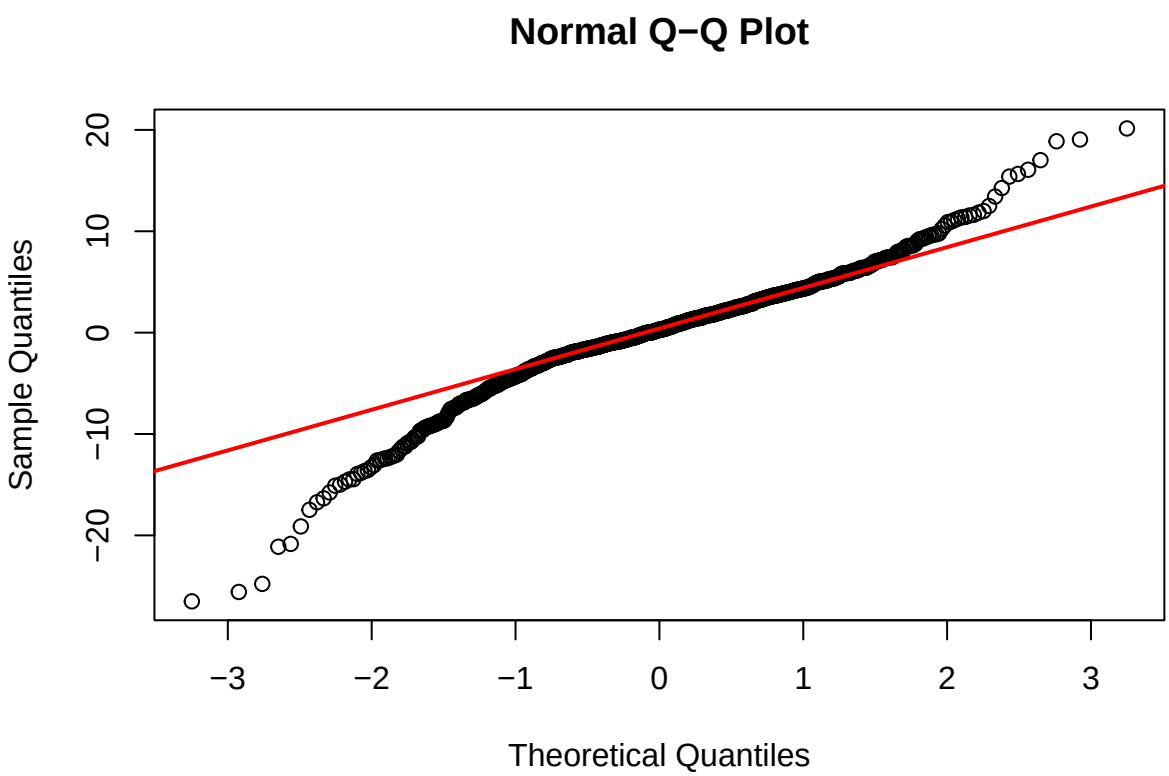
```
# The p-value for the test is essentially zero -> we prefer fm16.2mer
```

```
# Diagnostic plots
#--------------------
# 1) Assessing Assumption on the within-group errors

plot(fm16.2mer)
```

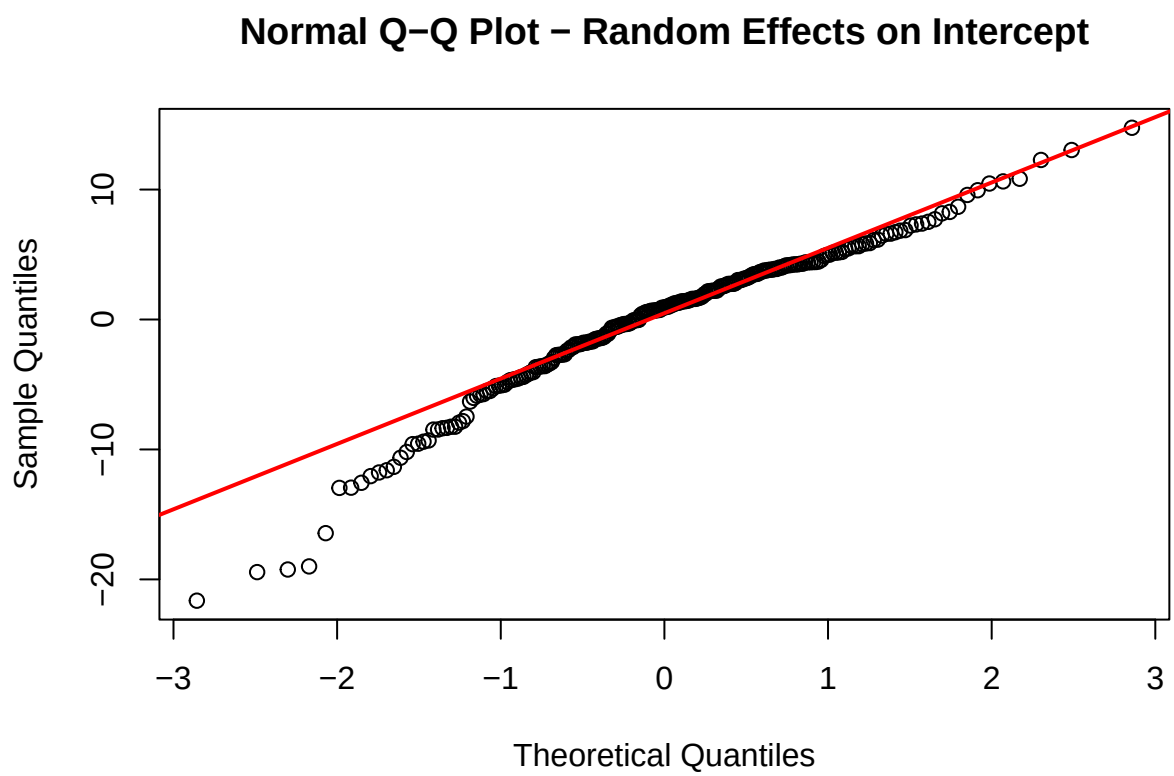
11.2.1.1.3 Diagnostic plots



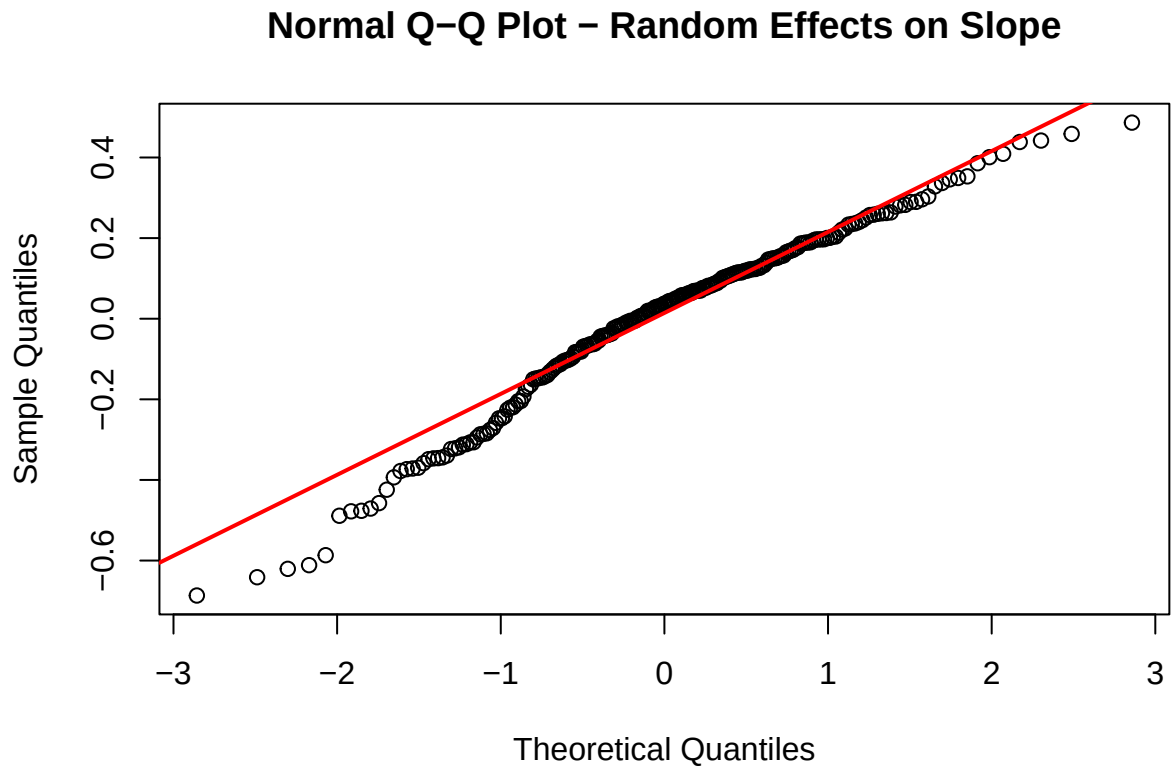

```
qqnorm(resid(fm16.2mer))
qqline(resid(fm16.2mer), col='red', lwd=2)
```



```r
# 2) Assessing Assumption on the Random Effects

qqnorm(unlist(ranef(fm16.2mer)$subject[,1]),
       main='Normal Q-Q Plot - Random Effects on Intercept')
qqline(unlist(ranef(fm16.2mer)$subject[,1]), col='red', lwd=2)
```




```r
qqnorm(unlist(ranef(fm16.2mer)$subject[,2]),
       main='Normal Q-Q Plot - Random Effects on Slope')
qqline(unlist(ranef(fm16.2mer)$subject[,2]), col='red', lwd=2)
```




```
## We observe that the correlation between d_11 and d_22 id very low,
## we fit a new model with a diagonal D matrix
```

```
################################################################################
## Model 2.2: diagonal D

fm16.2dmer = lmer(visual ~ visual0 + time * treat.f + (1|subject) + (0 + time|subject),
                  data = armd, control=lmerControl(optimizer="bobyqa",
                                                   optCtrl=list(maxfun=2e5)))

summary(fm16.2dmer)
```

11.2.1.2 2.2 diagonal D

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: visual ~ visual0 + time * treat.f + (1 | subject) + (0 + time |
##     subject)
##    Data: armd
## Control: lmerControl(optimizer = "bobyqa", optCtrl = list(maxfun = 2e+05))
##
## REML criterion at convergence: 6437.1
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9719 -0.3293  0.0403  0.4456  2.8335
##
## Random effects:
##  Groups    Name        Variance Std.Dev.
##  subject   (Intercept) 54.07117 7.3533
##  subject.1 time         0.07936 0.2817
##  Residual              44.66736 6.6834
## Number of obs: 867, groups:  subject, 234
##
## Fixed effects:
##                    Estimate Std. Error t value
## (Intercept)         5.34875    2.33258   2.293
```

```
## visual0             0.89847    0.03932  22.852
## time               -0.21537    0.03227  -6.674
## treat.fActive      -2.31374    1.20976  -1.913
## time:treat.fActive -0.05506    0.04709  -1.169
##
## Correlation of Fixed Effects:
##             (Intr) visul0 time   trt.fA
## visual0     -0.932
## time        -0.101  0.003
## treat.fActv -0.277  0.025  0.190
## tm:trt.fAct  0.069 -0.001 -0.685 -0.280
```

```
confint(fm16.2dmer,oldNames=TRUE)
```

```
## Computing profile confidence intervals ...
```

```
##                        2.5 %      97.5 %
## .sig01              6.3350931  8.33564230
## .sig02              0.2412765  0.32319814
## .sigma              6.2608338  7.14755781
## (Intercept)         0.7841963  9.95603406
## visual0             0.8207091  0.97545265
## time               -0.2786479 -0.15212516
## treat.fActive      -4.6812429  0.05186764
## time:treat.fActive -0.1474854  0.03716238
```

```
vcovb = vcov(fm16.2dmer)
vcovb
```

```
## 5 x 5 Matrix of class "dpoMatrix"
##                     (Intercept)       visual0          time treat.fActive
## (Intercept)         5.440928630 -8.546107e-02 -7.584297e-03  -0.782726258
## visual0            -0.085461066  1.545820e-03  3.223481e-06   0.001203490
## time               -0.007584297  3.223481e-06  1.041221e-03   0.007408596
## treat.fActive      -0.782726258  1.203490e-03  7.408596e-03   1.463507226
## time:treat.fActive  0.007547355 -2.555282e-06 -1.041219e-03  -0.015944229
##                    time:treat.fActive
## (Intercept)              7.547355e-03
## visual0                 -2.555282e-06
## time                    -1.041219e-03
## treat.fActive           -1.594423e-02
## time:treat.fActive       2.217771e-03
```

```
corb = cov2cor(vcovb)
nms = abbreviate(names(fixef(fm16.2dmer)), 5)
rownames(corb) = nms
corb
```

```
## 5 x 5 Matrix of class "dpoMatrix"
##                    (Intercept)      visual0         time treat.fActive
## (Intercept)         1.00000000 -0.931863571 -0.100764471   -0.27738052
## visual0            -0.93186357  1.000000000  0.002540822    0.02530263
## time               -0.10076447  0.002540822  1.000000000    0.18978727
## treat.fActive      -0.27738052  0.025302628  0.189787267    1.00000000
## time:treat.fActive  0.06870683 -0.001380069 -0.685192196   -0.27986442
##                    time:treat.fActive
## (Intercept)               0.068706827
## visual0                  -0.001380069
## time                     -0.685192196
## treat.fActive            -0.279864420
## time:treat.fActive        1.000000000
```

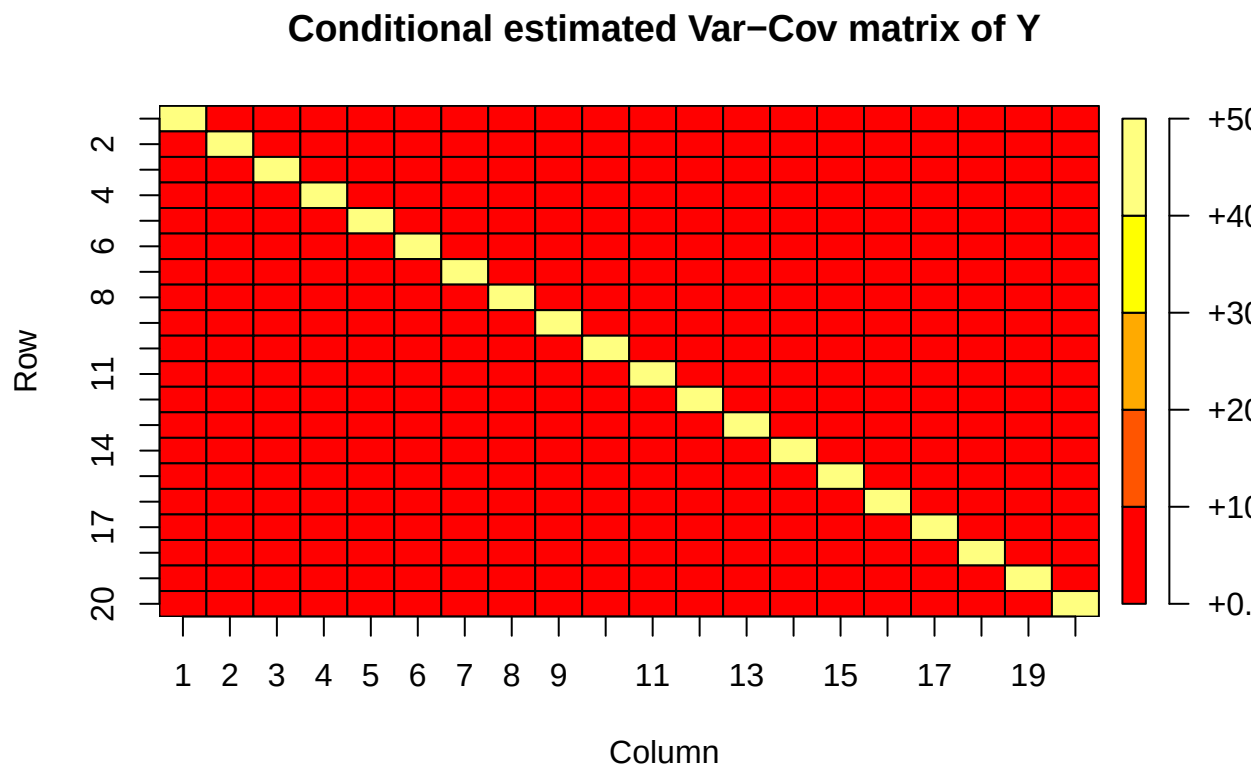
```
## Var-Cov matrix of random-effects and errors
vc = VarCorr(fm16.2dmer)
print(vc, comp = c("Variance", "Std.Dev."))
```

```
##  Groups    Name        Variance  Std.Dev.
##  subject   (Intercept) 54.071170 7.35331
##  subject.1 time         0.079359 0.28171
##  Residual              44.667357 6.68336
```

```r
## Let's compute the conditional and marginal var-cov matrix of Y
sgma = summary(fm16.2dmer)$sigma

A = getME(fm16.2dmer, "A") # A
I.n = Diagonal(ncol(A)) # IN

## the conditional variance-covariance matrix of Y (diagonal matrix)
SigmaErr = sgma^2 * (I.n)
SigmaErr[3:6, 3:6]  ## visualization of individual 2
```

```
## 4 x 4 diagonal matrix of class "ddiMatrix"
##       [,1]     [,2]     [,3]     [,4]
## [1,] 44.66736        .        .        .
## [2,]        . 44.66736        .        .
## [3,]        .        . 44.66736        .
## [4,]        .        .        . 44.66736
```

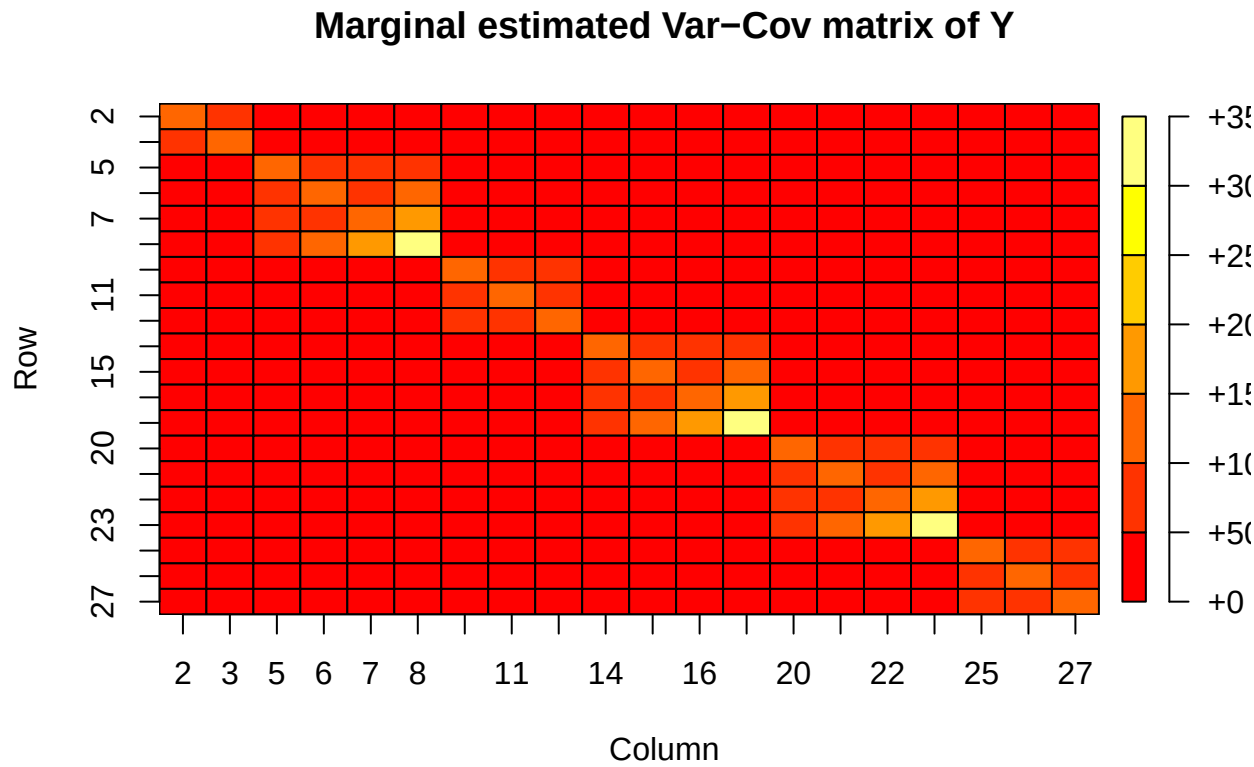
```r
# Conditioned to the random effects b_i, we observe the var-cov of the errors
# that are independent and homoscedastic

plot(as.matrix(SigmaErr[1:20,1:20]), main = 'Conditional estimated Var-Cov matrix of Y')
```




```r
## the marginal variance-covariance matrix of Y (block-diagonal matrix)
V = sgma^2 * (I.n + crossprod(A)) # V = s^2*(I_N+A*A)
V[3:6, 3:6]  #-> V is a block-diagional matrix, the marginal var-cov matrix
```

```
## 4 x 4 sparse Matrix of class "dsCMatrix"
##           5         6         7         8
## 5 100.00827  57.88040  61.68964  70.57785
## 6  57.88040 110.16623  76.92657 103.59121
## 7  61.68964  76.92657 144.44933 153.11125
## 8  70.57785 103.59121 153.11125 313.32537
```

```r
# visualization of the first 20 rows/columns
plot(as.matrix(V[1:20,1:20]), main = 'Marginal estimated Var-Cov matrix of Y')
```




```
# PVRE
#--------------------
# In this case the variance of random sigma2_R effects represents the mean random
# effect variance of the model and is given by
# sigma2_b = Var(b0,b1) = sigma2_b0 + 0 + sigma2_b1*mean(z^2)
# See equation (10) in Johnson (2014), Methods in Ecology and Evolution, 5(9), 944-946.
sigma2_eps = as.numeric(get_variance_residual(fm16.2dmer))
sigma2_eps
```

11.2.1.2.1 PVRE

```
## [1] 44.66736
```

```
sigma2_b = as.numeric(get_variance_random(fm16.2dmer)) +
        mean(armd$time^2)* as.numeric(get_variance_slope(fm16.2dmer))
# 54.07117 + 0.07935904*mean(armd$time^2)
sigma2_b
```

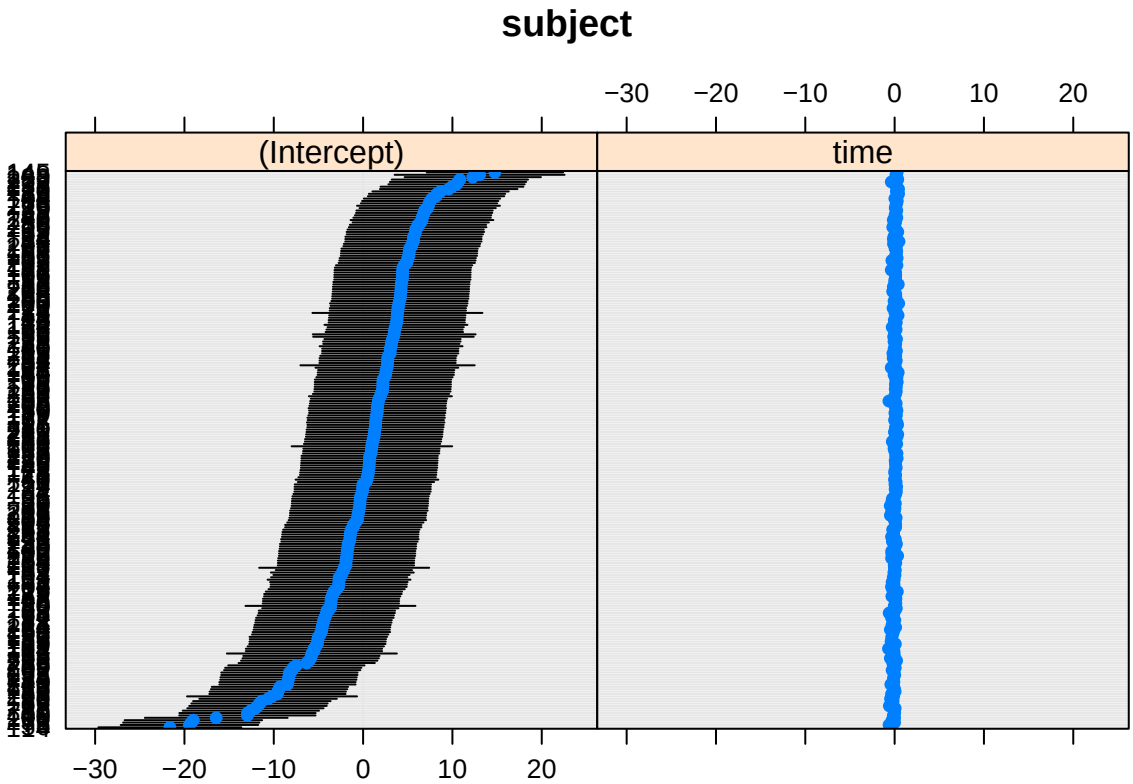
```
## [1] 116.9477
```

```
PVRE = sigma2_b/(sigma2_b+sigma2_eps)
PVRE # 72% is very high!
```

```
## [1] 0.7236188
```

```
## visualization of the random intercepts with their 95% confidence intervals
# Random effects: b_0i, b_1i for i=1,...,234
dotplot(ranef(fm16.2mer, condVar=T))
```

```
## $subject
```




```
# Comparing models
#-----------------
# The anova function, when given two or more arguments representing fitted models,
# produces likelihood ratio tests comparing the models.
anova(fm16.2mer, fm16.2dmer)
```

11.2.1.2.2 Comparing models

```
## refitting model(s) with ML (instead of REML)

## Data: armd
## Models:
## fm16.2dmer: visual ~ visual0 + time * treat.f + (1 | subject) + (0 + time | subject)
## fm16.2mer: visual ~ visual0 + time * treat.f + (1 + time | subject)
##            npar    AIC    BIC  logLik deviance  Chisq Df Pr(>Chisq)
## fm16.2dmer    8 6441.4 6479.5 -3212.7   6425.4
## fm16.2mer     9 6442.0 6484.9 -3212.0   6424.0 1.4087  1     0.2353
```

```
# The p-value for the test is essentially zero -> we prefer fm16.2dmer
```

11.2.2 With heteroschedastic rediduals

```
#! Till now we assumed eps_it ~ N(0,sgma^2_eps), so homoschedastic residuals
#! But we can also deal with...
#######################################################
### LINEAR MIXED MODELS WITH HETEROSCEDASTIC RESIDUALS ###
############ nlme package --> lme() function #############
#######################################################
#! Students nested into schools, patients nested into hospitals, this suits very
#! well LMMs


##############################################################################
## Model 1. Random intercept, heteroscedastic residuals (varPower of time)
```

```r
## fixed-effects formula
lm2.form = formula(visual ~ visual0 + time + treat.f + treat.f:time )

# LMM with homoscedastic residuals
fm16.1 = lme(lm2.form, random = ~1|subject, data = armd)

# update fm16.1 including heteroscedastic residuals
fm16.2 = update(fm16.1,
                weights = varPower(form = ~ time),
                data = armd)

summary(fm16.2)
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC    BIC    logLik
##   6537.126 6575.2 -3260.563
##
## Random effects:
##  Formula: ~1 | subject
##         (Intercept) Residual
## StdDev:    7.705553 3.606684
##
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
##  Parameter estimates:
##     power
## 0.3144089
## Fixed effects:  list(lm2.form)
##                        Value Std.Error  DF   t-value p-value
## (Intercept)         7.066881 2.3029227 631  3.068657  0.0022
## visual0             0.866544 0.0385591 231 22.473125  0.0000
## time               -0.212627 0.0262274 631 -8.107072  0.0000
## treat.fActive      -2.305034 1.2435005 231 -1.853665  0.0651
## time:treat.fActive -0.050888 0.0383676 631 -1.326323  0.1852
##  Correlation:
##                    (Intr) visul0 time   trt.fA
## visual0            -0.925
## time               -0.152 -0.001
## treat.fActive      -0.289  0.025  0.282
## time:treat.fActive  0.104  0.000 -0.684 -0.402
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.80887563 -0.37398177  0.08525419  0.51699910  3.08883820
##
## Number of Observations: 867
## Number of Groups: 234
```

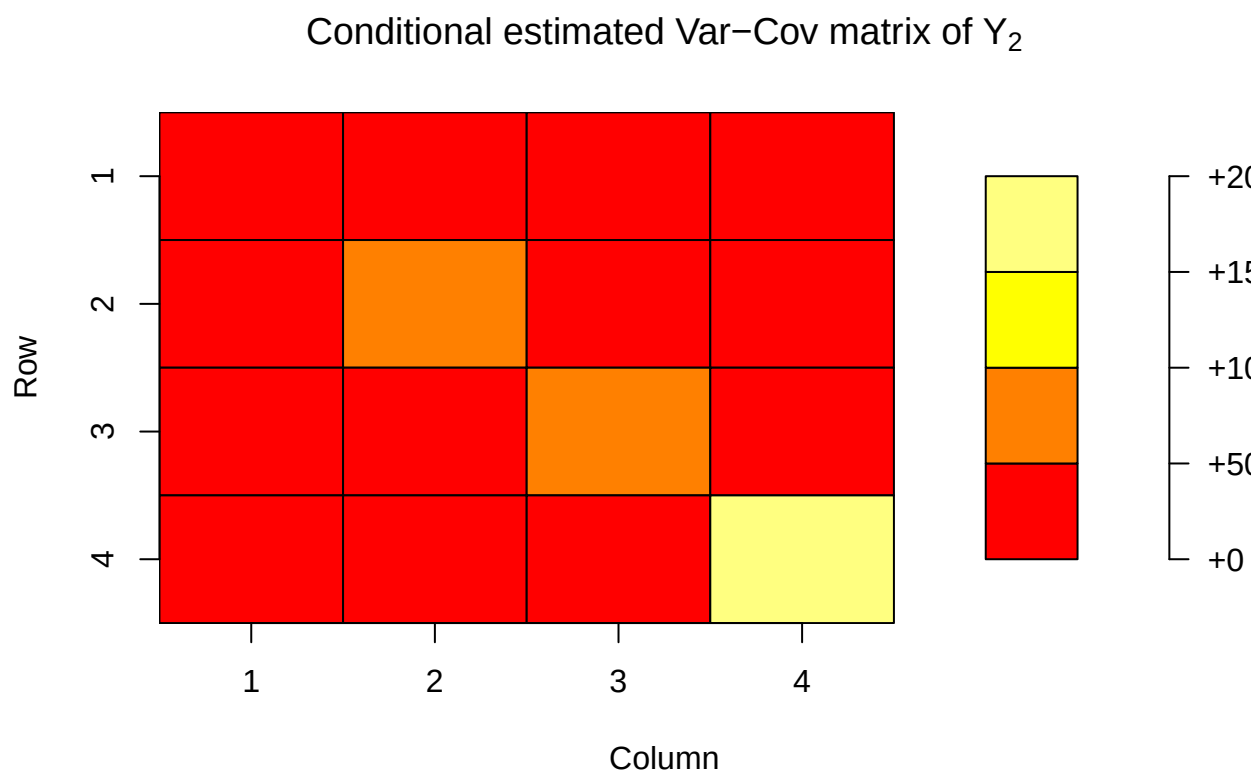
```r
VarCorr(fm16.2)
```

```
## subject = pdLogChol(1)
##             Variance StdDev
## (Intercept) 59.37555 7.705553
## Residual    13.00817 3.606684
```

```r
## var-cov matrix of the errors (ie of Y, conditional to the random effects),
## that are independent but heteroscedastic
fm16.2ccov = getVarCov(fm16.2, type = "conditional",  individual = "2")
fm16.2ccov
```

```
## subject 2
## Conditional variance covariance matrix
##        1      2      3      4
## 1 31.103  0.000  0.000   0.00
## 2  0.000 62.062  0.000   0.00
## 3  0.000  0.000 95.966   0.00
## 4  0.000  0.000  0.000 156.05
##   Standard Deviations: 5.577 7.8779 9.7962 12.492
```

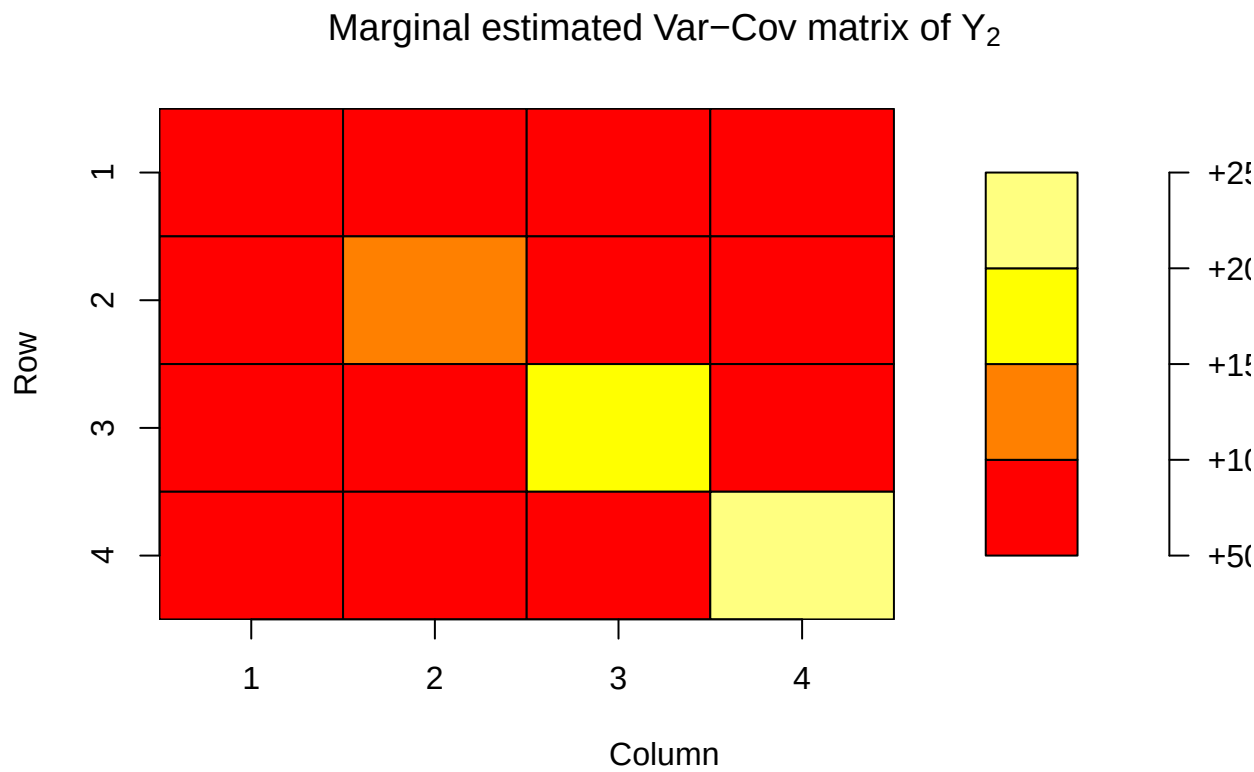
```r
plot(as.matrix(fm16.2ccov[[1]]),
     main = expression(paste('Conditional estimated Var-Cov matrix of ', Y[2])))
```




```r
## var-cov matrix of Y_i
fm16.2cov = getVarCov(fm16.2, type = "marginal", individual = "2")
fm16.2cov # (90.479 = 31.103 + 59.37555; 121.440 = 62.062 + 59.37555; ...)
```

```
## subject 2
## Marginal variance covariance matrix
##        1       2       3       4
## 1 90.479  59.376  59.376  59.376
## 2 59.376 121.440  59.376  59.376
## 3 59.376  59.376 155.340  59.376
## 4 59.376  59.376  59.376 215.430
##   Standard Deviations: 9.512 11.02 12.464 14.677
```

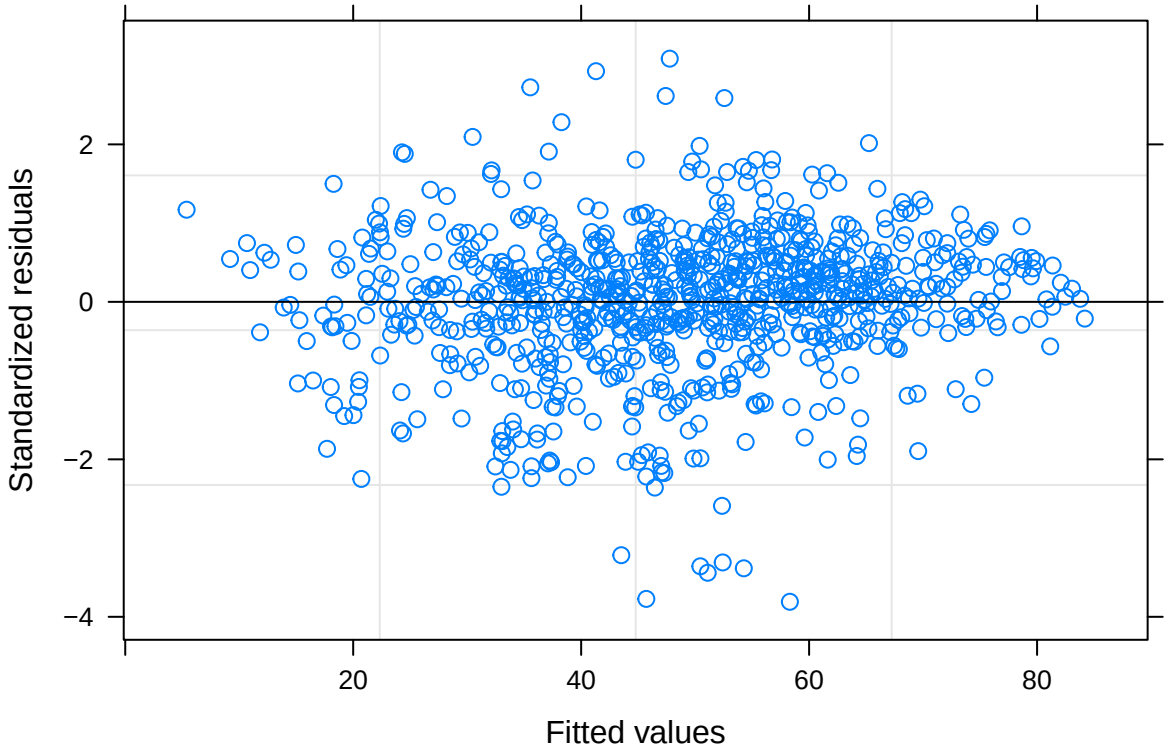
```r
plot(as.matrix(fm16.2cov[[1]]),
     main = expression(paste('Marginal estimated Var-Cov matrix of ', Y[2])))
```



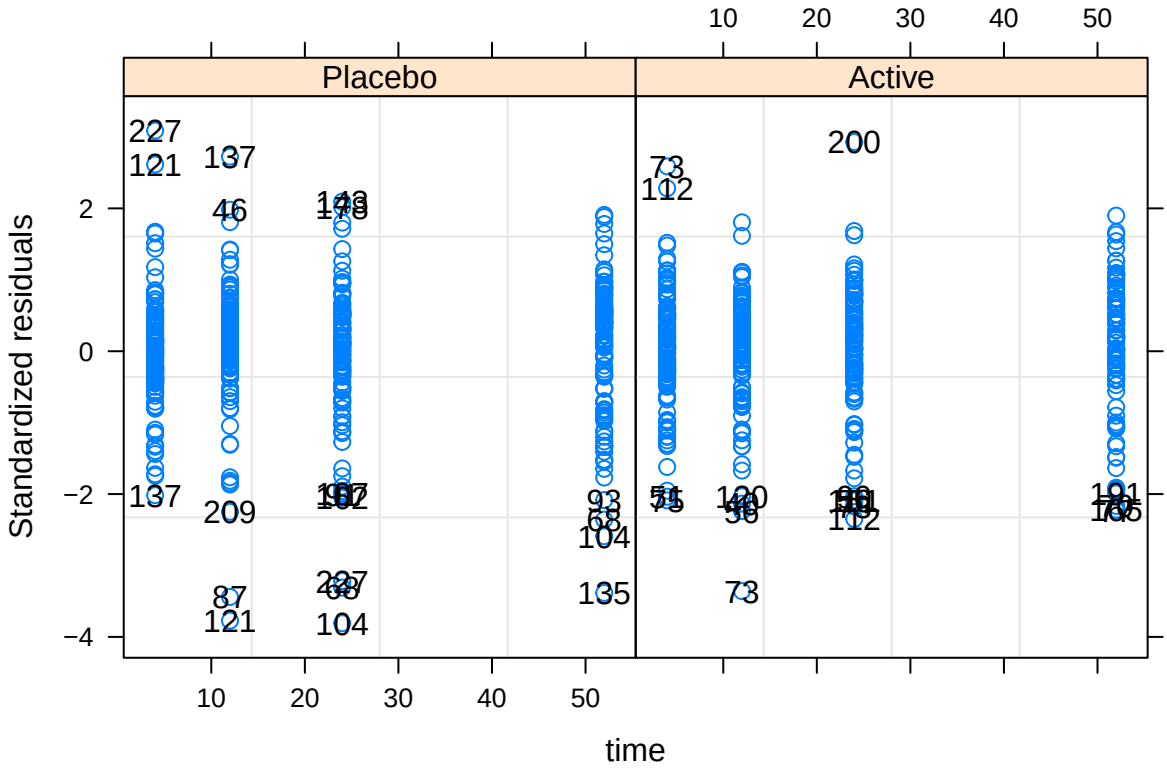

```
# var-cov matrix of y_i is the same for each subject i,
# except for the number of observations, ranging from 1 to 4

## correlation matrix of Y_i
cov2cor(fm16.2cov[[1]])
```

```
##           1         2         3         4
## 1 1.0000000 0.5664462 0.5008307 0.4252890
## 2 0.5664462 1.0000000 0.4323024 0.3670969
## 3 0.5008307 0.4323024 1.0000000 0.3245735
## 4 0.4252890 0.3670969 0.3245735 1.0000000
```

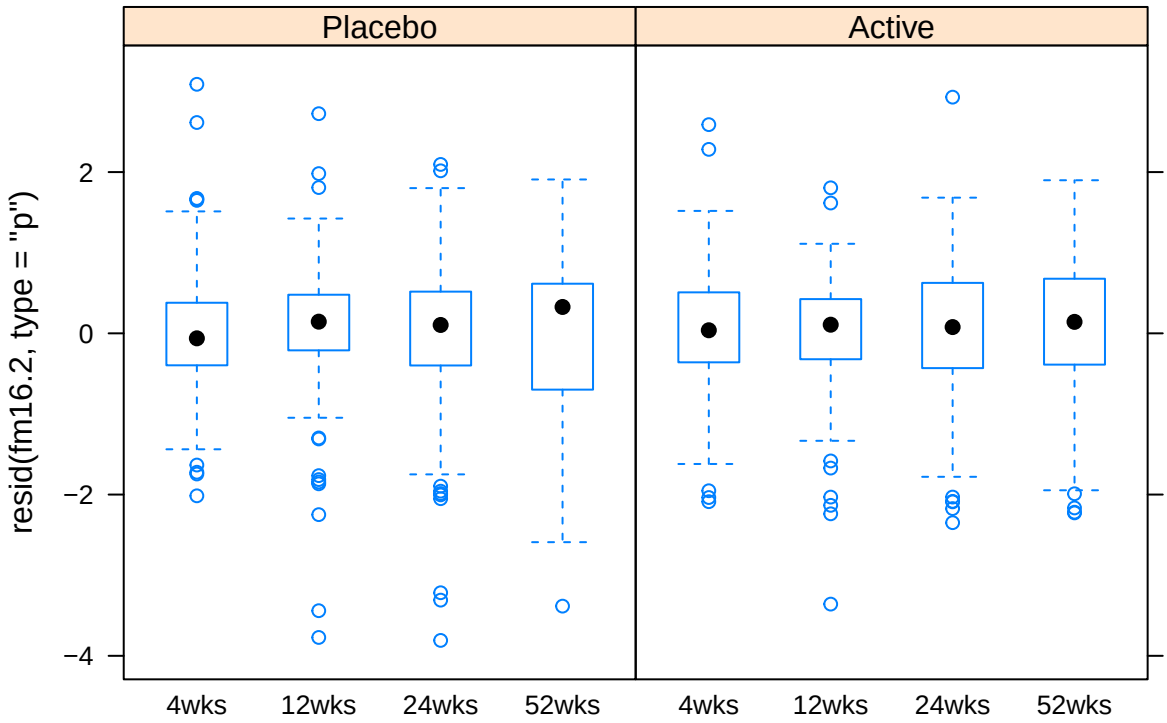
```
## ANALYSIS OF RESIDUALS
# Default residual plot of conditional Pearson residuals
plot(fm16.2)
```



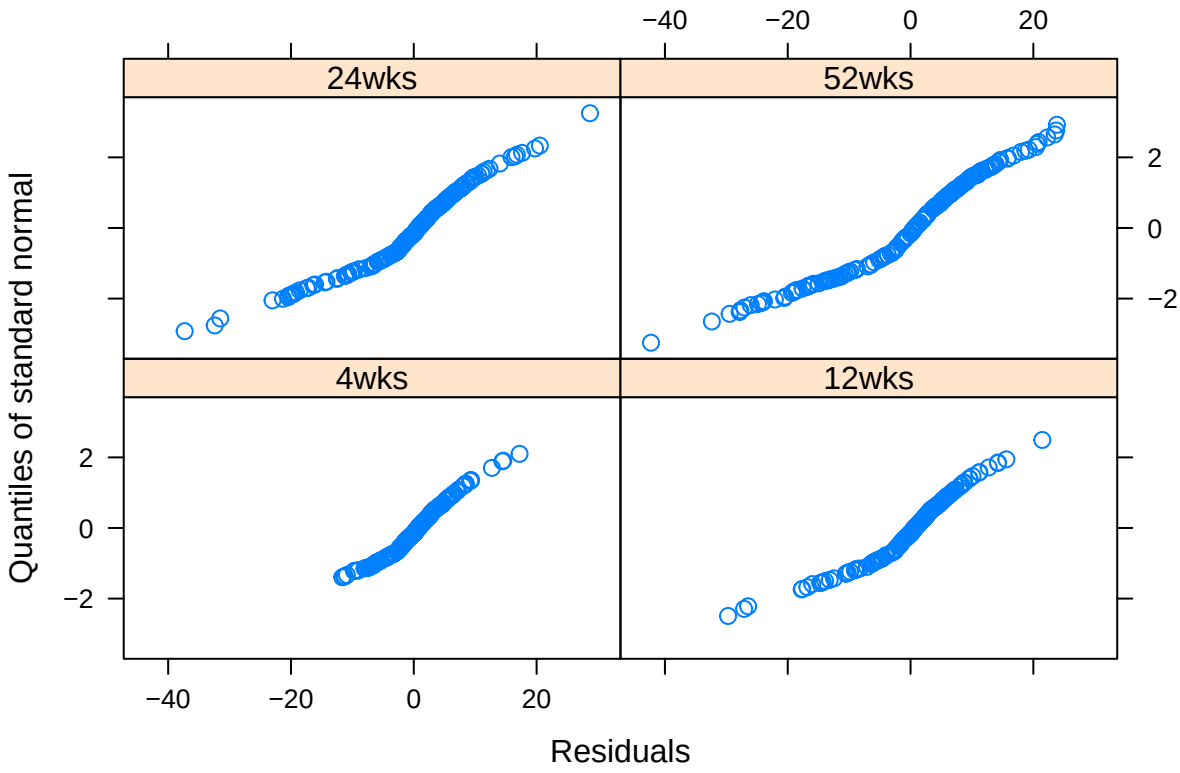

```r
# Plots (and boxplots) of Pearson residuals per time and treatment
plot(fm16.2, resid(., type = "pearson") ~ time | treat.f,
     id = 0.05)
```



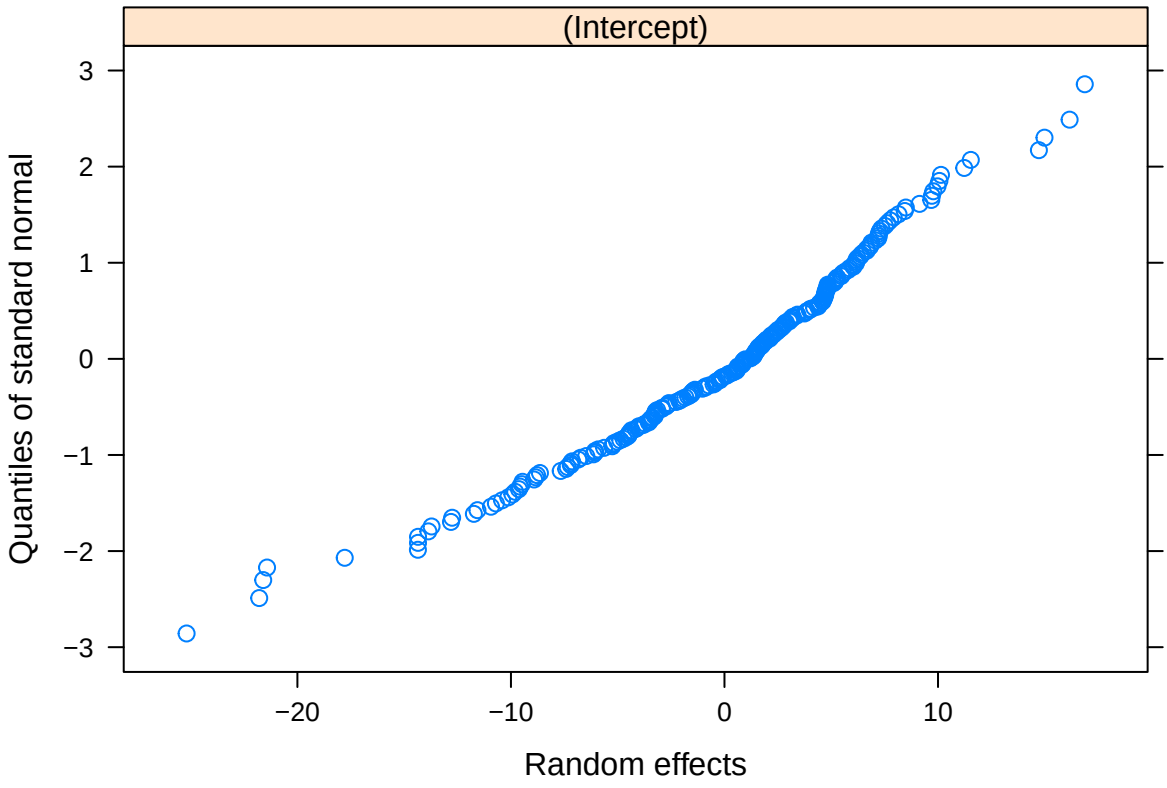
```r
bwplot(resid(fm16.2, type = "p") ~ time.f | treat.f,
       panel = panel.bwplot, # User-defined panel (not shown)
       data = armd)
```



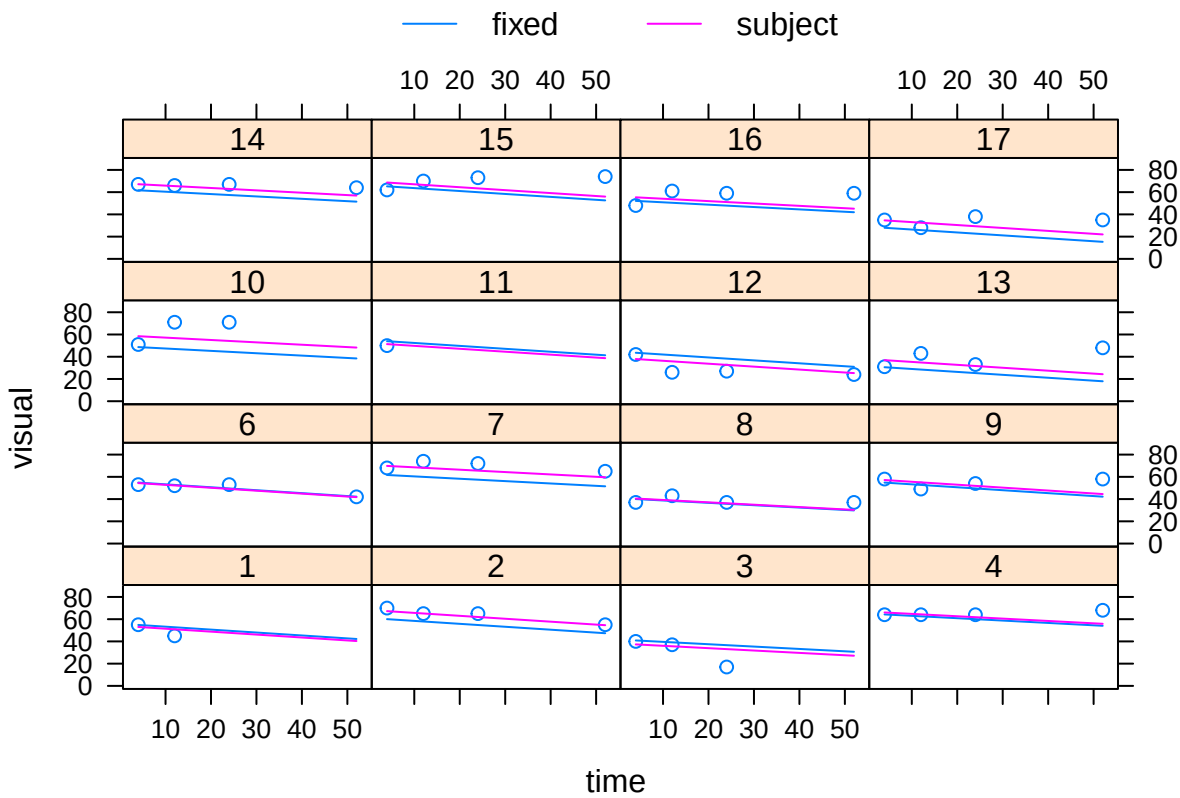

```r
# Despite standardization, the variability of the residuals seems to vary a bit.

# Normal Q-Q plots of Pearson residuals
qqnorm(fm16.2, ~resid(.) | time.f)
```




```
## ANALYSIS OF RANDOM EFFECTS
# Normal Q-Q plots of predicted random effects
qqnorm(fm16.2, ~ranef(.))
```



```r
## Computing predictions comparing population average predictions with patient-specific predictions

aug.Pred = augPred(fm16.2,
                   primary = ~time, # Primary covariate
                   level = 0:1, # fixed/marginal (0) and subj.-spec.(1)
                   length.out = 2) # evaluated in two time instants (4 e 52 wks)

plot(aug.Pred, layout = c(4, 4, 1))
```




```r
##############################################################################
## Model 2.1. random intercept + slope (correlated), heteroscedastic residuals (varPower of time)

fm16.3 = update(fm16.2,
                random = ~1 + time | subject,
                data = armd)
summary(fm16.3)
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC     BIC    logLik
##   6450.598 6498.19 -3215.299
##
## Random effects:
##  Formula: ~1 + time | subject
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev    Corr
## (Intercept) 6.9789085 (Intr)
## time        0.2722504 0.138
## Residual    5.1221953
##
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
##  Parameter estimates:
##     power
## 0.1074438
## Fixed effects:  list(lm2.form)
##                        Value Std.Error  DF   t-value p-value
```

```
## (Intercept)         4.740327 2.2561568 631  2.101063  0.0360
## visual0             0.909295 0.0381165 231 23.855712  0.0000
## time               -0.215062 0.0322436 631 -6.669904  0.0000
## treat.fActive      -2.260007 1.1497940 231 -1.965575  0.0505
## time:treat.fActive -0.057468 0.0470409 631 -1.221653  0.2223
##  Correlation:
##                    (Intr) visul0 time   trt.fA
## visual0            -0.934
## time               -0.071  0.003
## treat.fActive      -0.274  0.026  0.135
## time:treat.fActive  0.049 -0.002 -0.685 -0.203
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -4.08361467 -0.33477014  0.04144169  0.45248054  3.06682208
##
## Number of Observations: 867
## Number of Groups: 234
```

```
getVarCov(fm16.3, individual = "2")  # D_i italic (i=2)
```

```
## Random effects variance covariance matrix
##             (Intercept)    time
## (Intercept)    48.70500 0.26266
## time            0.26266 0.07412
##   Standard Deviations: 6.9789 0.27225
```

```
intervals(fm16.3, which = "var-cov")  # Estimate of theta_D, delta e sigma
```

```
## Approximate 95% confidence intervals
##
##  Random Effects:
##   Level: subject
##                            lower      est.     upper
## sd((Intercept))        5.9931243 6.9789085 8.1268401
## sd(time)               0.2301543 0.2722504 0.3220461
## cor((Intercept),time) -0.1215431 0.1382424 0.3803029
##
##  Variance function:
##            lower      est.     upper
## power 0.01728109 0.1074438 0.1976066
##
##  Within-group standard error:
##    lower     est.    upper
## 4.024129 5.122195 6.519892
```

```
################################################################################
## Model 2.2. random intercept + slope independent, heteroscedastic residuals (varPower of time)
fm16.4 = update(fm16.3,
                random = list(subject = pdDiag(~time)), # Diagonal D
                data = armd)
summary(fm16.4) ## results suggest to remove the Treat and Time interaction
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC      BIC    logLik
##   6449.792 6492.625 -3215.896
##
## Random effects:
##  Formula: ~time | subject
##  Structure: Diagonal
##         (Intercept)      time Residual
## StdDev:    7.231948 0.2809649 5.031159
##
## Variance function:
##  Structure: Power of variance covariate
##  Formula: ~time
##  Parameter estimates:
##     power
## 0.1110751
## Fixed effects:  list(lm2.form)
##                        Value Std.Error  DF   t-value p-value
```

```
## (Intercept)         5.262213 2.2658110 631  2.322441  0.0205
## visual0             0.899900 0.0381957 231 23.560227  0.0000
## time               -0.215031 0.0328493 631 -6.545989  0.0000
## treat.fActive      -2.278756 1.1724506 231 -1.943584  0.0532
## time:treat.fActive -0.056451 0.0479173 631 -1.178092  0.2392
##  Correlation:
##                    (Intr) visul0 time   trt.fA
## visual0            -0.932
## time               -0.098  0.002
## treat.fActive      -0.278  0.026  0.185
## time:treat.fActive  0.067 -0.001 -0.686 -0.272
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -4.1400193 -0.3231305  0.0474685  0.4405787  2.9422958
##
## Number of Observations: 867
## Number of Groups: 234
```

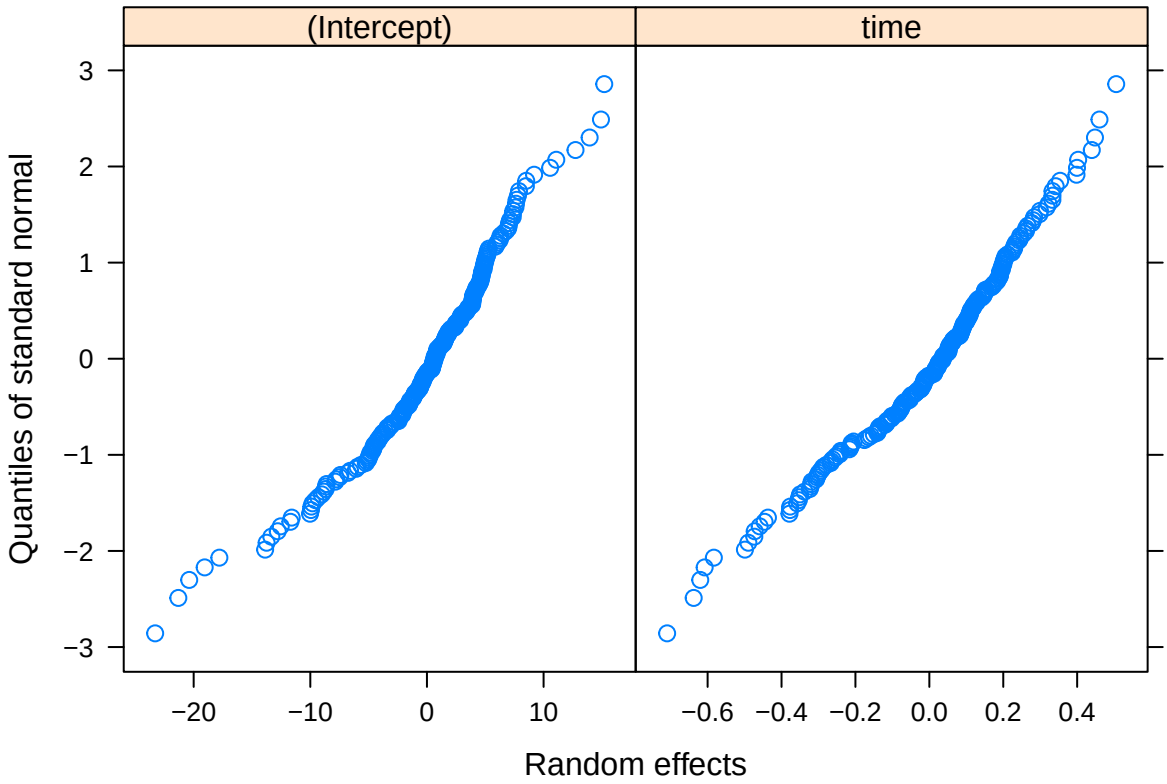
```
intervals(fm16.4)
```

```
## Approximate 95% confidence intervals
##
##  Fixed effects:
##                        lower       est.       upper
## (Intercept)         0.8127705  5.2622130  9.71165548
## visual0             0.8246435  0.8999000  0.97515652
## time               -0.2795380 -0.2150309 -0.15052381
## treat.fActive      -4.5888197 -2.2787559  0.03130786
## time:treat.fActive -0.1505476 -0.0564510  0.03764565
##
##  Random Effects:
##   Level: subject
##                     lower      est.     upper
## sd((Intercept)) 6.3306729 7.2319476 8.2615334
## sd(time)        0.2410843 0.2809649 0.3274426
##
##  Variance function:
##            lower      est.     upper
## power 0.01482336 0.1110751 0.2073269
##
##  Within-group standard error:
##    lower     est.    upper
## 3.897875 5.031159 6.493939
```

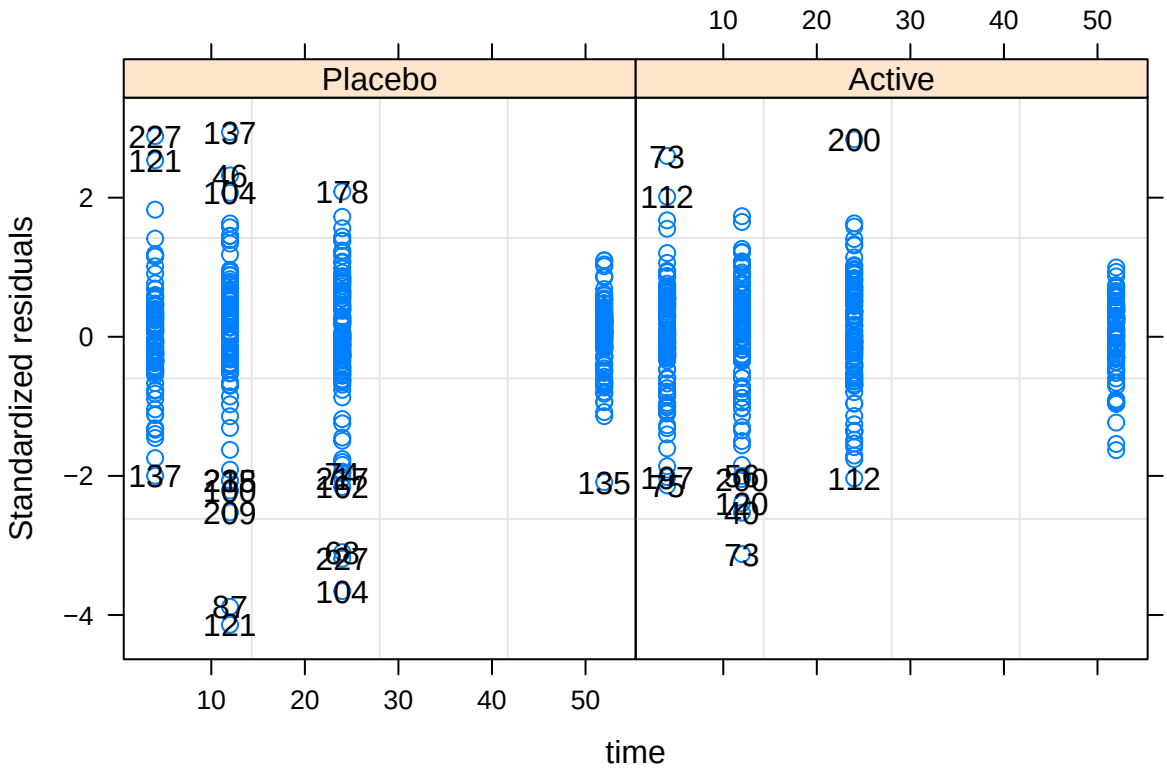
```
anova(fm16.4, fm16.3)
```

```
##        Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## fm16.4     1  9 6449.792 6492.625 -3215.896
## fm16.3     2 10 6450.598 6498.190 -3215.299 1 vs 2 1.194014  0.2745
```

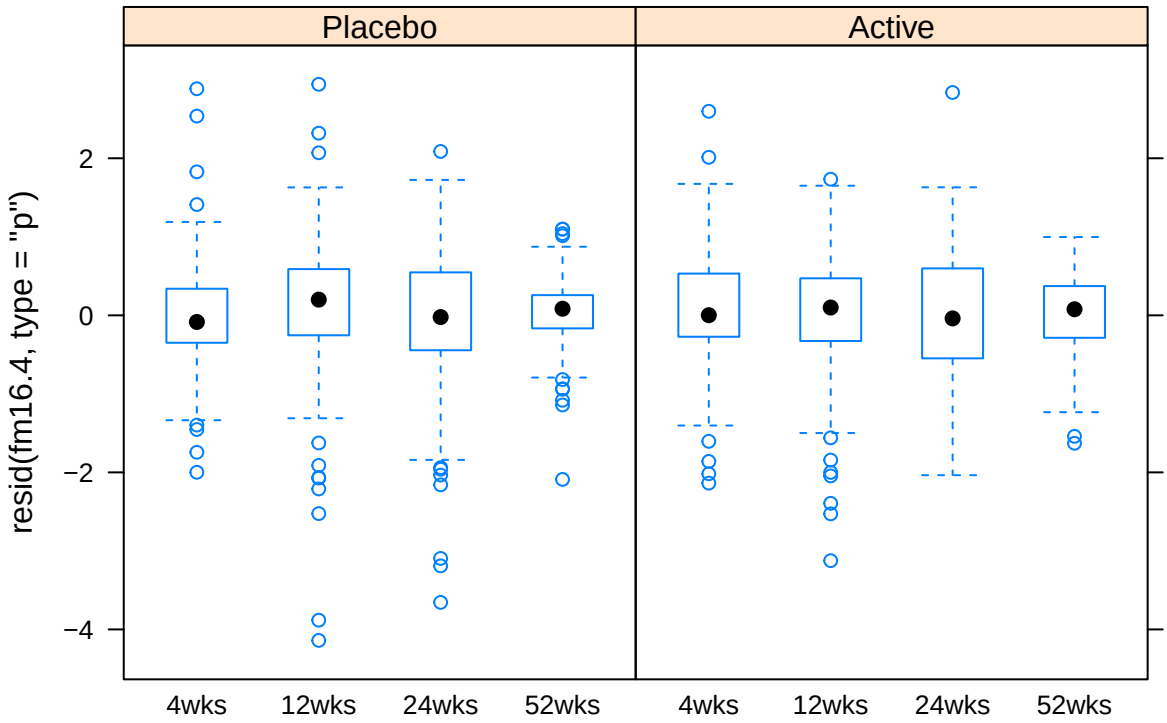
```
# We test if d_12 = 0 --> d_12 is not statistically
# different from 0, we can simplify the D structure in diagonal

qqnorm(fm16.4, ~ranef(.))
```



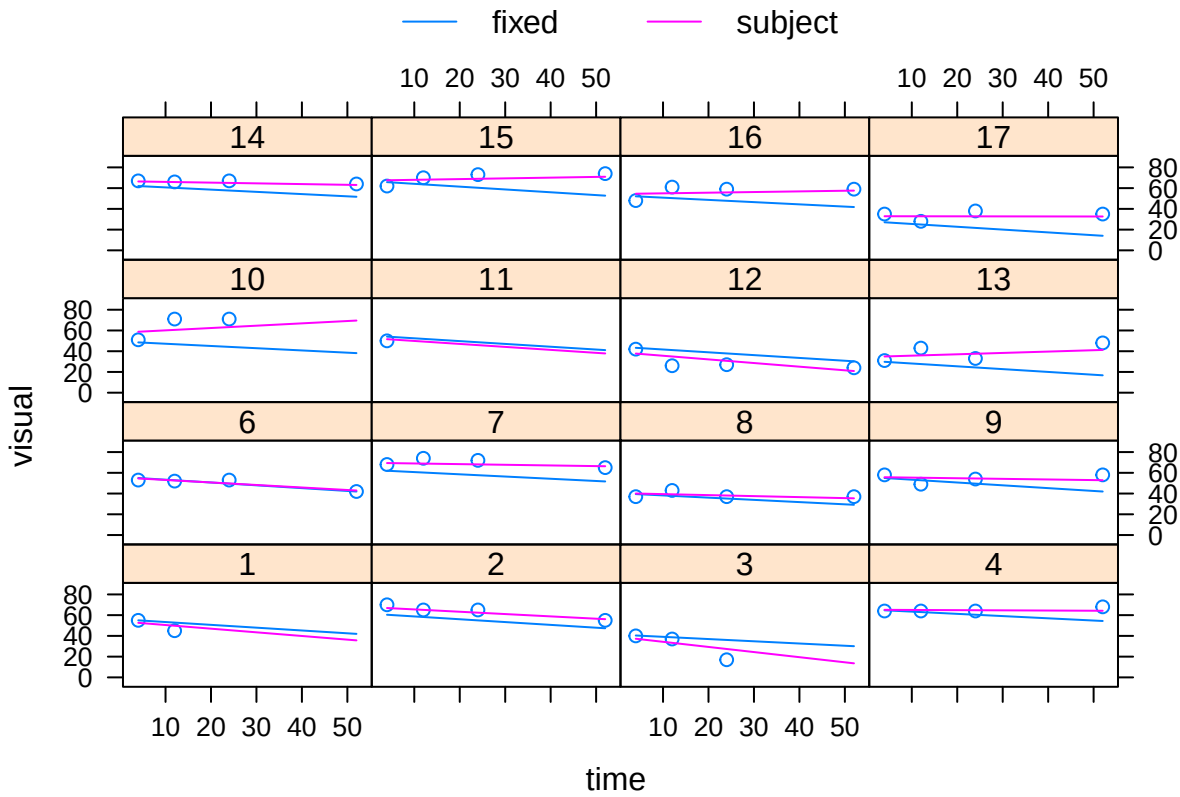

```
# to be interpreted with caution since it might not
# reflect the real unknown distribution

plot(fm16.4, resid(., type = "pearson") ~ time | treat.f,
     id = 0.05)
```



```r
bwplot(resid(fm16.4, type = "p") ~ time.f | treat.f,
       panel = panel.bwplot, # User-defined panel (not shown)
       data = armd)
```




```r
## We make predictions comparing population average predictions with patient specific predictions
aug.Pred = augPred(fm16.4,
                   primary = ~time, # Primary covariate
                   level = 0:1, # Marginal(0) and subj.-spec.(1)
                   length.out = 2) # Evaluated in two time instants (4 e 52 wks)

plot(aug.Pred, layout = c(4, 4, 1), columns = 2)
```




```
## let's compare the 4 fitted models
AIC(fm16.1,fm16.2,fm16.3, fm16.4)
```

```
##        df      AIC
## fm16.1  7 6591.971
## fm16.2  8 6537.126
## fm16.3 10 6450.598
## fm16.4  9 6449.792
```

```
anova(fm16.1,fm16.2,fm16.3, fm16.4)
```

```
##        Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## fm16.1     1  7 6591.971 6625.286 -3288.986
## fm16.2     2  8 6537.126 6575.200 -3260.563 1 vs 2 56.84531  <.0001
## fm16.3     3 10 6450.598 6498.190 -3215.299 2 vs 3 90.52824  <.0001
## fm16.4     4  9 6449.792 6492.625 -3215.896 3 vs 4  1.19401  0.2745
```

11.3 LMM 2 - Students and Schools

```
################################################################
#############  Students and Schools example  #################
################################################################

# Topics:
#   - Linear Mixed Models with Random Intercept
#   - Linear Mixed Models with Random Intercept + Random Slope
#   - Inference
#   - Assessing Assumptions
#   - Prediction

library(ggplot2)
library(insight)
library(lattice)
library(lme4)

# Dataset school - Student achievement
```

```r
# We have data from 1000 pupils who attend 50 different primary schools.
# Students are tested in mathematics at grade 5 with a standardized test across schools.
# The response variable is the achievement test score (numeric).
# We have two explanatory variables at the student level:
#   - pupil gender (1 = male, 0 = female)
#   - a scale centered in 0 for pupil socioeconomic status, pupil escs.
# Moreover, we know the anonymous school identification number.


school= read.table('Lab R MEMs 2022-2023/school.txt', header=T)
school$gender= as.factor(school$gender)
school$school_id= as.factor(school$school_id)
head(school)
```

```
##      achiev      escs gender school_id
## 1  9.622180 0.3735462      1         1
## 2  8.198914 1.1836433      1         1
## 3 11.060242 0.1643714      1         1
## 4 13.878223 2.5952808      0         1
## 5 14.165589 1.3295078      0         1
## 6 15.393015 0.1795316      0         1
```

```r
str(school)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ achiev   : num  9.62 8.2 11.06 13.88 14.17 ...
##  $ escs     : num  0.374 1.184 0.164 2.595 1.33 ...
##  $ gender   : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
##  $ school_id: Factor w/ 50 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
```

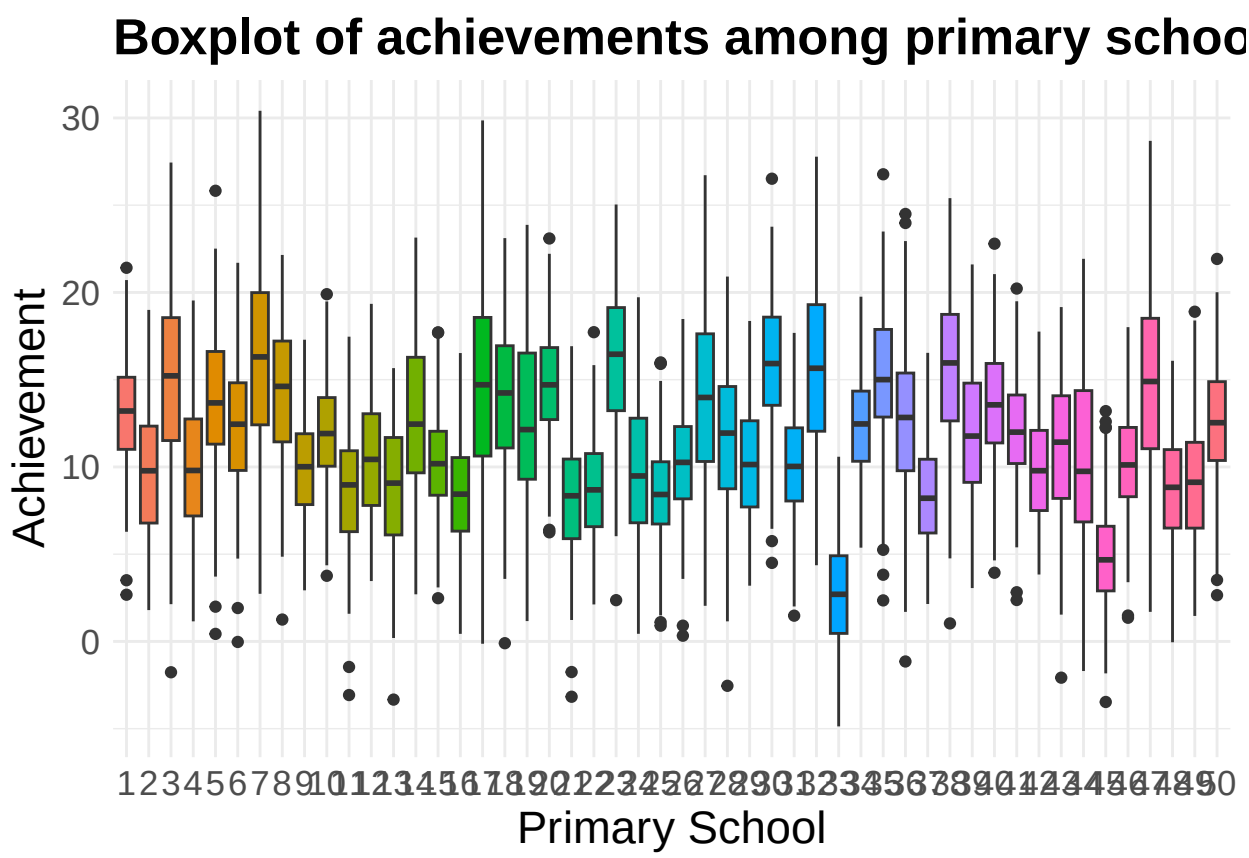
```r
# We look at achievement scores for students.
# The source of dependency is due to students attending the same primary school.

# For our mixed model we'll look at the effects for gender and socioeconomic status
# (escs) on scholastic achievement,
# taking into account the source of dependency given to the hierarchical structure.

summary(school)
```

```
##      achiev            escs          gender     school_id
##  Min.   :-4.868   Min.   :-2.0080   0:6260   1      : 200
##  1st Qu.: 8.213   1st Qu.: 0.3026   1:3740   2      : 200
##  Median :11.301   Median : 0.9647            3      : 200
##  Mean   :11.511   Mean   : 0.9884            4      : 200
##  3rd Qu.:14.666   3rd Qu.: 1.6884            5      : 200
##  Max.   :30.410   Max.   : 4.8103            6      : 200
##                                              (Other):8800
```

```r
sd(school$achiev)
```

```
## [1] 4.801066
```

```r
# Achievement variability in primary schools

ggplot(data=school, aes(x=as.factor(school_id), y=achiev, fill=as.factor(school_id))) +
  geom_boxplot() +
  labs(x='Primary School', y='Achievement') +
  ggtitle('Boxplot of achievements among primary schools') +
  theme_minimal() +
  theme(axis.text=element_text(size=rel(1.15)),axis.title=element_text(size=rel(1.5)),
        plot.title = element_text(face="bold", size=rel(1.75)),
        legend.text = element_text(size=rel(1.15)),
        legend.position = 'none')
```



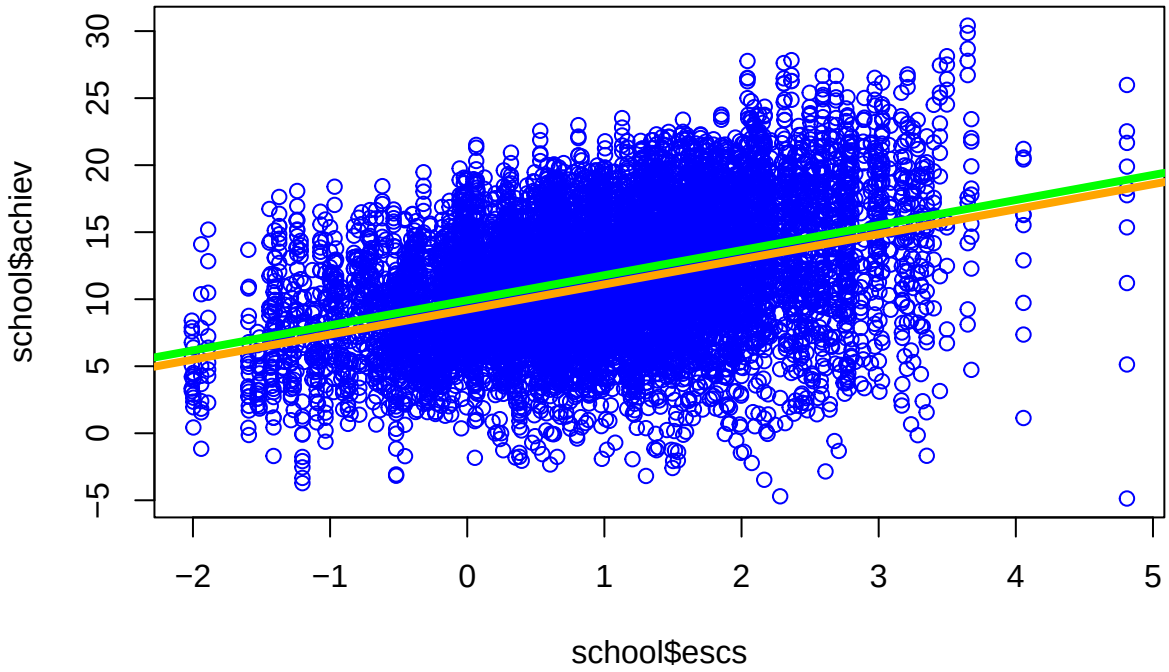

```
#! From all the scattered boxplots we can expect schools to have an impact on pupils
```

11.3.1 Linear model

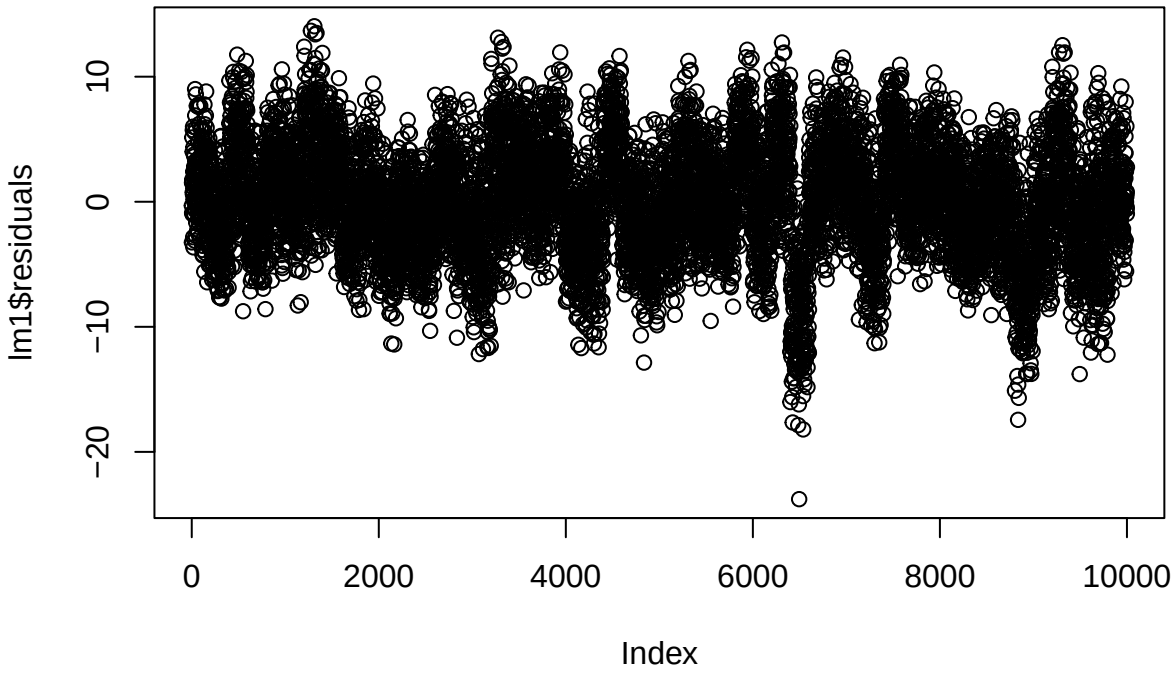
```
##--------------##
## Linear Model ##
##--------------##
# We start with a standard linear regression model, neglecting the dependence structure

# MODEL: achiev_i = beta_0 + beta_1*gender_i+ beta_2*escs_i + eps_i
# eps_i ~ N(0, sigma2_eps)

lm1 = lm(achiev ~ gender + escs, data = school)
summary(lm1)
```

```
##
## Call:
## lm(formula = achiev ~ gender + escs, data = school)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.781  -2.885   0.005   2.946  14.030
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.91880    0.06824 145.357  < 2e-16 ***
## gender1     -0.68298    0.09101  -7.505 6.68e-14 ***
## escs         1.86976    0.04257  43.922  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.392 on 9997 degrees of freedom
## Multiple R-squared:  0.1631, Adjusted R-squared:  0.163
## F-statistic: 974.4 on 2 and 9997 DF,  p-value: < 2.2e-16
```

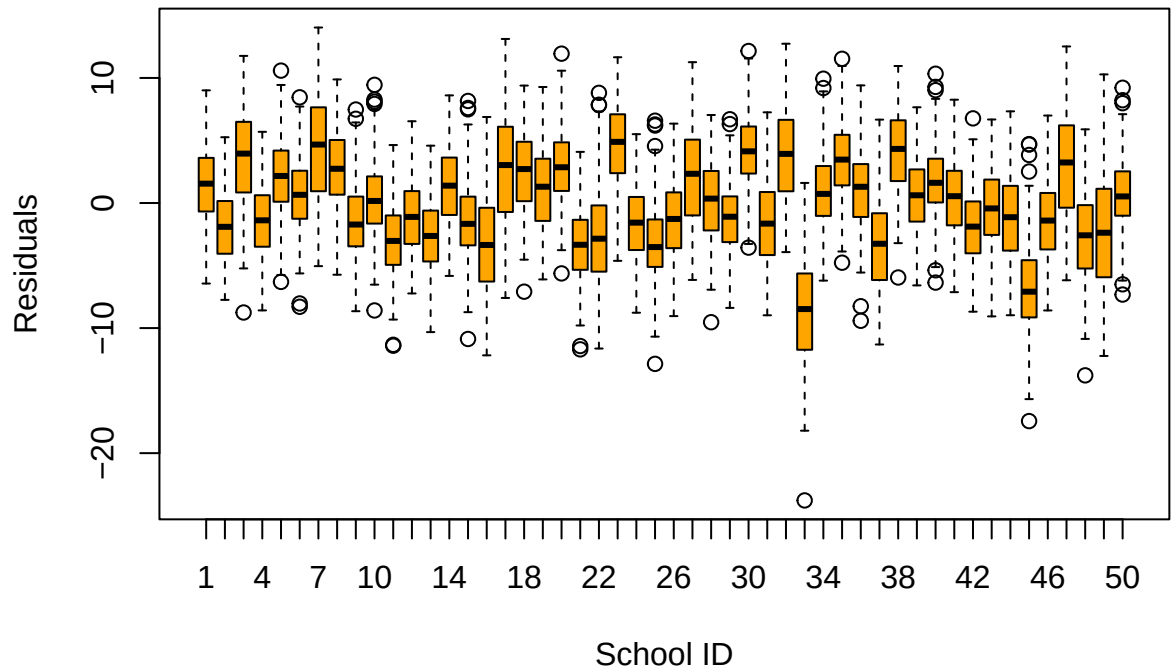
```
plot(school$escs,school$achiev, col='blue')
abline(9.91880,1.86976, col='green', lw=4)            # females
abline(9.91880 -0.68298,1.86976, col='orange', lw=4)  # males
```




```
plot(lm1$residuals)
```



```r
#! We see some pattern
boxplot(lm1$residuals ~ school$school_id, col='orange', xlab='School ID', ylab='Residuals')
```




```r
## residuals differ a lot across schools
```

11.3.2 LMM with random intercept

```r
#-----------------------------#
# Linear Mixed Effects Models #
#-----------------------------#
# We now take into account the clustering at primary school
# --> dependency among students within the same school

#! Random effect from the school
# MODEL: achiev_ij = beta_0 + beta_1*gender_ij + beta_2*escs_ij + b_i + eps_ij
# eps_ij ~ N(0, sigma2_eps)
# b_i ~ N(0, sigma2_b)

lmm1 = lmer(achiev ~ gender + escs + (1|school_id),
                      data = school)
summary(lmm1)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: achiev ~ gender + escs + (1 | school_id)
##    Data: school
##
## REML criterion at convergence: 52888.3
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.5186 -0.6925 -0.0124  0.6652  3.6889
##
## Random effects:
##  Groups    Name        Variance Std.Dev.
##  school_id (Intercept)  8.147   2.854
##  Residual              11.308   3.363
## Number of obs: 10000, groups:  school_id, 50
```

```
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  9.94471    0.40705   24.43
## gender1     -0.74963    0.07003  -10.71
## escs         1.86878    0.03263   57.27
##
## Correlation of Fixed Effects:
##         (Intr) gendr1
## gender1 -0.059
## escs    -0.075 -0.069
```

```r
# Fixed Effects and 95% CIs
#------------------------------
confint(lmm1, oldNames=TRUE)
```

11.3.2.1 Fixed Effects and 95% CIs

```
## Computing profile confidence intervals ...
```

```
##                 2.5 %     97.5 %
## .sig01       2.346938  3.4888342
## .sigma       3.316235  3.4096796
## (Intercept)  9.139738 10.7496943
## gender1     -0.886865 -0.6123712
## escs         1.804826  1.9327308
```

```r
fixef(lmm1)
```

```
## (Intercept)     gender1        escs
##   9.9447062  -0.7496277   1.8687782
```

```r
# The fixed effects tell us there is a negative effect of being male on achievement,
# and on average, students with higher escs are associated to higher achievement scores.
```

```r
# Variance components
#--------------------
# One thing that's new compared to the standard regression output is the estimated
# variance/standard deviation of the school effect.
# This tells us how much, on average, achievement bounces around as we move from school to school.
# In other words, even after making a prediction based on student covariates, each school has its
# own unique deviation, and that value (in terms of the standard deviation) is the estimated
# average deviation across schools.

vc = VarCorr(lmm1)
print(vc, comp = c("Variance", "Std.Dev."))
```

11.3.2.2 Variance components

```
##  Groups    Name        Variance Std.Dev.
##  school_id (Intercept)  8.1473  2.8544
##  Residual              11.3081  3.3628
```

```r
# help(get_variance)

sigma2_eps = as.numeric(get_variance_residual(lmm1))
sigma2_eps
```

```
## [1] 11.30812
```

```r
sigma2_b = as.numeric(get_variance_random(lmm1))
sigma2_b
```

```
## [1] 8.147344
```

```r
# Another way to interpret the variance output is to note percentage of the student variance out
# of the total, ie the Percentage of Variance explained by the Random Effect (PVRE).
# This is also called the intraclass correlation (ICC), because it is also an estimate of the within
# cluster correlation.
PVRE = sigma2_b/(sigma2_b+sigma2_eps)
PVRE
```

```
## [1] 0.418769
```

```r
# PVRE = 41.8% is very high!
```

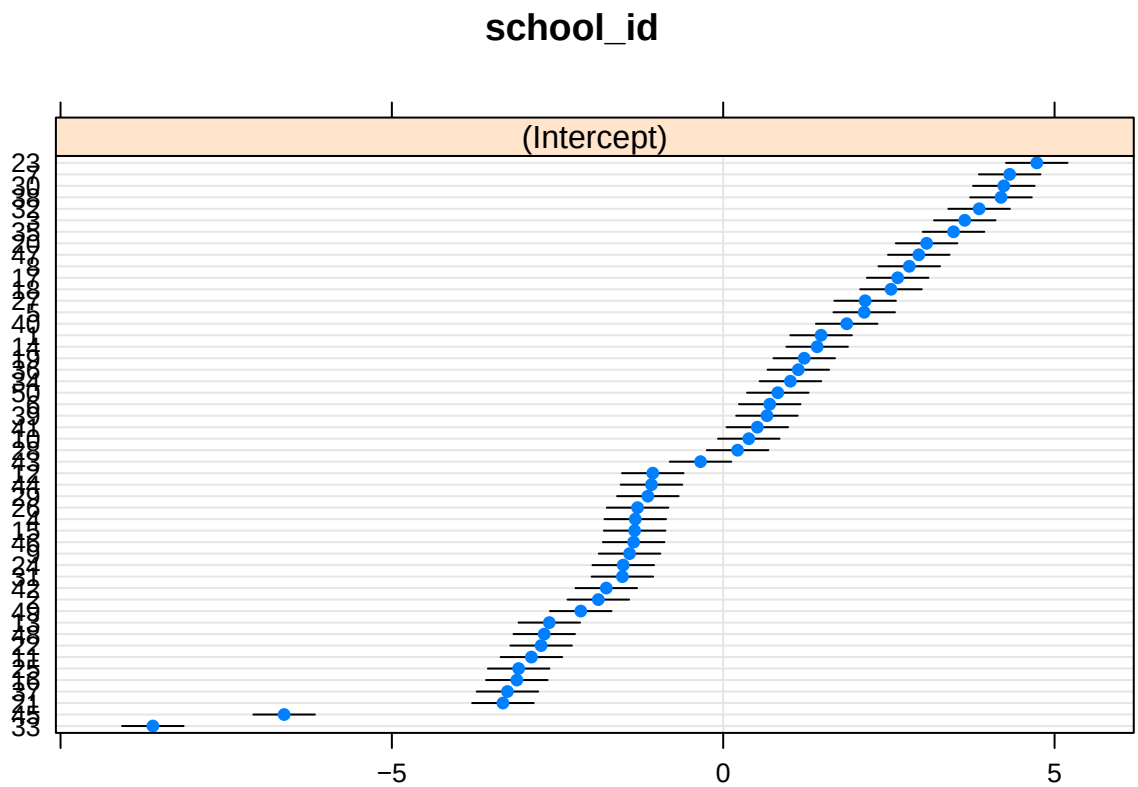
```r
# Random effects: b_0i
#----------------------------
ranef(lmm1)
```

11.3.2.3 Random effects: b_0i

```
## $school_id
##    (Intercept)
## 1    1.4766025
## 2   -1.8844992
## 3    3.6448403
## 4   -1.3273233
## 5    2.1276403
## 6    0.7020905
## 7    4.3226856
## 8    2.8074982
## 9   -1.4133226
## 10   0.3855486
## 11  -2.8943849
## 12  -1.0629283
## 13  -2.6244217
## 14   1.4176093
## 15  -1.3366242
## 16  -3.1140854
## 17   2.6328050
## 18   2.5318488
## 19   1.2223303
## 20   3.0693463
## 21  -3.3239385
## 22  -2.7477415
## 23   4.7321018
## 24  -1.5085914
## 25  -3.0870979
## 26  -1.2925018
## 27   2.1414746
## 28   0.2171461
## 29  -1.1372696
## 30   4.2345059
## 31  -1.5212744
## 32   3.8622612
## 33  -8.6067935
## 34   1.0140872
## 35   3.4757288
## 36   1.1327471
## 37  -3.2562603
## 38   4.1920185
## 39   0.6598417
## 40   1.8637605
## 41   0.5149956
## 42  -1.7648624
## 43  -0.3423086
## 44  -1.0824888
## 45  -6.6256707
## 46  -1.3512671
## 47   2.9513336
## 48  -2.7011042
## 49  -2.1495245
## 50   0.8234367
##
## with conditional variances for "school_id"
```

```r
#! ranef extracts the random effect values

# The dotplot shows the point and interval estimates for the random effects,
# ordering them and highlighting which are significantly different from the mean (0)


dotplot(ranef(lmm1))
```

```
## $school_id
```




```r
# Random intercepts and fixed slopes: (beta_0+b_0i, beta_1, beta_2)
coef(lmm1)
```

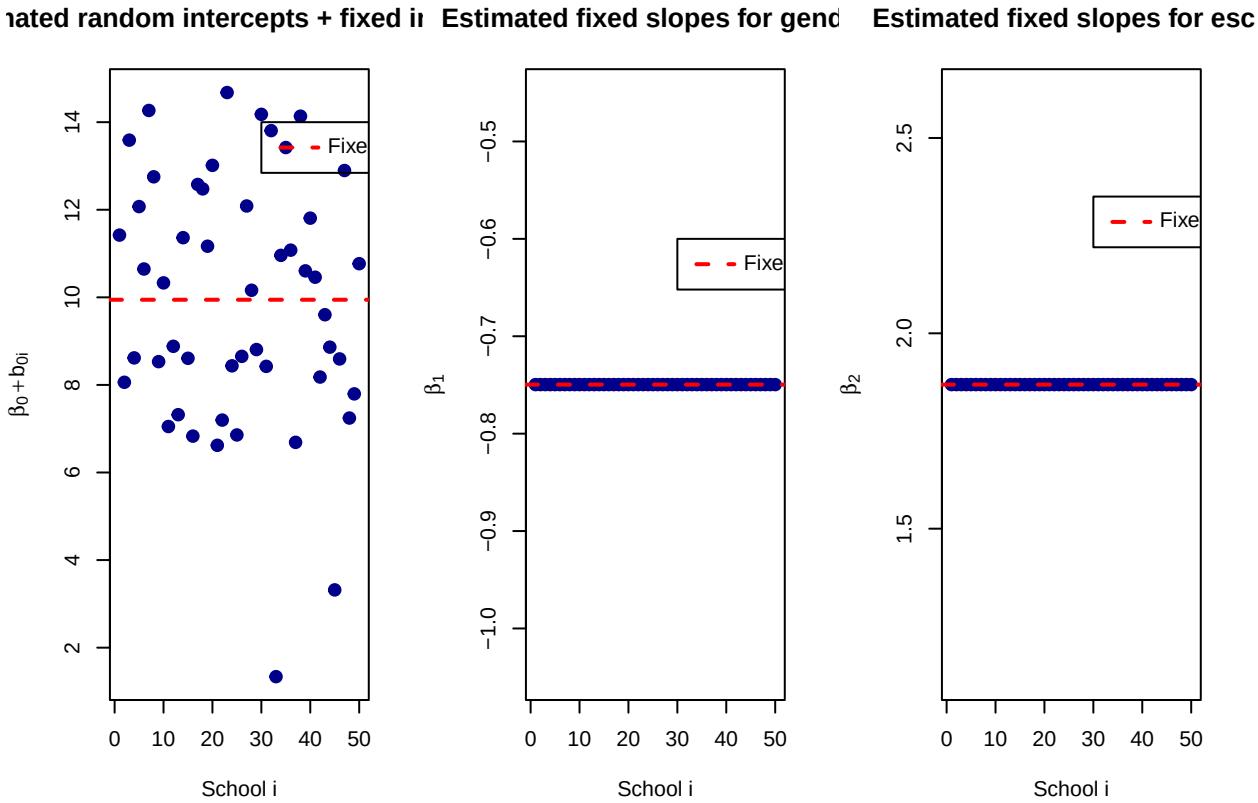
```
## $school_id
##    (Intercept)    gender1     escs
## 1    11.421309 -0.7496277 1.868778
## 2     8.060207 -0.7496277 1.868778
## 3    13.589547 -0.7496277 1.868778
## 4     8.617383 -0.7496277 1.868778
## 5    12.072346 -0.7496277 1.868778
## 6    10.646797 -0.7496277 1.868778
## 7    14.267392 -0.7496277 1.868778
## 8    12.752204 -0.7496277 1.868778
## 9     8.531384 -0.7496277 1.868778
## 10   10.330255 -0.7496277 1.868778
## 11    7.050321 -0.7496277 1.868778
## 12    8.881778 -0.7496277 1.868778
## 13    7.320285 -0.7496277 1.868778
## 14   11.362316 -0.7496277 1.868778
## 15    8.608082 -0.7496277 1.868778
## 16    6.830621 -0.7496277 1.868778
## 17   12.577511 -0.7496277 1.868778
## 18   12.476555 -0.7496277 1.868778
## 19   11.167037 -0.7496277 1.868778
## 20   13.014053 -0.7496277 1.868778
## 21    6.620768 -0.7496277 1.868778
## 22    7.196965 -0.7496277 1.868778
## 23   14.676808 -0.7496277 1.868778
```

```
## 24    8.436115 -0.7496277 1.868778
## 25    6.857608 -0.7496277 1.868778
## 26    8.652204 -0.7496277 1.868778
## 27   12.086181 -0.7496277 1.868778
## 28   10.161852 -0.7496277 1.868778
## 29    8.807437 -0.7496277 1.868778
## 30   14.179212 -0.7496277 1.868778
## 31    8.423432 -0.7496277 1.868778
## 32   13.806967 -0.7496277 1.868778
## 33    1.337913 -0.7496277 1.868778
## 34   10.958793 -0.7496277 1.868778
## 35   13.420435 -0.7496277 1.868778
## 36   11.077453 -0.7496277 1.868778
## 37    6.688446 -0.7496277 1.868778
## 38   14.136725 -0.7496277 1.868778
## 39   10.604548 -0.7496277 1.868778
## 40   11.808467 -0.7496277 1.868778
## 41   10.459702 -0.7496277 1.868778
## 42    8.179844 -0.7496277 1.868778
## 43    9.602398 -0.7496277 1.868778
## 44    8.862217 -0.7496277 1.868778
## 45    3.319036 -0.7496277 1.868778
## 46    8.593439 -0.7496277 1.868778
## 47   12.896040 -0.7496277 1.868778
## 48    7.243602 -0.7496277 1.868778
## 49    7.795182 -0.7496277 1.868778
## 50   10.768143 -0.7496277 1.868778
##
## attr(,"class")
## [1] "coef.mer"
```

```
head(coef(lmm1)$school_id)
```

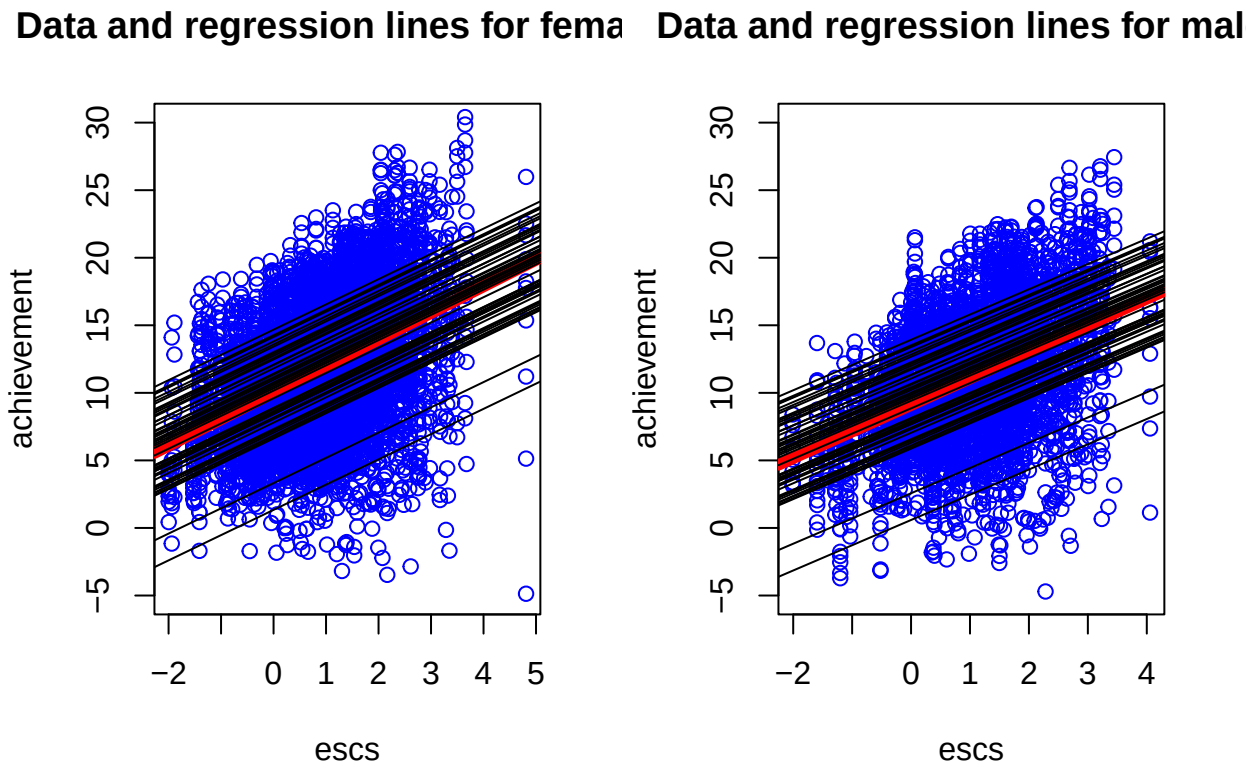
```
##   (Intercept)    gender1     escs
## 1   11.421309 -0.7496277 1.868778
## 2    8.060207 -0.7496277 1.868778
## 3   13.589547 -0.7496277 1.868778
## 4    8.617383 -0.7496277 1.868778
## 5   12.072346 -0.7496277 1.868778
## 6   10.646797 -0.7496277 1.868778
```

```
## visualization of the coefficients

par(mfrow=c(1,3))
plot(c(1:50), unlist(coef(lmm1)$school_id[1]),
     xlab='School i', ylab=expression(beta[0]+b['0i']),
     pch=19, lwd=2, col='darkblue',
     main='Estimated random intercepts + fixed intercepts')
abline(h=fixef(lmm1)[1], lty=2, col='red', lwd=2)
legend(30, 14, legend=expression(paste('Fixed intercept ',beta[0])), lwd=2, lty=2,
       col='red', x.intersp=0.5)
plot(c(1:50), unlist(coef(lmm1)$school_id[2]),
     xlab='School i', ylab=expression(beta[1]),
     pch=19, lwd=2, col='darkblue',
     main='Estimated fixed slopes for gender')
abline(h=fixef(lmm1)[2], lty=2, col='red', lwd=2)
legend(30, -0.6, legend=expression(paste('Fixed slope ',beta[1])), lwd=2, lty=2,
       col='red', x.intersp=0.5)
plot(c(1:50), unlist(coef(lmm1)$school_id[3]),
     xlab='School i', ylab=expression(beta[2]),
     pch=19, lwd=2, col='darkblue',
     main='Estimated fixed slopes for escs')
abline(h=fixef(lmm1)[3], lty=2, col='red', lwd=2)
legend(30, 2.35, legend=expression(paste('Fixed slope ',beta[2])), lwd=2, lty=2,
       col='red', x.intersp=0.5)
```



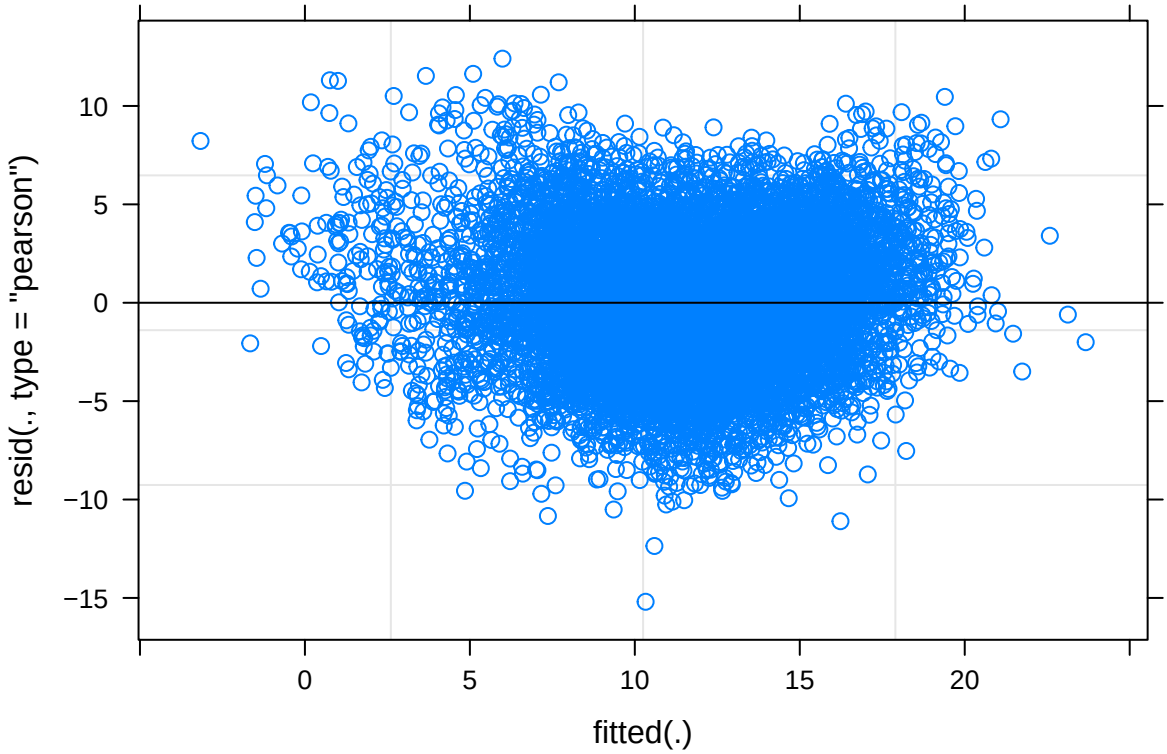

```
# Let's plot all the regression lines
## FEMALES

par(mfrow=c(1,2))
plot(school$escs[school$gender==0], school$achiev[school$gender==0],col='blue',
     xlab='escs', ylab='achievement',ylim=c(-5,30),
     main='Data and regression lines for females')
abline(10.02507,1.96618, col='red', lw=6)

for(i in 1:50){
  abline(coef(lmm1)$school_id[i,1], coef(lmm1)$school_id[i,3])
}

## MALES
plot(school$escs[school$gender==1], school$achiev[school$gender==1],col='blue',
     xlab='escs', ylab='achievement',ylim=c(-5,30),
     main='Data and regression lines for males')
abline(10.02507-0.91180,1.96618, col='red', lw=6)

for(i in 1:50){
  abline(coef(lmm1)$school_id[i,1] + coef(lmm1)$school_id[i,2], coef(lmm1)$school_id[i,3])
}
```

11.3.2.4 All the regression lines

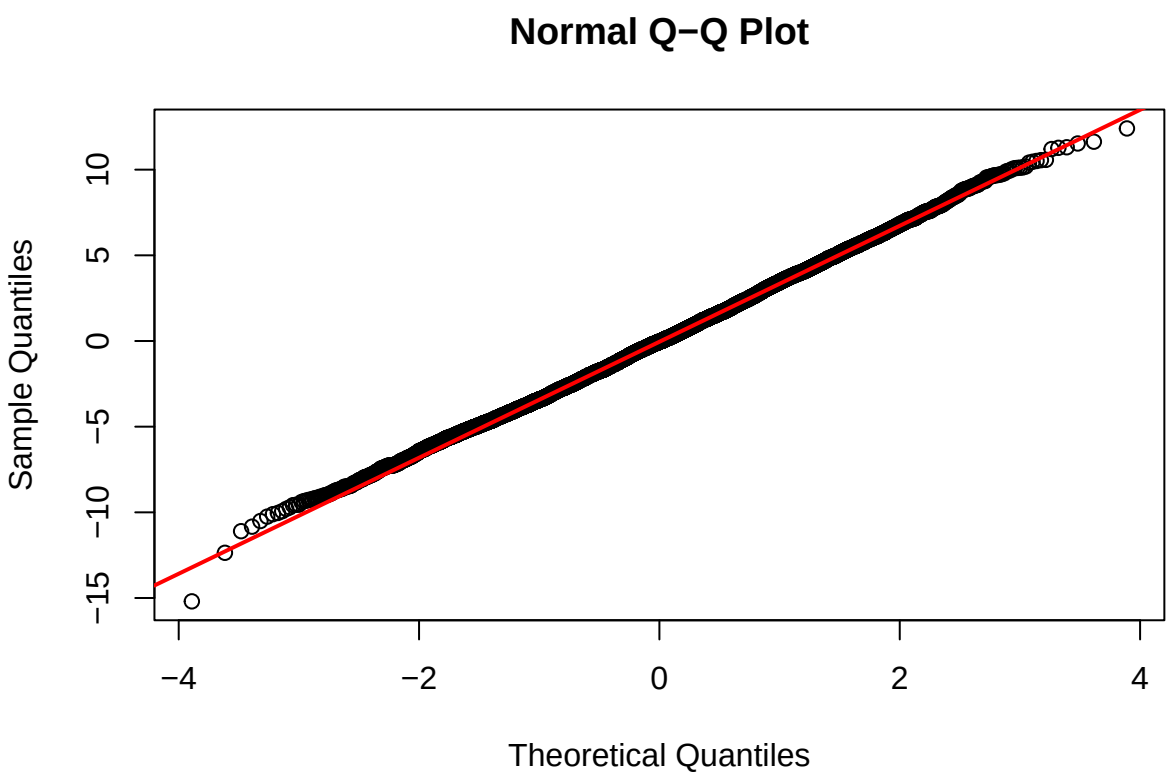



```
# Diagnostic plots
#------------------
# 1) Assessing Assumption on the within-group errors

plot(lmm1)
```

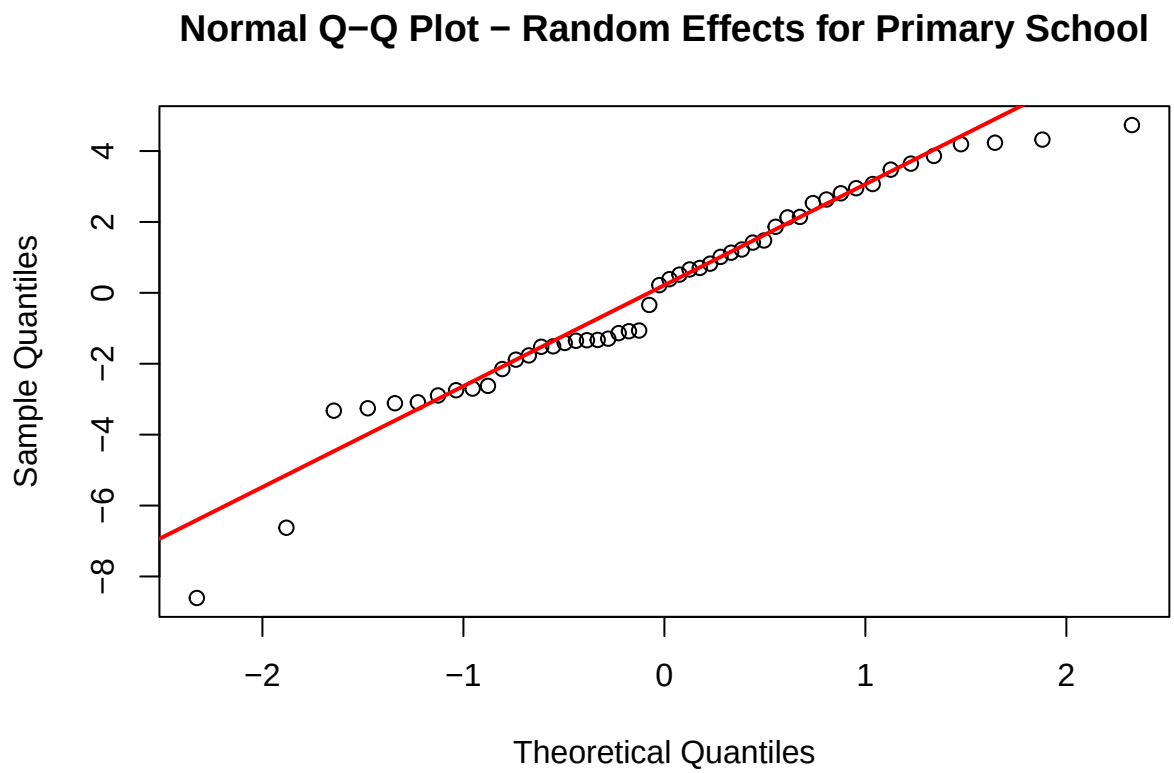
11.3.2.5 Diagnostic plots




```
qqnorm(resid(lmm1))
qqline(resid(lmm1), col='red', lwd=2)
```



```r
# 2) Assessing Assumption on the Random Effects

qqnorm(unlist(ranef(lmm1)$school_id), main='Normal Q-Q Plot - Random Effects for Primary School')
qqline(unlist(ranef(lmm1)$school_id), col='red', lwd=2)
```




```r
# Prediction
#-------------
# Let's now examine standard predictions vs. cluster-specific predictions.
# As with most R models, we can use the predict function on the model object.

# Prediction from regression model
predict_lm = predict(lm1)
head(predict_lm)
```

11.3.2.6 Prediction

```
##         1         2         3         4         5         6
##  9.934271 11.448961  9.543163 14.771367 12.404670 10.254486
```

```r
# Prediction from mixed model:
# 1) Without random effects ->  re.form=NA
predict_no_re = predict(lmm1, re.form=NA)
head(predict_no_re) # same predictions
```

```
##         1         2         3         4         5         6
##  9.893153 11.407045  9.502252 14.794710 12.429261 10.280211
```

```r
# 2) With random effects
predict_re = predict(lmm1)     ## --> remember to allow new levels in the RE if any
head(predict_re)
```

```
##        1        2        3        4        5        6
## 11.36976 12.88365 10.97885 16.27131 13.90586 11.75681
```

```r
## Scenario Analysis

# Let's imagine to observe three new students with the same personal characteristics
# but enrolled in different schools, two of them are observed and one is new

new_student1 = data.frame(gender=as.factor(1), escs=0.7, school_id=32) # observed school
new_student2 = data.frame(gender=as.factor(1), escs=0.7, school_id=11) # observed school
new_student3= data.frame(gender=as.factor(1), escs=0.7, school_id=53) # new school

predict(lmm1, new_student1, re.form=NA)
```

11.3.2.7 Scenario Analysis

```
##        1
## 10.50322
```

```r
predict(lmm1, new_student1)
```

```
##        1
## 14.36548
```

```r
predict(lmm1, new_student2, re.form=NA)
```

```
##        1
## 10.50322
```

```r
predict(lmm1, new_student2)
```

```
##        1
## 7.608838
```

```r
predict(lmm1, new_student3, re.form=NA)
```

```
##        1
## 10.50322
```

```r
predict(lmm1, new_student3, allow.new.levels = T)
```

```
##        1
## 10.50322
```

11.3.3 LMM with random intercept and slope

```r
#-----------------------------------------------------#
# Linear Mixed Model with Random Intercept & Slope #
#-----------------------------------------------------#

## We now consider the possibility that the association between escs and student achievements
## differs across schools.
## We include a random slope for the escs to model this additional source of heterogeneity.

# MODEL:  achiev_ij = beta_0 + b_0i + (beta_1 + b_1i)*escs_i + eps_i --> homoscedastic residuals

# eps_i ~ N(0, sigma2_eps)
# Random effects: b_i ~ N(0, Sigma)

# To allow both the intercept, represented by 1, and the slope, represented by escs,
# to vary by student we can add the term:
#   - (1+escs|school_id)
# or, in alternative, without 1
#   - (escs|school_id)

lmm2 = lmer(achiev ~ gender + escs + (1 + escs|school_id),
                data = school)
summary(lmm2)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: achiev ~ gender + escs + (1 + escs | school_id)
##    Data: school
##
## REML criterion at convergence: 50229.1
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.90633 -0.68740 -0.02044  0.66667  2.88284
##
## Random effects:
##  Groups    Name        Variance Std.Dev. Corr
##  school_id (Intercept) 4.323    2.079
##            escs        2.706    1.645    0.16
##  Residual              8.504    2.916
## Number of obs: 10000, groups:  school_id, 50
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   9.9573     0.2976  33.463
## gender1      -0.7073     0.0609 -11.614
## escs          1.8437     0.2344   7.866
##
## Correlation of Fixed Effects:
##         (Intr) gendr1
## gender1 -0.070
## escs     0.150 -0.008
```

```
confint(lmm2, oldNames=TRUE)
```

```
## Computing profile confidence intervals ...
```

```
##                  2.5 %     97.5 %
## .sig01       1.7047748  2.5461007
## .sig02      -0.1174097  0.4219796
## .sig03       1.3499952  2.0133876
## .sigma       2.8759146  2.9571592
## (Intercept)  9.3687405 10.5457934
## gender1     -0.8266161 -0.5878956
## escs         1.3799996  2.3072871
```

```
# Note that the mean slope for the escs effect, our fixed effect, is 1.84, but
# from school to school it bounces around.

# Yet another point of interest is the correlation of the intercepts and slopes.
# In this case it's 0.16.
# That's pretty small, but the interpretation is the same as with any correlation.
```

```
# Variance components
#--------------------
# In this case the variance of random sigma2_R effects represents the mean random
# effect variance of the model and is given by
# sigma2_b = Var(b0,b1) = sigma2_b0 + 2Cov(b0,b1)*mean(w) + sigma2_b1*mean(w^2)
# See equation (10) in Johnson (2014), Methods in Ecology and Evolution, 5(9), 944-946.

vc = VarCorr(lmm2)
print(vc, comp = c("Variance", "Std.Dev."))
```

11.3.3.1 Variance components

```
##  Groups    Name        Variance Std.Dev. Corr
##  school_id (Intercept) 4.3228   2.0791
##            escs        2.7063   1.6451   0.164
##  Residual              8.5043   2.9162
```

```
sigma2_eps = as.numeric(get_variance_residual(lmm2))
sigma2_eps
```

```
## [1] 8.504294
```

```r
sigma2_b = as.numeric(get_variance_random(lmm2))  ## it automatically computes Var(b0,b1)
# 4.3228 + 2*0.164*2.0791*1.6451* mean(school$escs, na.rm=T) + 2.7063*mean(school$escs^2, na.rm=T)
sigma2_b
```

```
## [1] 10.97399
```

```r
PVRE = sigma2_b/(sigma2_b+sigma2_eps)
PVRE
```

```
## [1] 0.5633962
```

```r
# PVRE = 56%
```

```r
# Estimates of fixed and random effects
#-------------------------------------

# Fixed effects: (beta_0, beta_1, beta_2)
fixef(lmm2)
```

11.3.3.2 Estimates of fixed and random effects

```
## (Intercept)     gender1        escs
##   9.9572683  -0.7072477   1.8436695
```

```r
# Random effects: (b_0i, b_1i) for i=1,...,200
ranef(lmm2)
```

```
## $school_id
##     (Intercept)        escs
## 1   2.179993751 -0.69400715
## 2  -2.319869379  0.42737507
## 3   1.559561525  2.18335357
## 4  -1.912985711  0.66055405
## 5   1.006952872  1.11550060
## 6   0.194803168  0.49171821
## 7   1.283751457  2.94072681
## 8   2.383852262  0.43058017
## 9  -0.298223514 -1.25173068
## 10  1.122718334 -0.74636335
## 11 -2.970736519  0.08154332
## 12 -1.033760088 -0.02695618
## 13 -2.602850485 -0.02178885
## 14  0.344454713  1.19907429
## 15 -0.118099931 -1.22474707
## 16  0.007548065 -3.04357814
## 17 -0.819477791  3.34792012
## 18  1.651083512  0.91445042
## 19 -0.350679060  1.76187436
## 20  3.554513494 -0.50153832
## 21 -3.179443945 -0.13299986
## 22  0.125898066 -2.78842671
## 23  3.930367280  0.82365675
## 24 -2.291881042  0.88343516
## 25 -1.654219983 -1.43467498
## 26 -0.261106656 -1.00611187
## 27 -0.292186273  2.35867938
## 28 -0.681692125  0.94122040
## 29 -0.985302023 -0.16961714
## 30  3.713775017  0.50582903
## 31  0.724537209 -2.19241293
## 32  1.658748708  2.12957753
## 33 -6.102245985 -2.62526790
## 34  2.120645144 -1.25069329
## 35  2.518253185  0.94720131
## 36 -0.368746411  1.46216053
## 37 -1.003757222 -2.18273937
## 38  3.060338646  1.17404190
## 39  0.016445784  0.71846450
## 40  1.962939121 -0.10977231
```

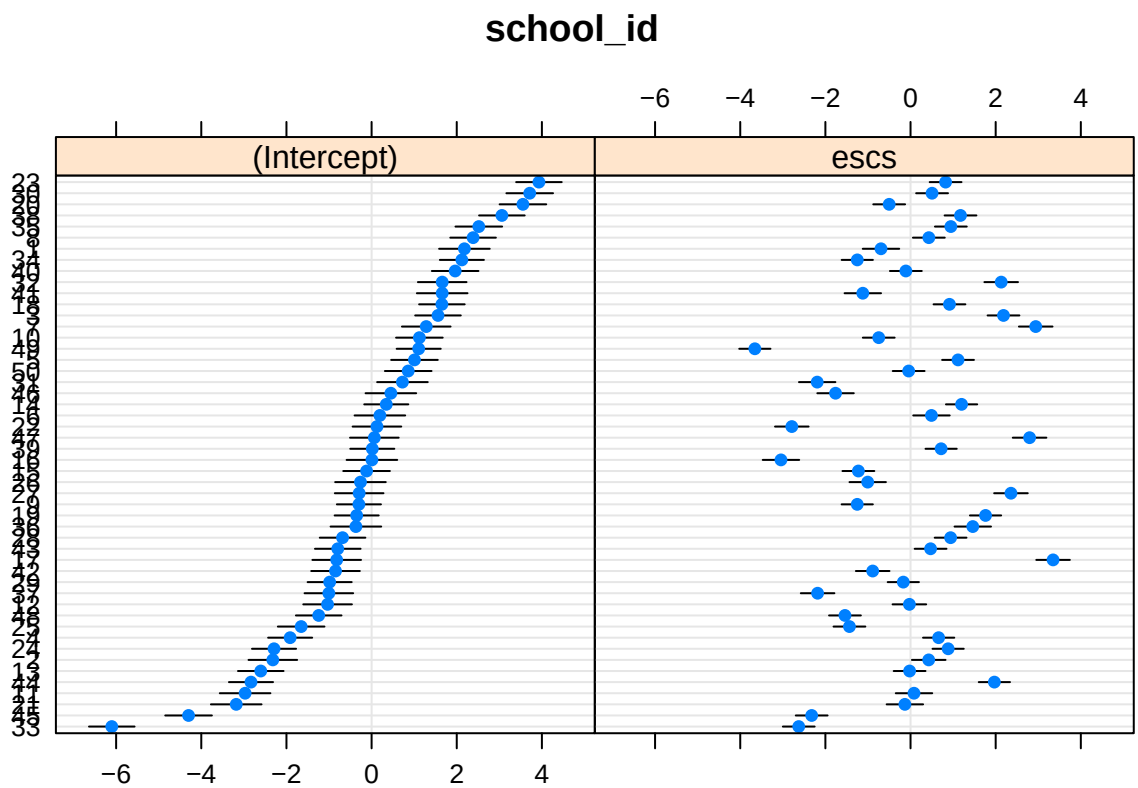
```
## 41  1.658054070 -1.12079954
## 42 -0.847237839 -0.88895126
## 43 -0.794387714  0.47106516
## 44 -2.832924312  1.96953342
## 45 -4.299048238 -2.32182783
## 46  0.452730207 -1.76109345
## 47  0.064826536  2.79665827
## 48 -1.242137695 -1.54017288
## 49  1.106131811 -3.65629200
## 50  0.860076003 -0.04363129
##
## with conditional variances for "school_id"
```

```
head(ranef(lmm2)$school_id)
```

```
##   (Intercept)       escs
## 1   2.1799938 -0.6940071
## 2  -2.3198694  0.4273751
## 3   1.5595615  2.1833536
## 4  -1.9129857  0.6605541
## 5   1.0069529  1.1155006
## 6   0.1948032  0.4917182
```

```
dotplot(ranef(lmm2))
```

```
## $school_id
```




```
# Random intercepts and slopes: (beta_0+b_0i, beta_1, beta_2+b_2i)
coef(lmm2)
```

```
## $school_id
##    (Intercept)    gender1        escs
## 1    12.137262 -0.7072477  1.14966234
## 2     7.637399 -0.7072477  2.27104455
## 3    11.516830 -0.7072477  4.02702306
## 4     8.044283 -0.7072477  2.50422354
## 5    10.964221 -0.7072477  2.95917008
```

```
## 6    10.152071 -0.7072477  2.33538770
## 7    11.241020 -0.7072477  4.78439630
## 8    12.341121 -0.7072477  2.27424965
## 9     9.659045 -0.7072477  0.59193881
## 10   11.079987 -0.7072477  1.09730613
## 11    6.986532 -0.7072477  1.92521280
## 12    8.923508 -0.7072477  1.81671331
## 13    7.354418 -0.7072477  1.82188064
## 14   10.301723 -0.7072477  3.04274377
## 15    9.839168 -0.7072477  0.61892242
## 16    9.964816 -0.7072477 -1.19990865
## 17    9.137790 -0.7072477  5.19158960
## 18   11.608352 -0.7072477  2.75811991
## 19    9.606589 -0.7072477  3.60554384
## 20   13.511782 -0.7072477  1.34213117
## 21    6.777824 -0.7072477  1.71066962
## 22   10.083166 -0.7072477 -0.94475723
## 23   13.887636 -0.7072477  2.66732623
## 24    7.665387 -0.7072477  2.72710465
## 25    8.303048 -0.7072477  0.40899450
## 26    9.696162 -0.7072477  0.83755761
## 27    9.665082 -0.7072477  4.20234886
## 28    9.275576 -0.7072477  2.78488988
## 29    8.971966 -0.7072477  1.67405235
## 30   13.671043 -0.7072477  2.34949852
## 31   10.681805 -0.7072477 -0.34874345
## 32   11.616017 -0.7072477  3.97324701
## 33    3.855022 -0.7072477 -0.78159841
## 34   12.077913 -0.7072477  0.59297620
## 35   12.475521 -0.7072477  2.79087079
## 36    9.588522 -0.7072477  3.30583001
## 37    8.953511 -0.7072477 -0.33906989
## 38   13.017607 -0.7072477  3.01771139
## 39    9.973714 -0.7072477  2.56213398
## 40   11.920207 -0.7072477  1.73389718
## 41   11.615322 -0.7072477  0.72286994
## 42    9.110030 -0.7072477  0.95471822
## 43    9.162881 -0.7072477  2.31473464
## 44    7.124344 -0.7072477  3.81320290
## 45    5.658220 -0.7072477 -0.47815834
## 46   10.409998 -0.7072477  0.08257603
## 47   10.022095 -0.7072477  4.64032775
## 48    8.715131 -0.7072477  0.30349660
## 49   11.063400 -0.7072477 -1.81262252
## 50   10.817344 -0.7072477  1.80003820
##
## attr(,"class")
## [1] "coef.mer"
```

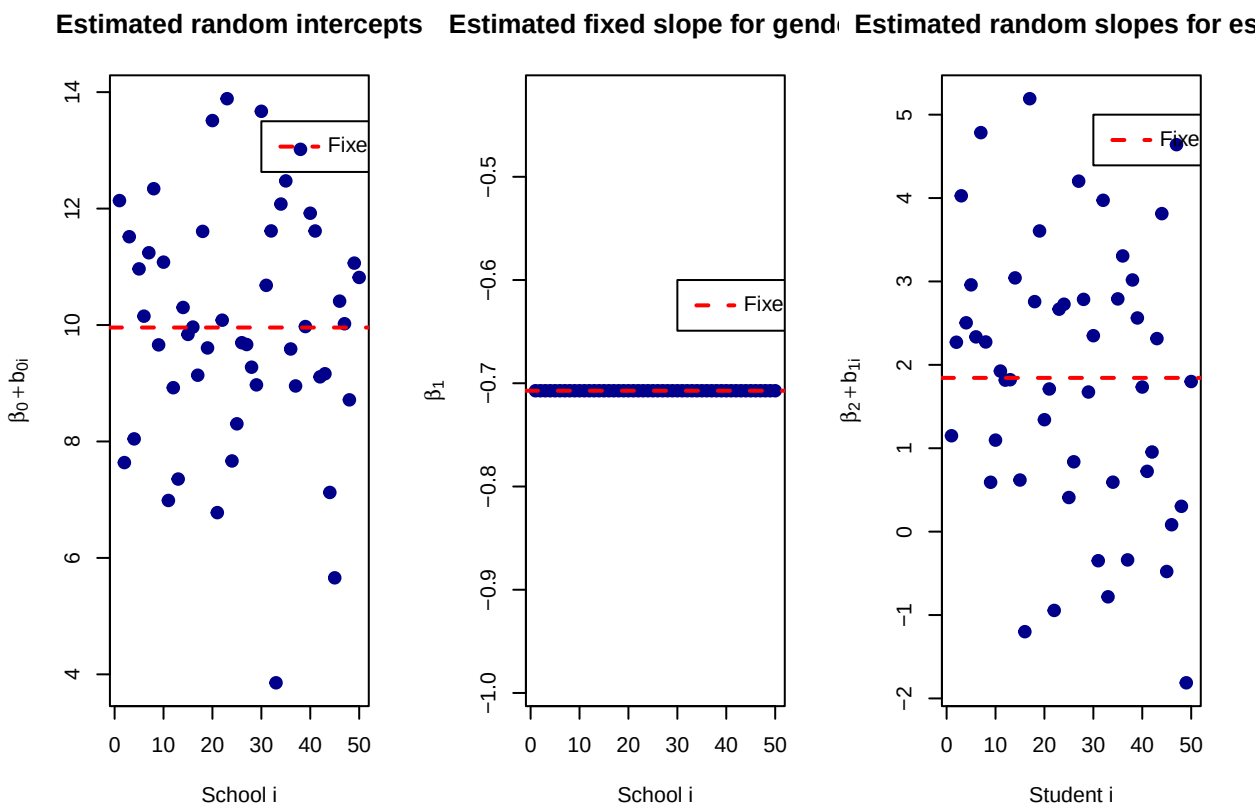
```
head(coef(lmm2)$school)
```

```
##   (Intercept)    gender1     escs
## 1   12.137262 -0.7072477 1.149662
## 2    7.637399 -0.7072477 2.271045
## 3   11.516830 -0.7072477 4.027023
## 4    8.044283 -0.7072477 2.504224
## 5   10.964221 -0.7072477 2.959170
## 6   10.152071 -0.7072477 2.335388
```

```
## Visualization of random effects

par(mfrow=c(1,3))
plot(c(1:50), unlist(coef(lmm2)$school_id[1]),
     xlab='School i', ylab=expression(beta[0]+b['0i']),
     pch=19, lwd=2, col='darkblue',
     main='Estimated random intercepts')
abline(h=fixef(lmm2)[1], lty=2, col='red', lwd=2)
legend(30, 13.5, legend=expression(paste('Fixed intercept ',beta[0])), lwd=2, lty=2,
       col='red', x.intersp=0.5)

plot(c(1:50), unlist(coef(lmm2)$school_id[2]),
```

```
     xlab='School i', ylab=expression(beta[1]),
     pch=19, lwd=2, col='darkblue',
     main='Estimated fixed slope for gender')
abline(h=fixef(lmm2)[2], lty=2, col='red', lwd=2)
legend(30,-0.6, legend=expression(paste('Fixed slope ',beta[1])), lwd=2, lty=2,
       col='red', x.intersp=0.5)

plot(c(1:50), unlist(coef(lmm2)$school_id[3]),
     xlab='Student i', ylab=expression(beta[2]+b['1i']),
     pch=19, lwd=2, col='darkblue',
     main='Estimated random slopes for escs')
abline(h=fixef(lmm2)[3], lty=2, col='red', lwd=2)
legend(30, 5, legend=expression(paste('Fixed slope ',beta[2])), lwd=2, lty=2,
       col='red', x.intersp=0.5)
```

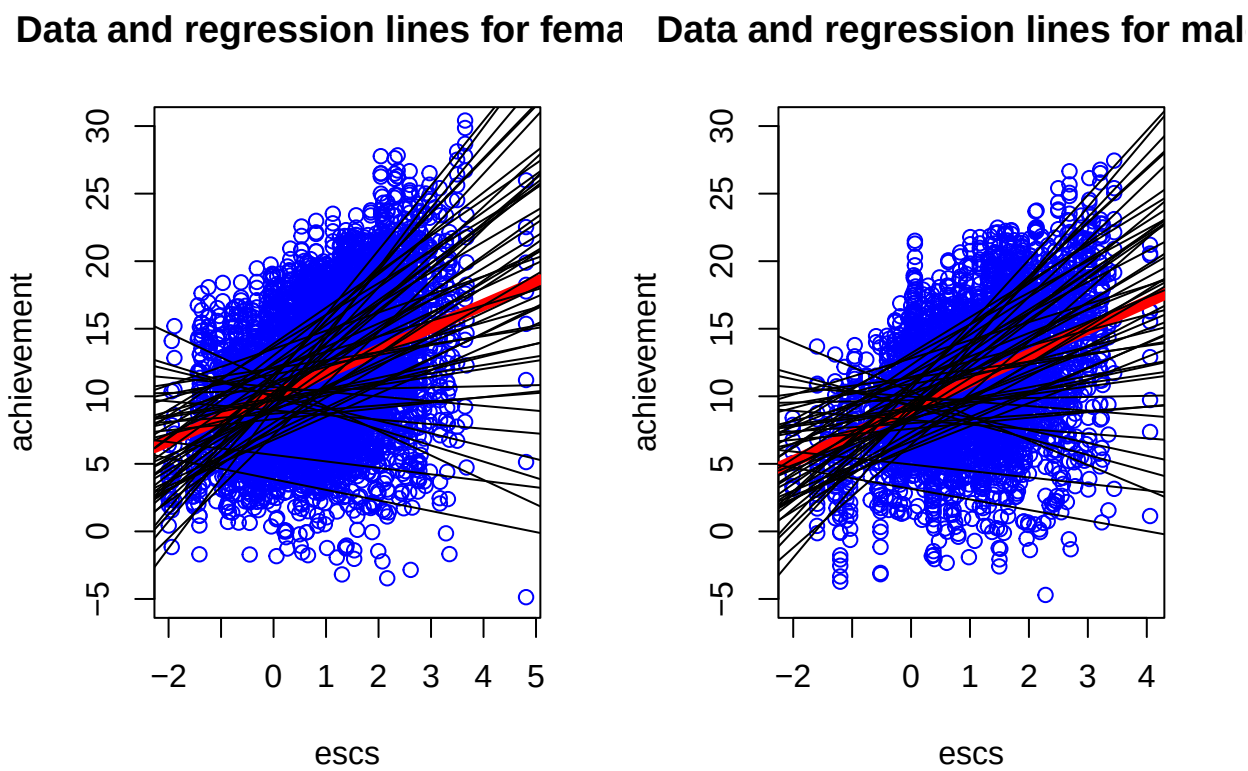
11.3.3.3 Visualization of random effects




```
# Lines Visualization
#---------------------

# Let's plot all the regression lines
#! we estimate a regression line with different intercept for each school, and
#! also different slope according to escs
#
#! Anyway gender should be a fixed effect, not a random one as is not really a "grouping"
#! Also we would have just 2 groups, while rand effect works well on more groups
#
## FEMALES

par(mfrow=c(1,2))
plot(school$escs[school$gender==0], school$achiev[school$gender==0],col='blue',
     xlab='escs', ylab='achievement',ylim=c(-5,30),
     main='Data and regression lines for females')
abline(10.0546535,1.6790886, col='red', lw=6)

for(i in 1:50){
```

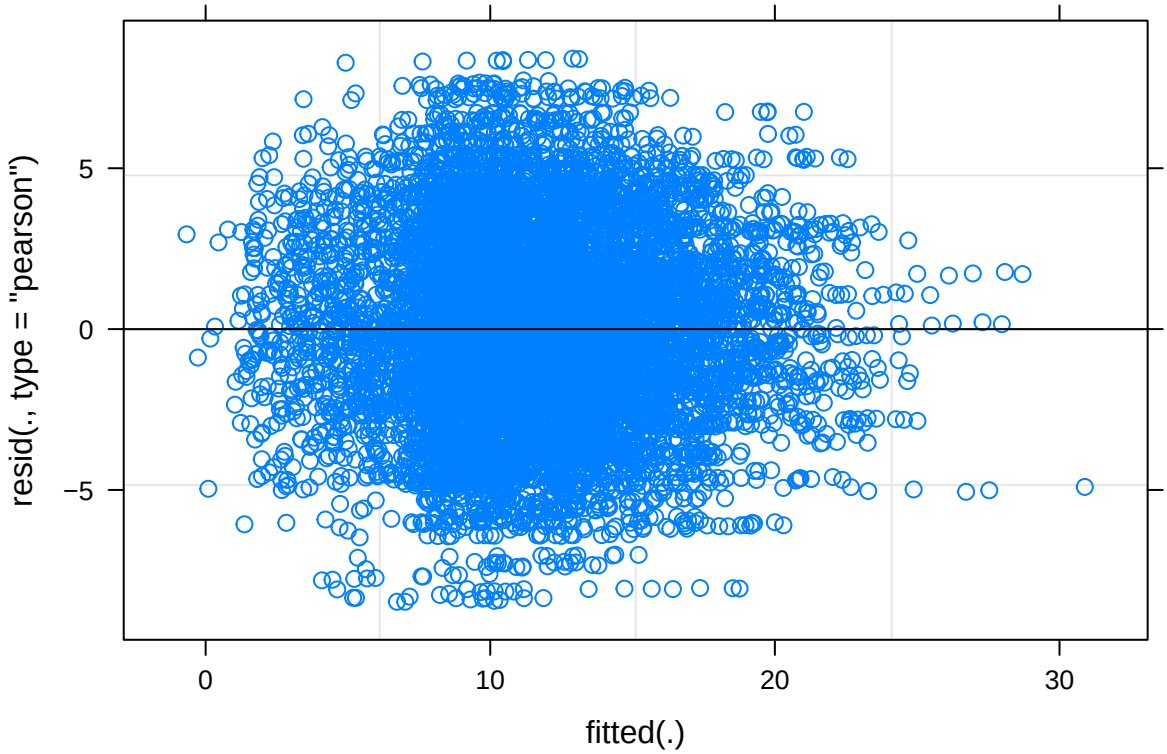
```
  abline(coef(lmm2)$school_id[i,1], coef(lmm2)$school_id[i,3])
}

## MALES
plot(school$escs[school$gender==1], school$achiev[school$gender==1],col='blue',
     xlab='escs', ylab='achievement',ylim=c(-5,30),
     main='Data and regression lines for males')
abline(10.02507-0.91180,1.96618, col='red', lw=6)

for(i in 1:50){
  abline(coef(lmm2)$school_id[i,1] + coef(lmm2)$school_id[i,2], coef(lmm2)$school_id[i,3])
}
```

11.3.3.4 Lines Visualization

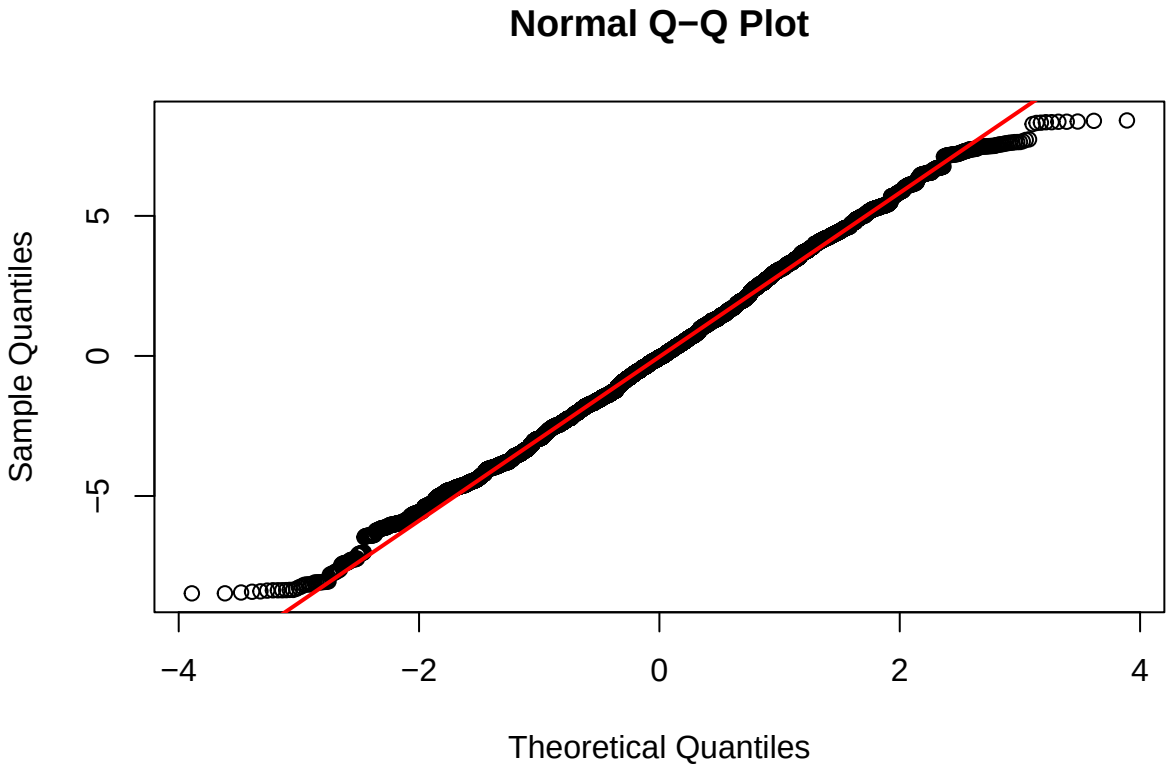



```
# Diagnostic plots
#--------------------
# 1) Assessing Assumption on the within-group errors

plot(lmm2)
```

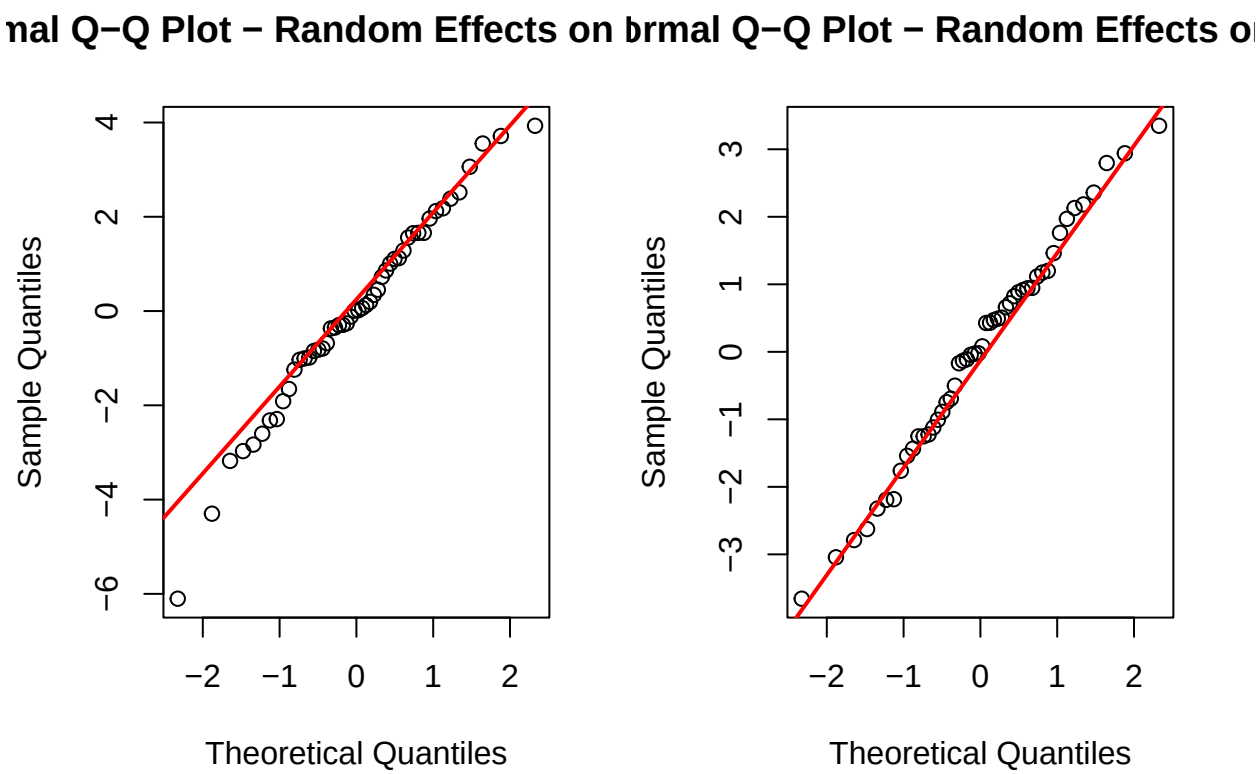
11.3.3.5 Diagnostic plots




```
qqnorm(resid(lmm2))
qqline(resid(lmm2), col='red', lwd=2)
```

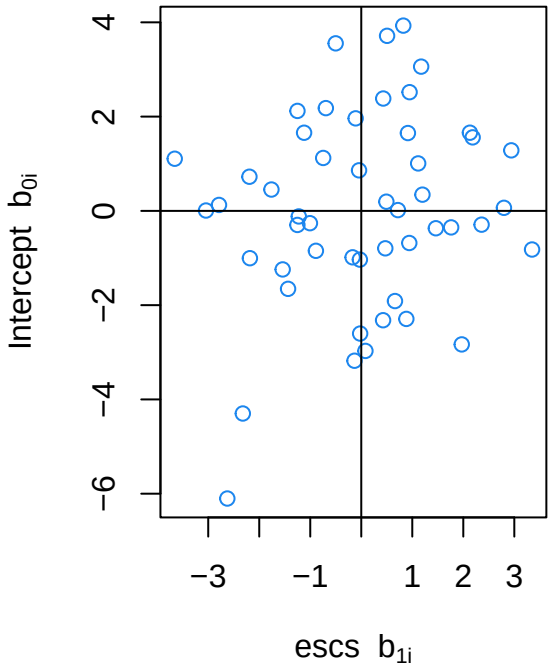


```r
# 2) Assessing Assumption on the Random Effects

par(mfrow=c(1,2))
qqnorm(unlist(ranef(lmm2)$school_id[1]), main='Normal Q-Q Plot - Random Effects on Intercept')
qqline(unlist(ranef(lmm2)$school_id[1]), col='red', lwd=2)
qqnorm(unlist(ranef(lmm2)$school_id[2]), main='Normal Q-Q Plot - Random Effects on escs')
qqline(unlist(ranef(lmm2)$school_id[2]), col='red', lwd=2)
```




```r
plot(unlist(ranef(lmm2)$school_id[2]),unlist(ranef(lmm2)$school_id[1]),
     ylab=expression(paste('Intercept  ', b['0i'])),
     xlab=expression(paste('escs  ', b['1i'])), col='dodgerblue2',
     main='Scatterplot of estimated random effects')
abline(v=0,h=0)
# Alternative plot(ranef(lmm2))


# Comparing models
#------------------
# The anova function, when given two or more arguments representing fitted models,
# produces likelihood ratio tests comparing the models.
anova(lmm1, lmm2)
```

```
## refitting model(s) with ML (instead of REML)

## Data: school
## Models:
## lmm1: achiev ~ gender + escs + (1 | school_id)
## lmm2: achiev ~ gender + escs + (1 + escs | school_id)
##      npar   AIC   BIC logLik deviance  Chisq Df Pr(>Chisq)
## lmm1    5 52890 52926 -26440    52880
## lmm2    7 50238 50288 -25112    50224 2656.2  2  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# The p-value for the test is essentially zero -> we prefer lmm2
```

Scatterplot of estimated random eff