Lab2 Block1 - Machine Learning

Thijs Quast

6-12-2018

Contents

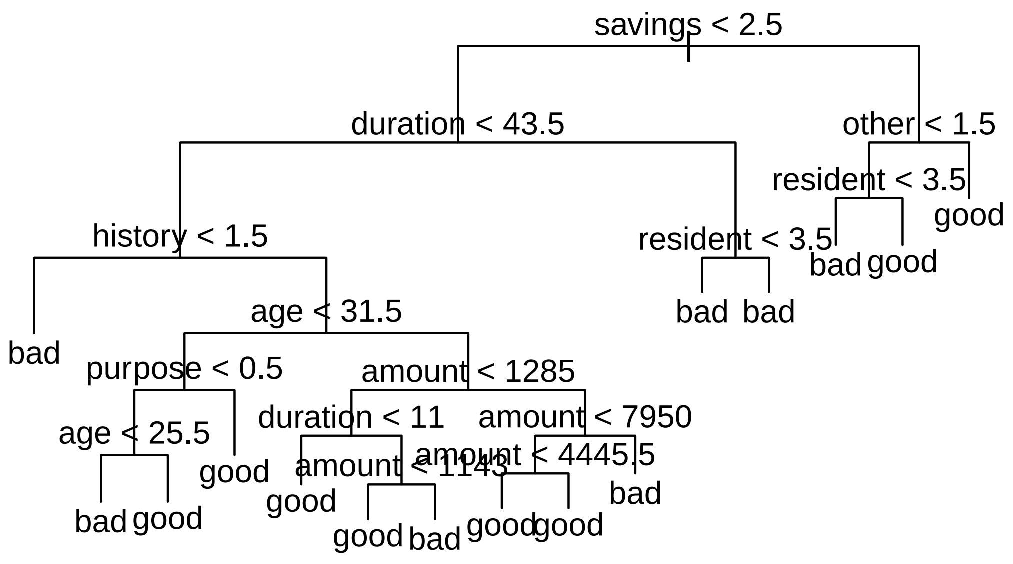
Assignment 2 - Analysis of credit scoring

2.1

```r
library(readxl)
data <- read_excel("creditscoring.xls")

n <- dim(data)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- data[id,]

id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n*0.25))
valid <- data[id2,]

id3 <- setdiff(id1, id2)
test <- data[id3,]
```

2.2

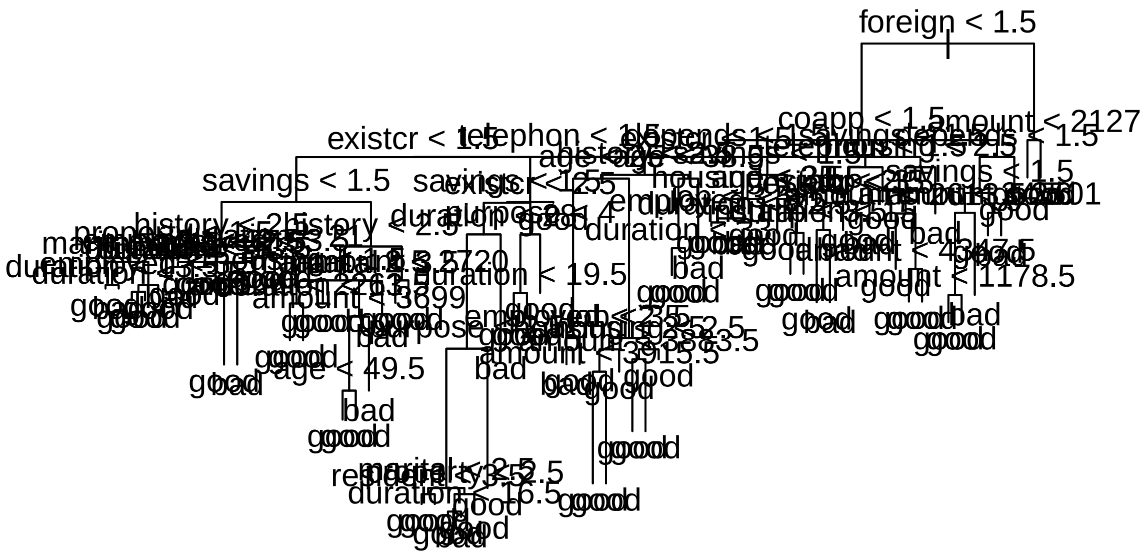
```r
# Target is classification, so a classifiation tree
library(tree)
# Question 2.2 ####
# Transform target variables into factors
train$good_bad <- as.factor(train$good_bad)
test$good_bad <- as.factor(test$good_bad)
valid$good_bad <- as.factor(valid$good_bad)

# Fit decision tree with measure: 'deviance'
fit_dev <- tree(good_bad ~ ., data = train, split = "deviance")
plot(fit_dev)
text(fit_dev, pretty = 0)
```



```r
# predict on train
pred_fit_dev_train <- predict(fit_dev, newdata = train, type = c("class"))
confusion_train_dev <- table(train$good_bad, pred_fit_dev_train)
miss_class_train_dev <- (confusion_train_dev[1,2] + confusion_train_dev[2,1])/nrow(train)

# predict on test
pred_fit_dev_test <- predict(fit_dev, newdata = test, type = c("class"))
confusion_test_dev <- table(test$good_bad, pred_fit_dev_test)
miss_class_test_dev <- (confusion_test_dev[1,2] + confusion_test_dev[2,1])/nrow(test)
summary(fit_dev)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "deviance")
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"  "age"      "purpose"  "amount"
## [7] "resident" "other"
## Number of terminal nodes:  15
## Residual mean deviance:  0.9569 = 458.3 / 479
## Misclassification error rate: 0.2105 = 104 / 494
```

```r
# Fit decision tree with measure: 'gini'
fit_gini <- tree(good_bad ~ ., data = train, split = "gini")
plot(fit_gini)
text(fit_gini, pretty = 0)
```



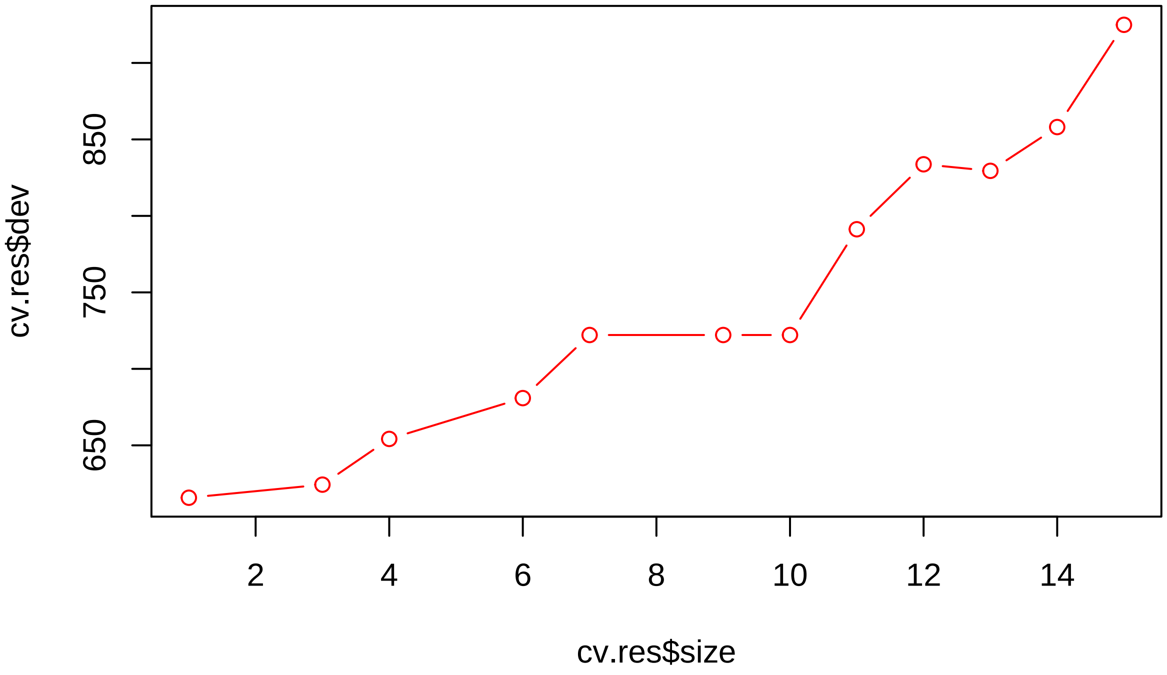
```r
# predict on train
pred_fit_gini_train <- predict(fit_gini, newdata = train, type = c("class"))
confusion_train_gini <- table(train$good_bad, pred_fit_gini_train)
miss_class_train_gini <- (confusion_train_gini[1,2] + confusion_train_gini[2,1])/nrow(train)

# predict on test
pred_fit_gini_test <- predict(fit_gini, newdata = test, type = c("class"))
confusion_test_gini <- table(test$good_bad, pred_fit_gini_test)
miss_class_test_gini <- (confusion_test_gini[1,2] + confusion_test_gini[2,1])/nrow(test)

results <- data.frame(miss_class_train_dev, miss_class_test_dev, miss_class_train_gini, miss_class_test
results
```

```
##   miss_class_train_dev miss_class_test_dev miss_class_train_gini
## 1                0.212               0.268                 0.238
##   miss_class_test_gini
## 1                0.372
```

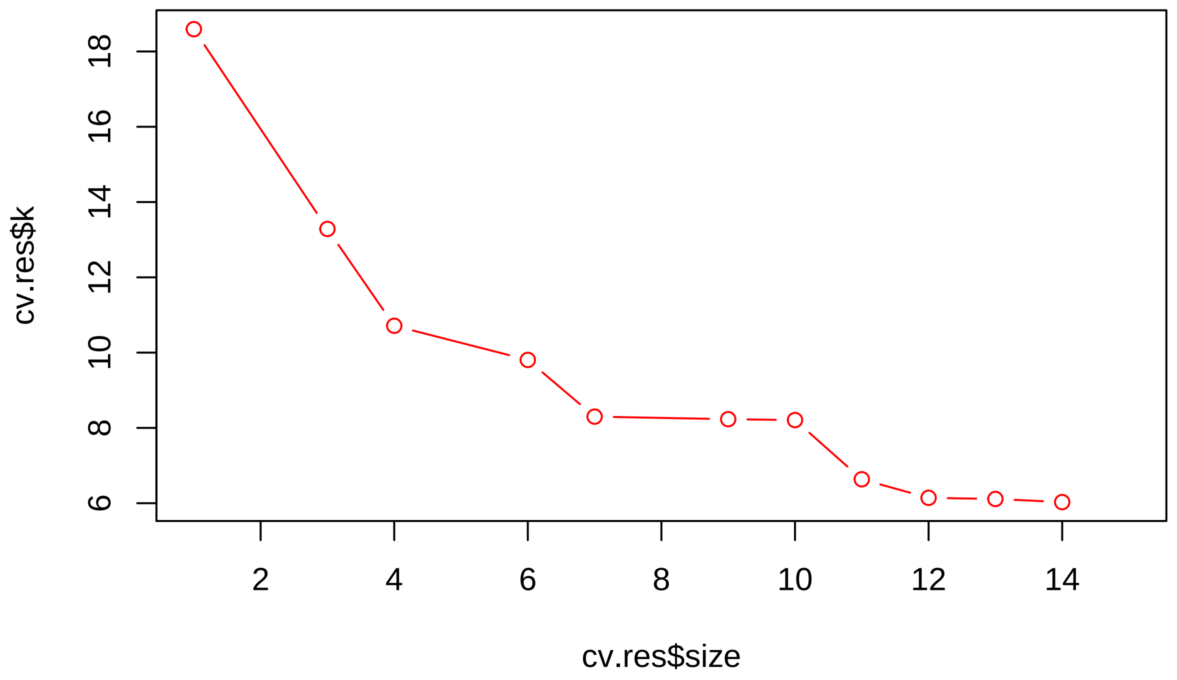
Results are the best for a classification tree fitted according to 'deviance'. Lowest missclassificatino error on the test set.

2.3

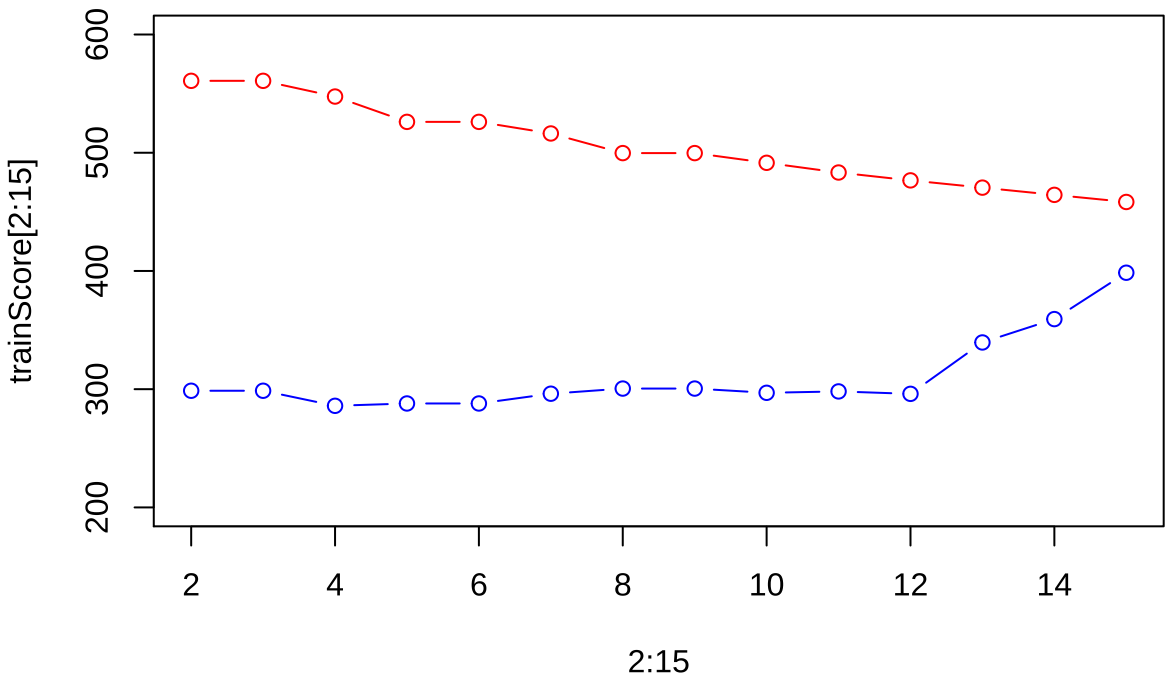
```
fit_dev <- tree(good_bad ~ ., data = train, split = "deviance")
cv.res <- cv.tree(fit_dev)
plot(cv.res$size, cv.res$dev, type = "b", col = "red")
```



```
plot(cv.res$size, cv.res$k, type = "b", col = "red")
```



```r
trainScore <- rep(0,15)
testScore <- rep(0,15)
for(i in 2:15) {
  prunedTree <- prune.tree(fit_dev,best=i)
  pred <- predict(prunedTree, newdata=valid,
type="tree")
  trainScore[i] <- deviance(prunedTree)
  testScore[i] <- deviance(pred)
}

plot(2:15, trainScore[2:15], type="b", col="red", ylim = c(200, 600))
points(2:15, testScore[2:15], type="b", col="blue")
```



```
optimal_tree <- prune.tree(fit_dev, best = 4)
optimal_pred <- predict(optimal_tree, newdata = test, type = "class")
class_table <- table(test$good_bad, optimal_pred)
missclassification_rate <- (class_table[1,2] + class_table[2,1])/nrow(test)
```

The optimal tree has 4 leaves, as this results in the lowest error on the test data. When analyzing this tree, it thus has 4 leaves. The tree uses the variables: savings, duration and history. Splits are at the values 2.5, 43.5 and 1.5 respectively. The misclassification rate on the test data is 25.6%.

2.4

```
library(MASS)
library(e1071)
fit_bayes <- naiveBayes(good_bad ~., data = train)

predict_bayes_train <- predict(fit_bayes, newdata = train)
confusion_bayes_train <- table(train$good_bad, predict_bayes_train)
miss_class_train_bayes <- (confusion_bayes_train[1,2] + confusion_bayes_train[2,1])/nrow(train)


predict_bayes_test <- predict(fit_bayes, newdata = test)
confusion_bayes_test <- table(test$good_bad, predict_bayes_test)
miss_class_test_bayes <- (confusion_bayes_test[1,2] + confusion_bayes_test[2,1])/nrow(test)

df_bayes <- data.frame(miss_class_train_bayes, miss_class_test_bayes)

confusion_bayes_train
```

```
##       predict_bayes_train
##         bad good
##   bad    95   52
##   good   98  255
```

```
confusion_bayes_test
```

```
##       predict_bayes_test
##        bad good
##   bad   46   30
##   good  49  125
```

```
df_bayes
```

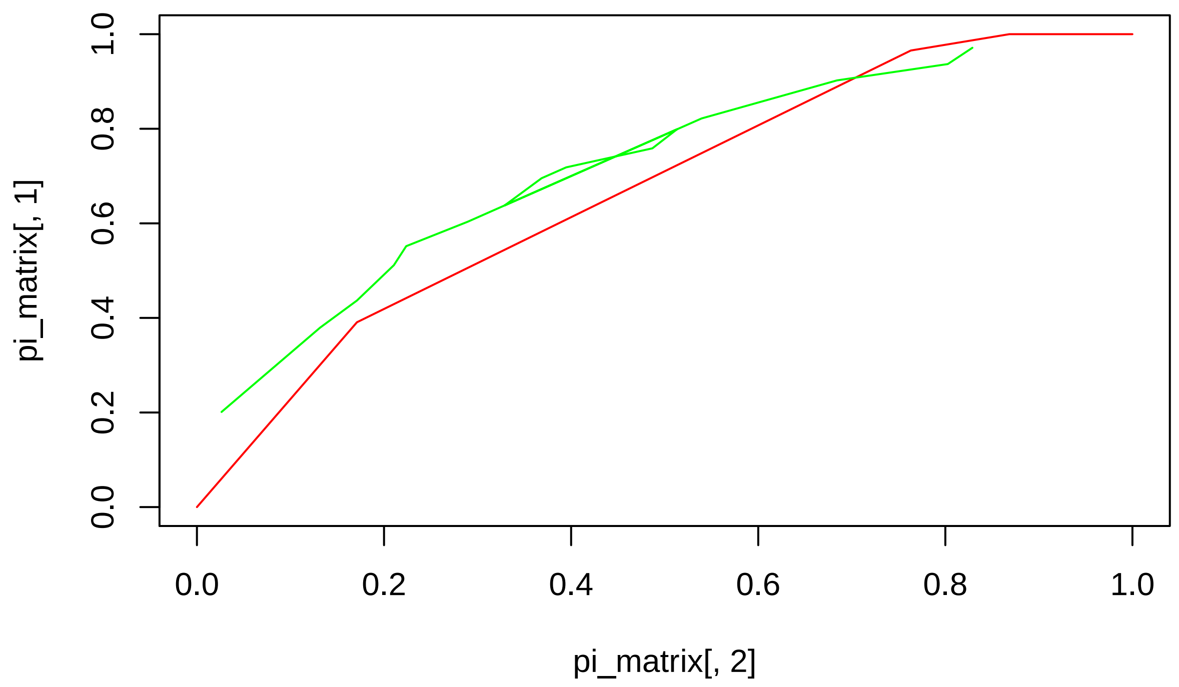
```
##   miss_class_train_bayes miss_class_test_bayes
## 1                    0.3                 0.316
```

The naive bayes classifier performs worse on the test data.

2.5

```
final_tree <- prune.tree(fit_dev, best = 4)
pred_final <- predict(final_tree, newdata = test, type = c("vector"))
pred_final <- pred_final[,2]
pi <- seq(from = 0.05, to=0.95, by=0.05)


pi_matrix <- matrix(0, nrow = 19, ncol = 2)

for (i in pi[1:3]){
  pi_value <- pred_final
  pi_value[pi_value > i] <- "good"
  pi_value[pi_value <= i] <- "bad"
  pi_value <- as.factor(pi_value)
  pi_result <- table(test$good_bad, pi_value)
  tpr <- pi_result[2,1]
  fpr <- pi_result[1,1]
  pi_matrix[i/0.05, 1] <- 1
  pi_matrix[i/0.05, 2] <- 1
}

for (i in pi[4:16]){
  pi_value <- pred_final
  pi_value[pi_value > i] <- "good"
  pi_value[pi_value <= i] <- "bad"
  pi_value <- as.factor(pi_value)
  pi_result <- table(test$good_bad, pi_value)
  tpr <- pi_result[2,2]/ (pi_result[2,2] + pi_result[2,1])
  fpr <- pi_result[1,2]/ (pi_result[1,2] + pi_result[1,1])
  pi_matrix[i/0.05, 1] <- tpr
  pi_matrix[i/0.05, 2] <- fpr
}

pi_0.30 <- pred_final
pi_0.30[pi_0.30 > 0.30] <- "good"
pi_0.30[pi_0.30 <= 0.30] <- "bad"
pi_conf_0.30 <-table(test$good_bad, pi_0.30)
tpr <- pi_conf_0.30[2,2]/ (pi_result[2,2] + pi_result[2,1])
fpr <- pi_conf_0.30[1,2]/ (pi_result[1,2] + pi_result[1,1])
```

```r
pi_matrix[6, 1] <- tpr
pi_matrix[6, 2] <- fpr


pi_df <- data.frame(pi_matrix)
colnames(pi_df) <- c("tpr", "fpr")
rownames(pi_df) <- pi

bayes_matrix <- matrix(0, nrow = 19, ncol = 2)
predict_bayes_test2 <- predict(fit_bayes, newdata = test, type = "raw")

for (i in pi[1:19]){
  pi_value <- predict_bayes_test2[,2]
  pi_value <- round(pi_value, digits = 3)
  pi_value[pi_value > i] <- "good"
  pi_value[pi_value <= i] <- "bad"
  pi_value <- as.factor(pi_value)
  pi_result <- table(test$good_bad, pi_value)
  tpr <- pi_result[2,2]/ (pi_result[2,2] + pi_result[2,1])
  fpr <- pi_result[1,2]/ (pi_result[1,2] + pi_result[1,1])
  bayes_matrix[i/0.05, 1] <- tpr
  bayes_matrix[i/0.05, 2] <- fpr
}

bayes_0.95 <- predict_bayes_test2[,2]
bayes_0.95 <- round(bayes_0.95, digits = 3)
bayes_0.95[bayes_0.95 > 0.95] <- "good"
bayes_0.95[bayes_0.95 <= 0.95] <- "bad"
bayes_conf_0.95 <-table(test$good_bad, bayes_0.95)
tpr <- bayes_conf_0.95[2,2]/ (bayes_conf_0.95[2,2] + bayes_conf_0.95[2,1])
fpr <- bayes_conf_0.95[1,2]/ (bayes_conf_0.95[1,2] + bayes_conf_0.95[1,1])
bayes_matrix[19, 1] <- tpr
bayes_matrix[19, 2] <- fpr

bayes_0.60 <- predict_bayes_test2[,2]
bayes_0.60 <- round(bayes_0.60, digits = 3)
bayes_0.60[bayes_0.60 > 0.60] <- "good"
bayes_0.60[bayes_0.60 <= 0.60] <- "bad"
bayes_conf_0.60 <-table(test$good_bad, bayes_0.60)
tpr <- bayes_conf_0.60[2,2]/ (bayes_conf_0.60[2,2] + bayes_conf_0.60[2,1])
fpr <- bayes_conf_0.60[1,2]/ (bayes_conf_0.60[1,2] + bayes_conf_0.60[1,1])
bayes_matrix[6, 1] <- tpr
bayes_matrix[6, 2] <- fpr

plot(pi_matrix[,2], pi_matrix[,1],type="l",col="red")
lines(bayes_matrix[,2],  bayes_matrix[,1], col="green")
```



The ROC curve for the naive Bayesian classifier is slightly smaller. As we want the classifer with the greatest area under curve (AUC) this means that the decision tree is a more suitable model in this case.

2.6

```r
fit_bayes2 <- naiveBayes(good_bad ~., data = train)

predict_bayes_test2 <- predict(fit_bayes2, newdata = test)
confusion_bayes_test2 <- table(test$good_bad, predict_bayes_test2)
loss_matrix <- matrix(c(0, 10, 1, 0), nrow = 2, ncol = 2)
newconfusiontest <- confusion_bayes_test2 * loss_matrix
misstest <- (newconfusiontest[1,2] + newconfusiontest[2,1])/nrow(test)


predict_bayes_train2 <- predict(fit_bayes2, newdata = train)
confusion_bayes_train2 <- table(train$good_bad, predict_bayes_train2)
loss_matrix <- matrix(c(0, 10, 1, 0), nrow = 2, ncol = 2)
newconfusiontrain <- confusion_bayes_train2 * loss_matrix
misstrain <- (newconfusiontrain[1,2] + newconfusiontrain[2,1])/nrow(train)

misstest
```

```
## [1] 2.08
```

```r
misstrain
```

```
## [1] 2.064
```

Missclassifiation rates have drastically increased because very heavy weights are put on false negatives.

Assignment 3 - Uncertainty estimation

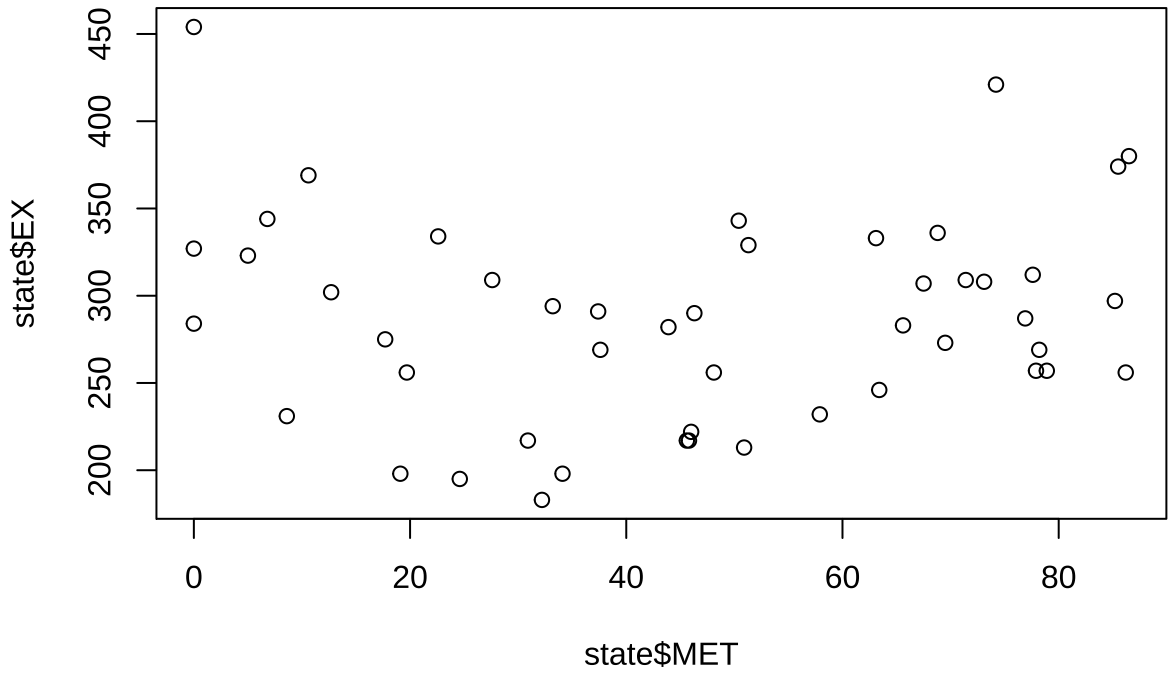
3.1

```
library(readr)
state <- read_csv2("State.csv")
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
## Parsed with column specification:
## cols(
##   EX = col_integer(),
##   ECAB = col_double(),
##   MET = col_double(),
##   GROW = col_double(),
##   YOUNG = col_double(),
##   OLD = col_double(),
##   WEST = col_integer(),
##   STATE = col_character()
## )
```

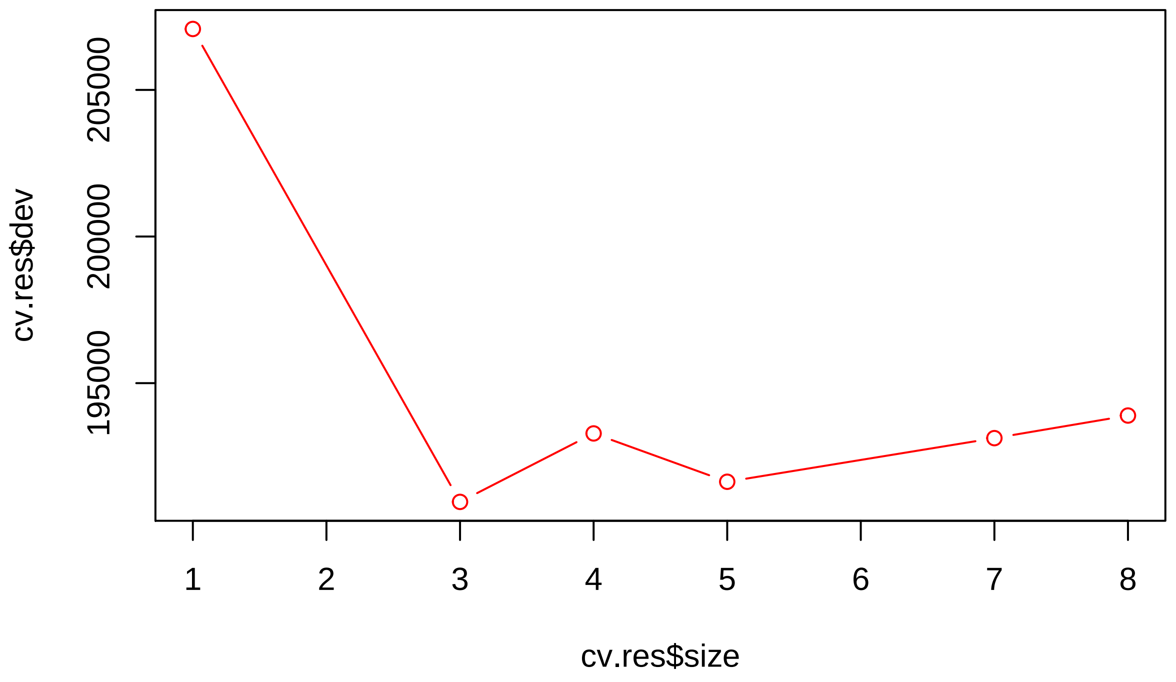
```
state <- as.data.frame(state)

state <- state[order(state$MET),]
plot(state$MET, state$EX)
```



There is no clear trend in the data, I would say it would be appropriate to fit this data with a quadratic model.

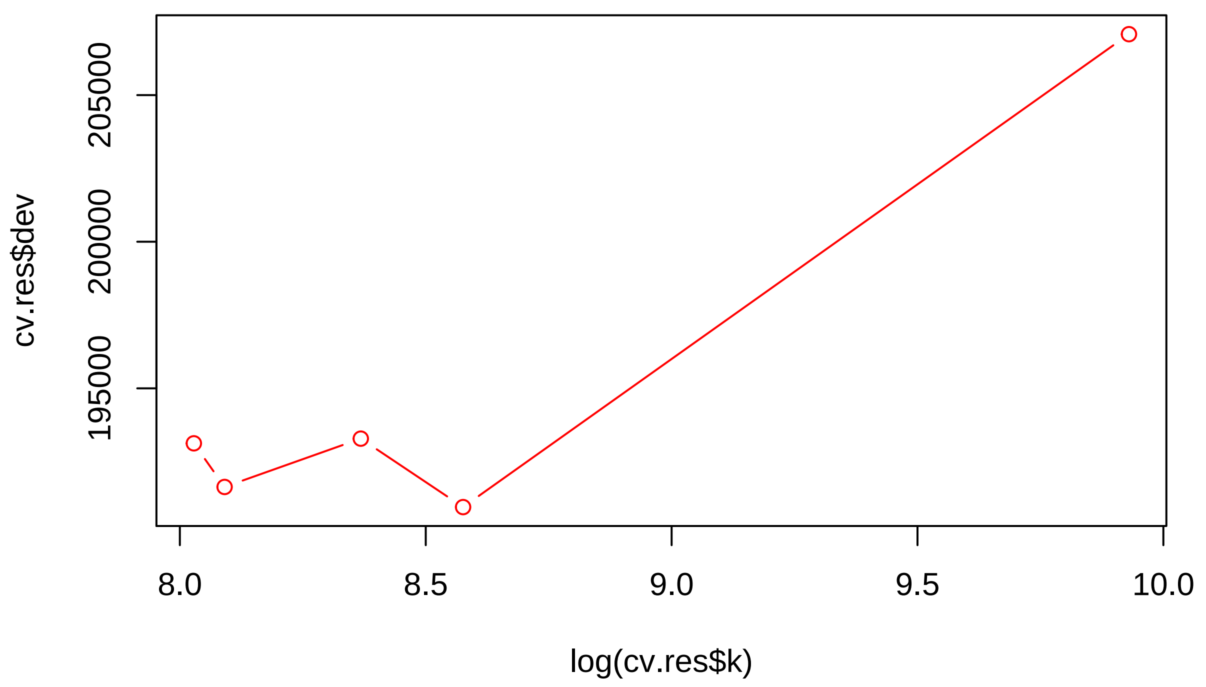
3.2

```r
library(tree)
set.seed(12345)
control <- tree.control(nobs = nrow(state),minsize = 8)
reg_tree <- tree(EX ~ MET, data = state, control = control)
cv.res <- cv.tree(reg_tree)

plot(cv.res$size, cv.res$dev, type="b", col="red")
```

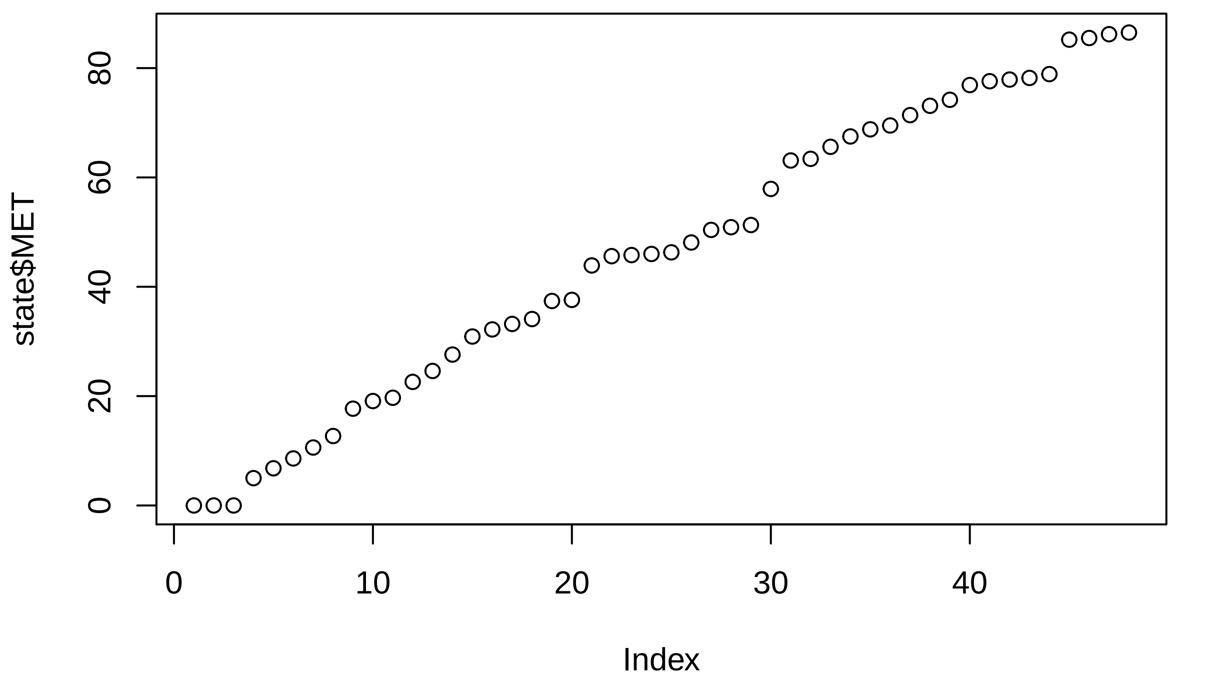


```r
plot(log(cv.res$k), cv.res$dev,
     type="b", col="red")
```

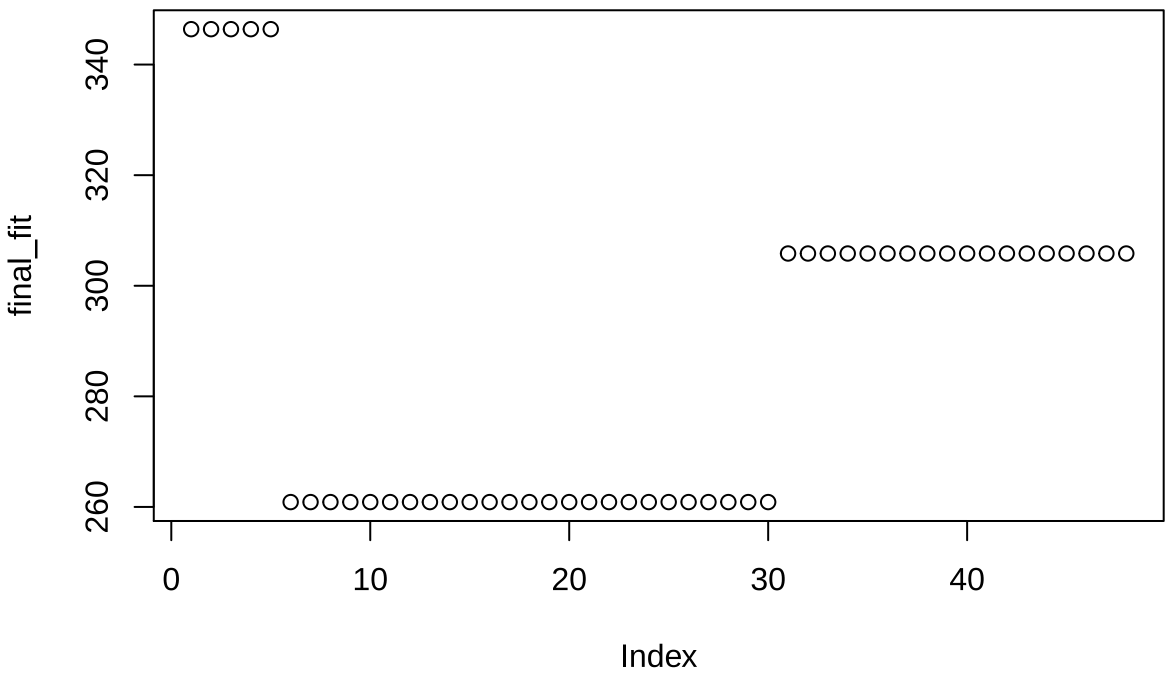
```
## Warning in log(cv.res$k): NaNs produced
```



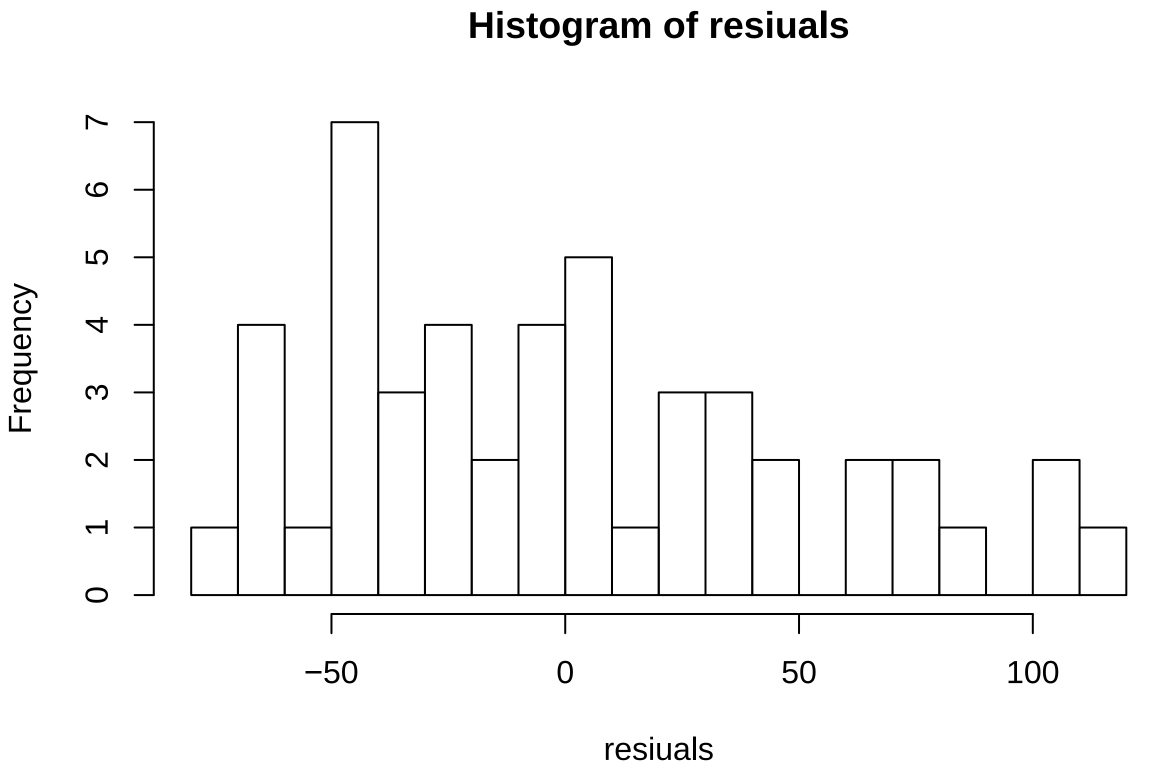
```
final <- prune.tree(reg_tree, best = 3)
final_fit <- predict(final, newdata = state)
# The optimal tree is with 3 leaves.

plot(state$MET)
```



```
plot(final_fit)
```



```
resiuals <- state$EX - final_fit
residuals_hist <-hist(resiuals, breaks = 20)
```



```
# The residuals are not normally distributed, therefore I would say the fit is not very well.
final_fit
```

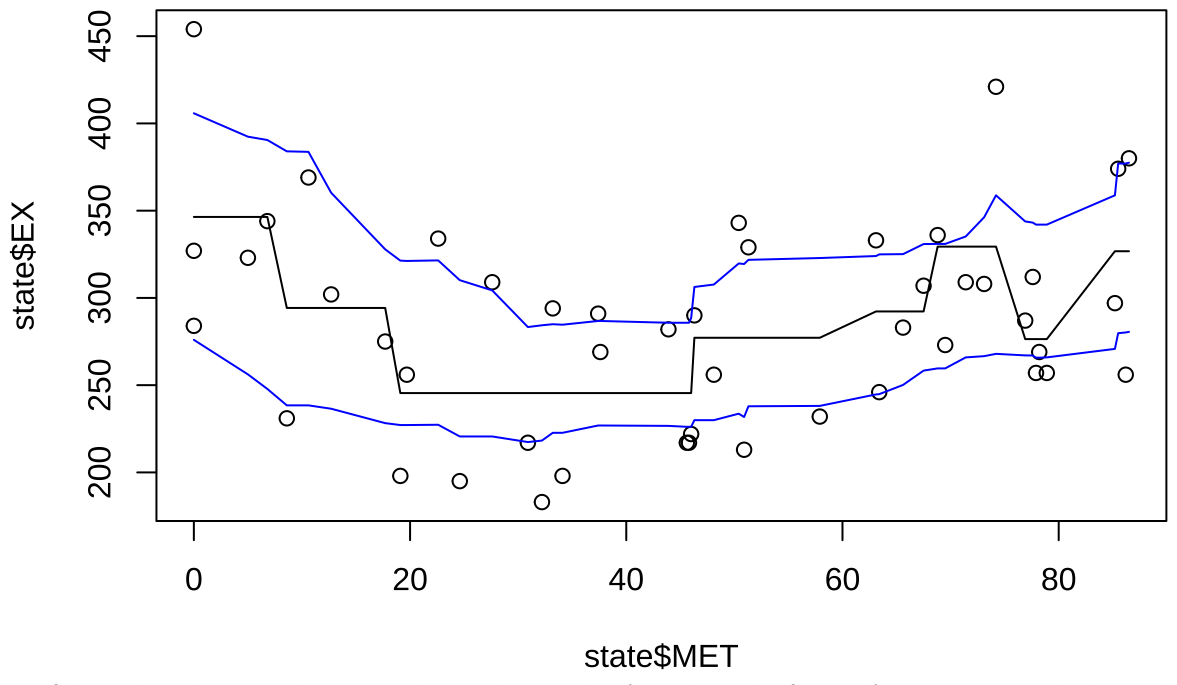
```
##        3        41        42        34        43        26        30        31
## 346.4000 346.4000 346.4000 346.4000 346.4000 260.8800 260.8800 260.8800
##        2        35         1        40        21        38        18        22
## 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800
##       28       19       33       32       36       25       20       23
## 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800
##       17       15       46       12       27       29       45       37
## 260.8800 260.8800 260.8800 260.8800 260.8800 260.8800 305.8333 305.8333
##       24       44       10       14       39       13       47       16
## 305.8333 305.8333 305.8333 305.8333 305.8333 305.8333 305.8333 305.8333
##        6        9       11        8        4        7        5       48
## 305.8333 305.8333 305.8333 305.8333 305.8333 305.8333 305.8333 305.8333
```

The optimal tree fits with 3 leaves. The residuals are not normally distributed, therefore I would say the fit is not very well.

```r
library(boot)
# computing bootstrap samples
bootstrap <- function(data, indices){
  data <- state[indices,]
  control_boot <- tree.control(nobs = nrow(data), minsize = 8)
  reg_tree_boot <- tree(EX ~ MET, data = data, control = control)
  final_boot <- prune.tree(reg_tree_boot, best = 3)
  final_fit_boot <- predict(final_boot, newdata = state)
  return(final_fit_boot)
}

res <- boot(state, bootstrap, R=1000) #make bootstrap
```

```
## Warning in prune.tree(reg_tree_boot, best = 3): best is bigger than tree
## size
```

```r
e <- envelope(res)
fit_for_ci <- tree(EX ~ MET, data = state, control = control)
predict_for_ci <- predict(fit_for_ci)

plot(state$MET, state$EX)
points(state$MET,predict_for_ci,type="l")
points(state$MET,e$point[2,], type="l", col="blue")
points(state$MET,e$point[1,], type="l", col="blue")
```



The confidence band is very bumpy. This is due to the fact that the fit itself is not a straight line, therefore confidence bounds around each part of the fitted line vary greatly, which results in a bumpy confidence band. The confidence band is rather wide, therefore perhaps the resutls from step 2 are not very reliable.

3.5

Assignment 4 - Principal components

4.1

```r
library(readr)
spectra <- read_csv2("NIRSpectra.csv")
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

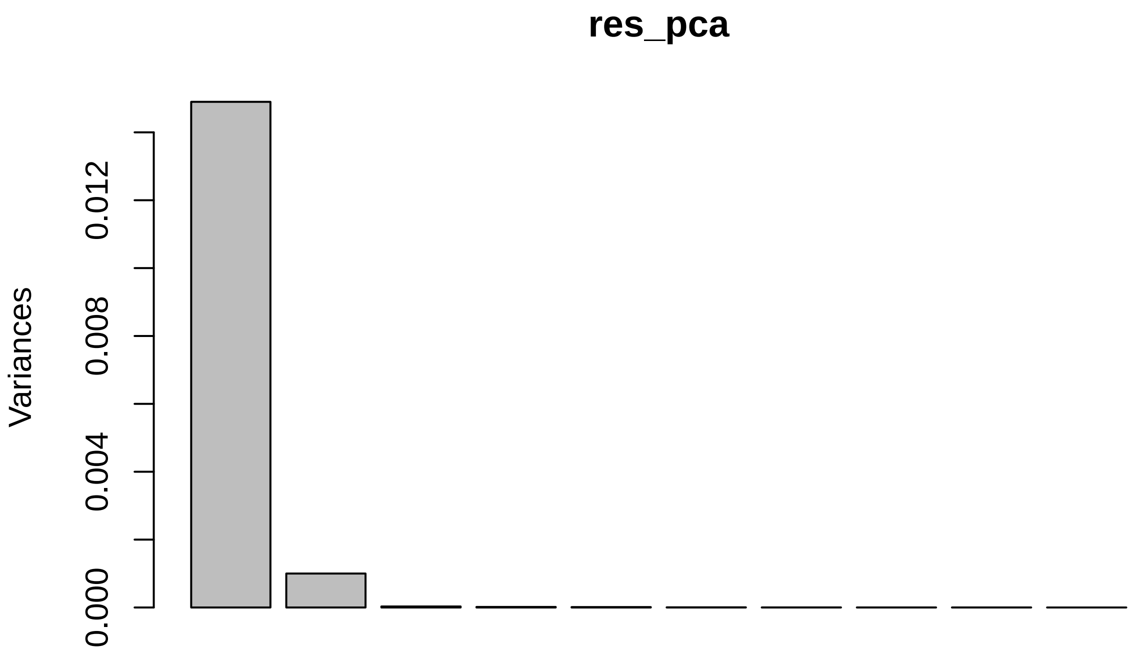
```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```r
set.seed(12345)
data_spectra <- spectra
data_spectra$Viscosity <- c()
res_pca <- prcomp(data_spectra)
lambda <- res_pca$sdev^2
sprintf("%2.3f",lambda/sum(lambda)*100)
```

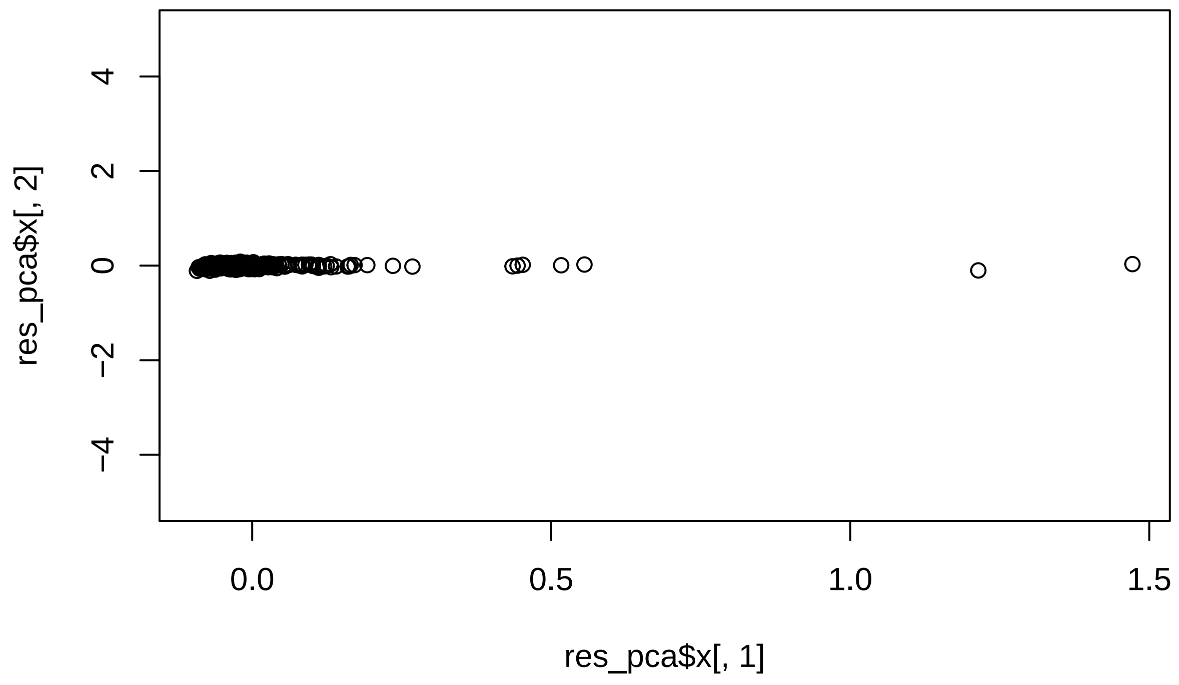
```
##   [1] "93.332" "6.263"  "0.185"  "0.101"  "0.068"  "0.025"  "0.009"
##   [8] "0.003"  "0.003"  "0.002"  "0.001"  "0.001"  "0.001"  "0.001"
##  [15] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [22] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [29] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [36] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [43] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [50] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [57] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [64] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [71] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [78] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [85] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [92] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [99] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [106] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [113] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [120] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
```

```
screeplot(res_pca)
```



The plot does not show exactly how many components should be extracted. If one however prints the results from the components, a total of 14 component capture 99.995% of the variance. The first two components together capture 99.60% of the variance.

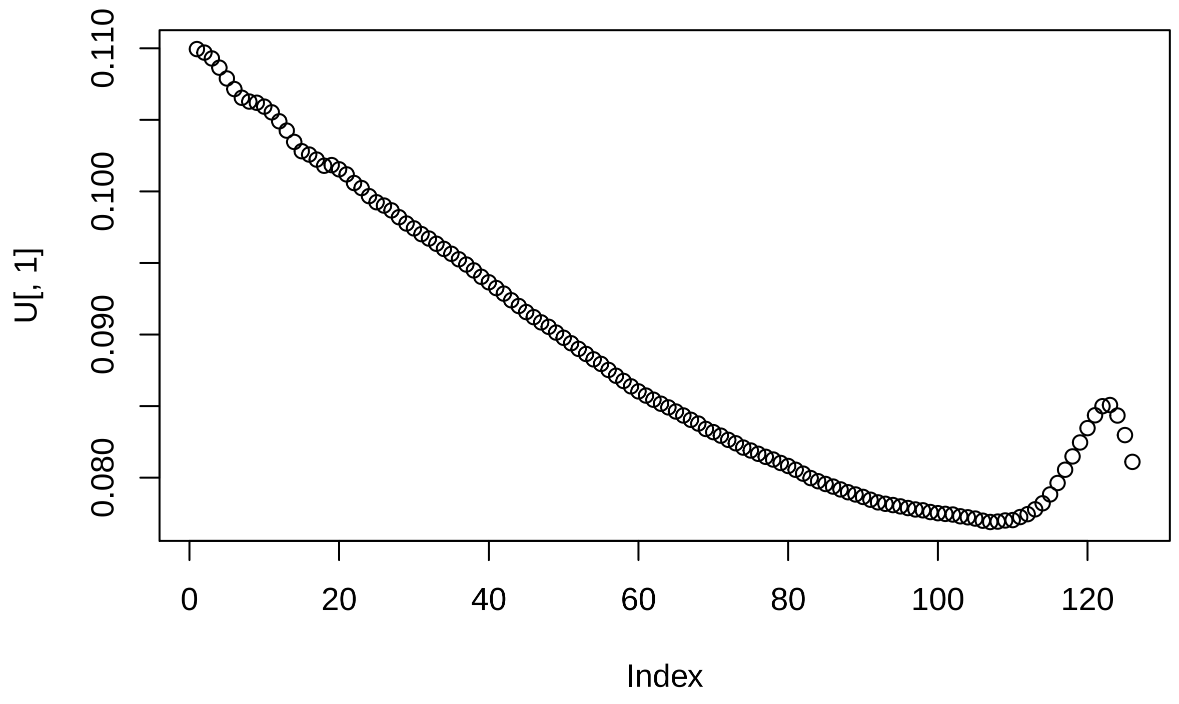
```
U <- res_pca$rotation
plot(res_pca$x[,1], res_pca$x[,2], ylim=c(-5,5))
```



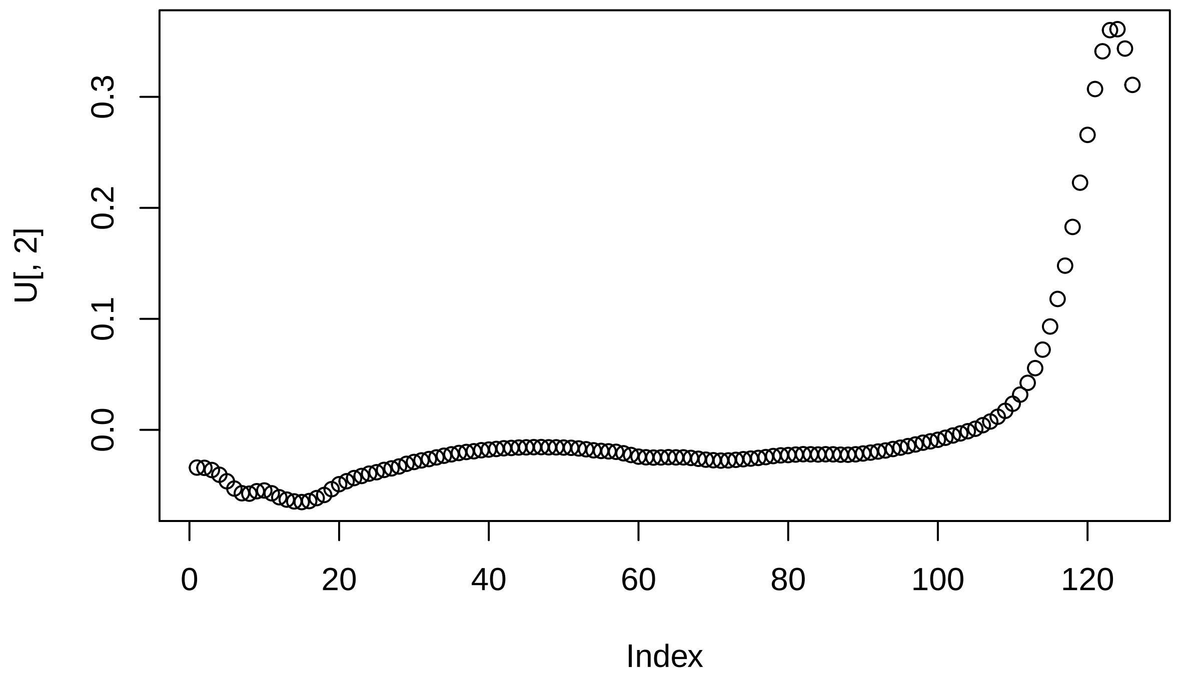
A few outliers in the plot imply there are some unusual diesel fuels.

4.2

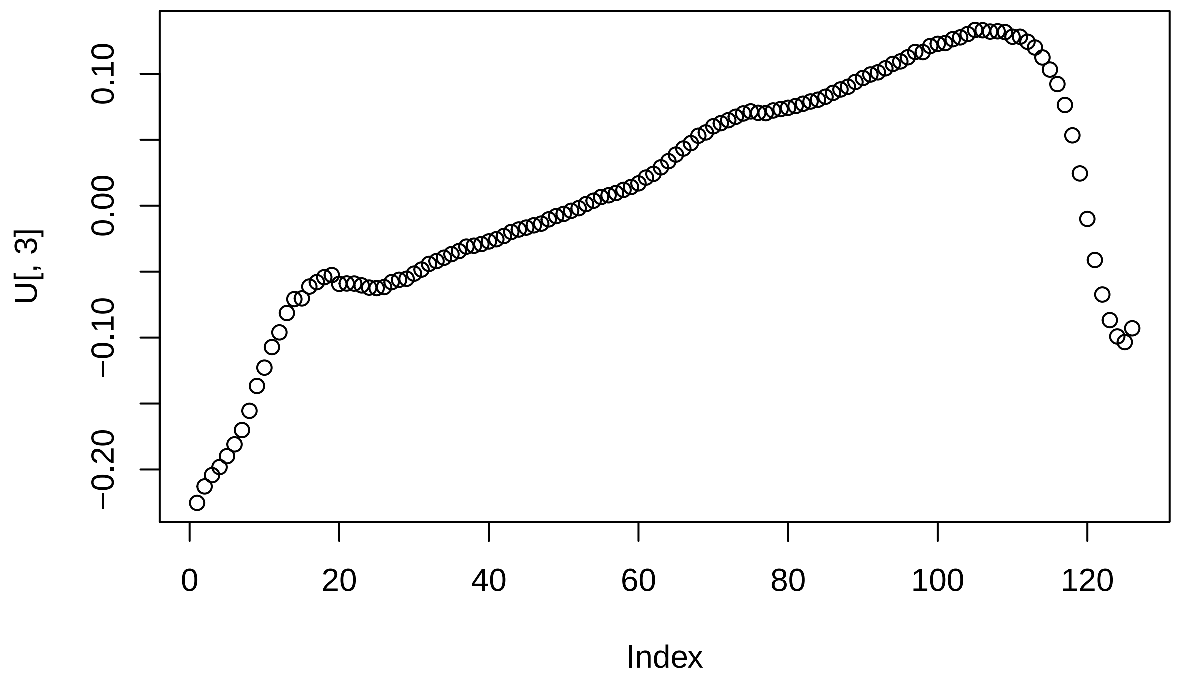
```r
plot(U[,1])
```



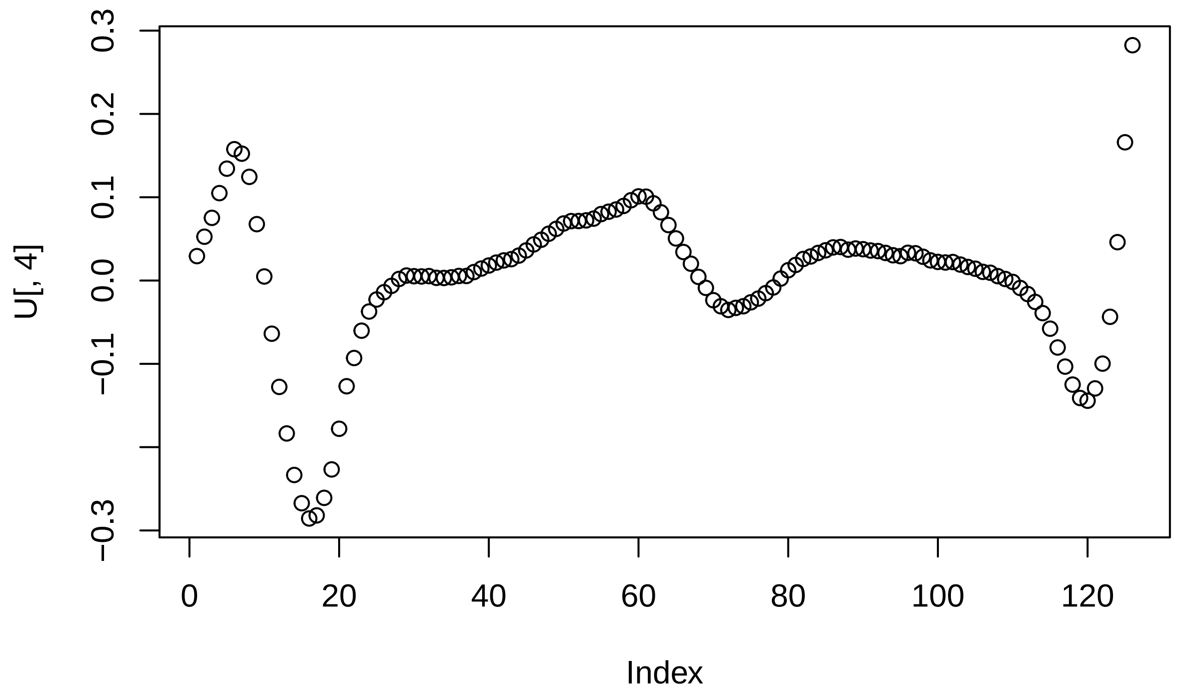
```r
plot(U[,2])
```



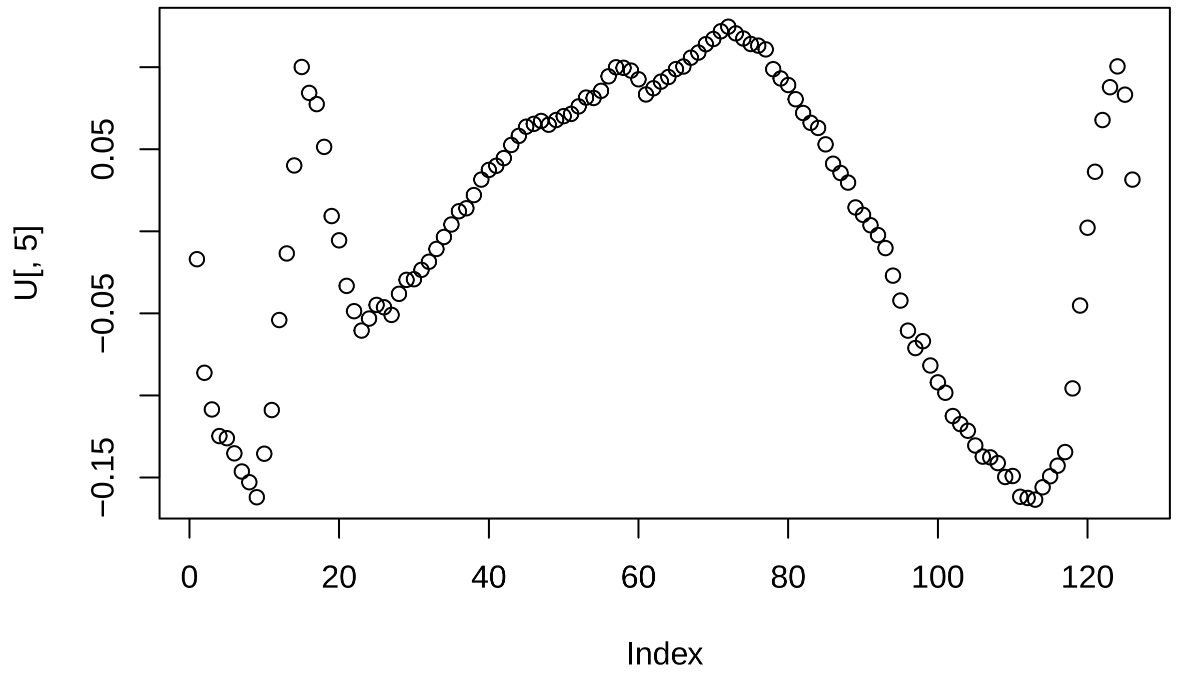
```
plot(U[,3])
```



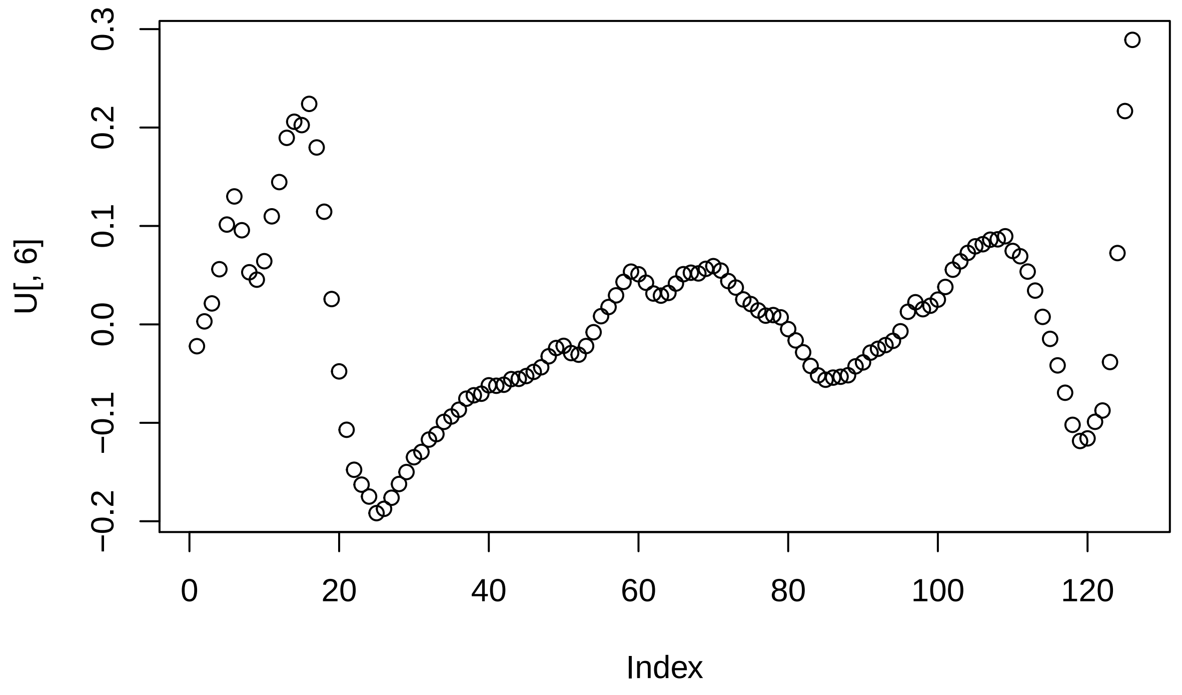
```
plot(U[,4])
```



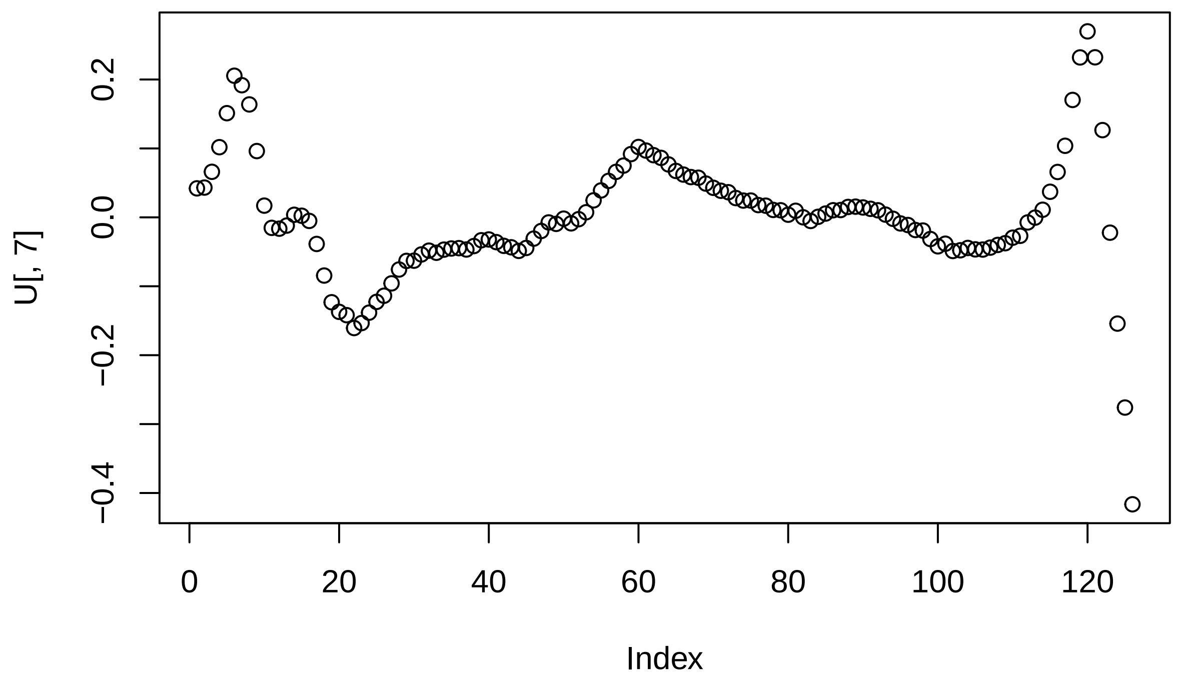
```r
plot(U[,5])
```



```r
plot(U[,6])
```



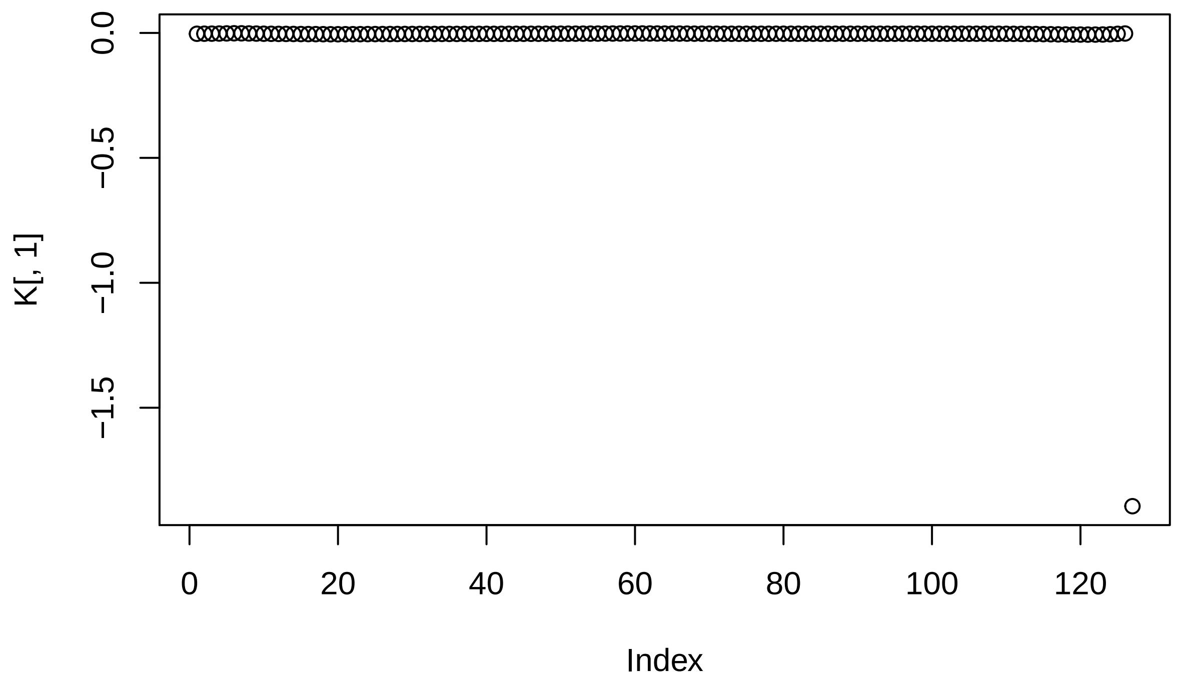
```
plot(U[,7])
```



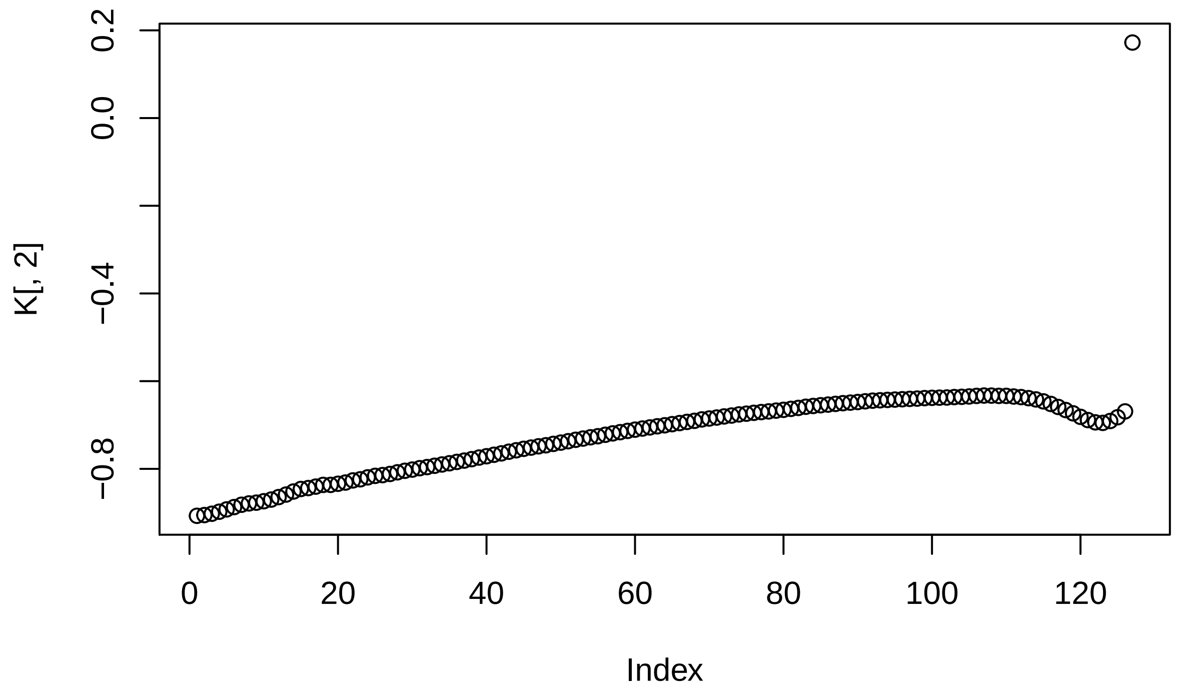
```
library(fastICA)
ica <- fastICA(spectra, 7, alg.typ = "parallel", fun = "logcosh", alpha = 1,
               method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001, verbose = TRUE)
```

```
## Centering

## Whitening

## Symmetric FastICA using logcosh approx. to neg-entropy function

## Iteration 1 tol = 0.9504831
## Iteration 2 tol = 0.7806835

## Iteration 3 tol = 0.6104179

## Iteration 4 tol = 0.3334712

## Iteration 5 tol = 0.1012066

## Iteration 6 tol = 0.006643659

## Iteration 7 tol = 0.00213235

## Iteration 8 tol = 0.0006705378

## Iteration 9 tol = 0.0001954602

## Iteration 10 tol = 5.64342e-05
```

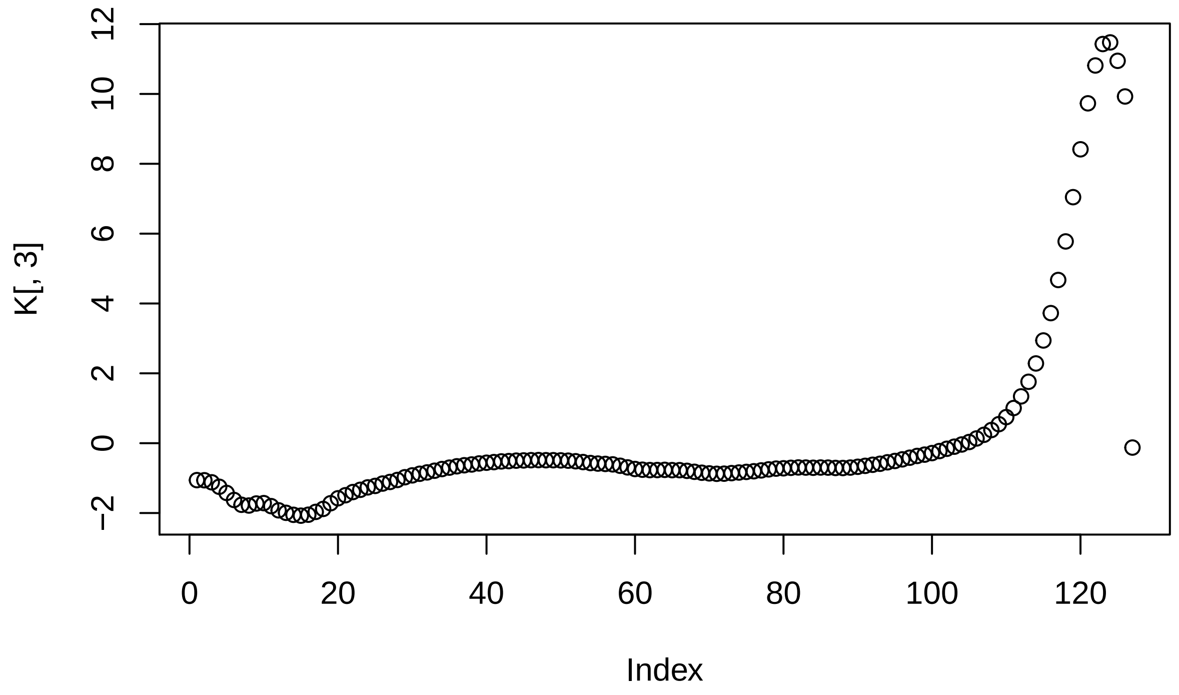
```r
#X  pre-processed data matrix
#K  pre-whitening matrix that projects data onto the first n.comp principal components.
#W  estimated un-mixing matrix (see definition in details)
#A  estimated mixing matrix
#S  estimated source matrix
# (source: R-help ?fastICA)

#?fastICA
X <- ica$X
K <- ica$K
W <- ica$W
A <- ica$A
S <- ica$S

plot(K[,1])
```



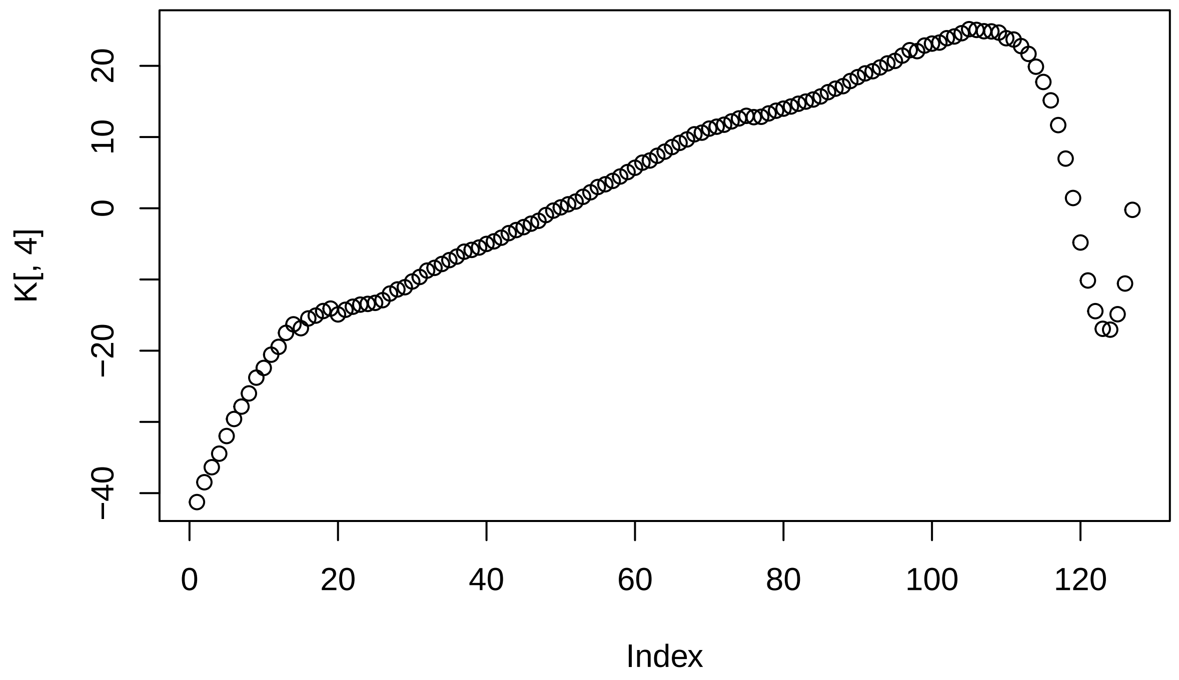
```r
plot(K[,2])
```



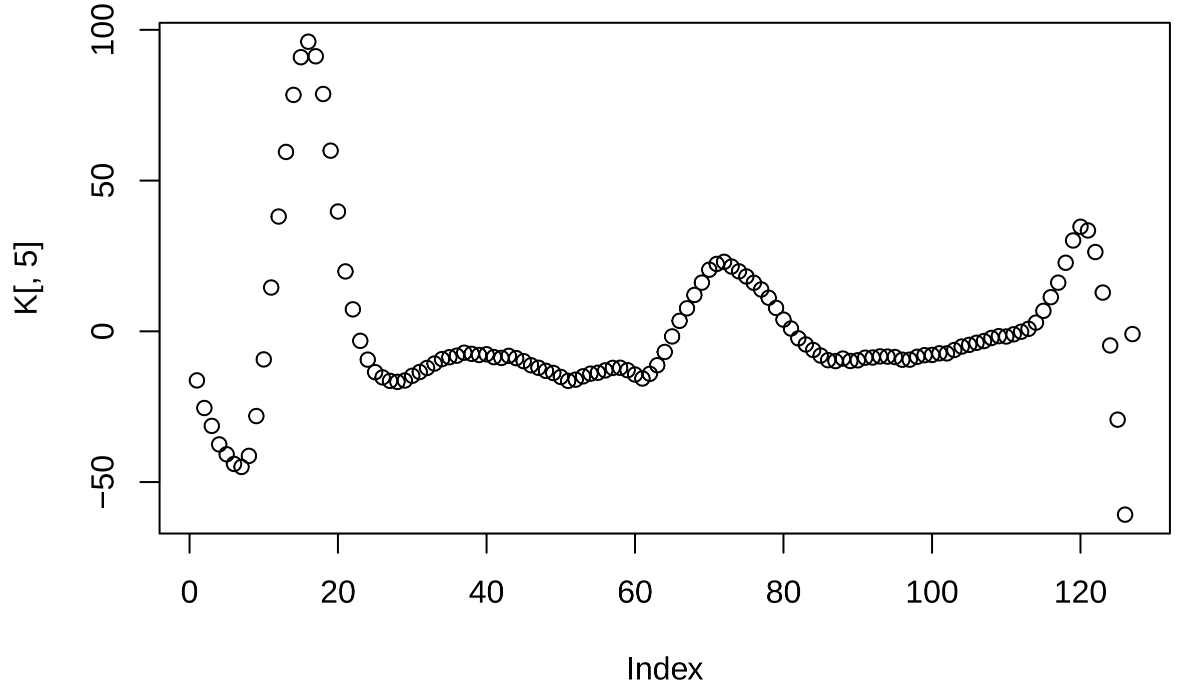
```r
plot(K[,3])
```



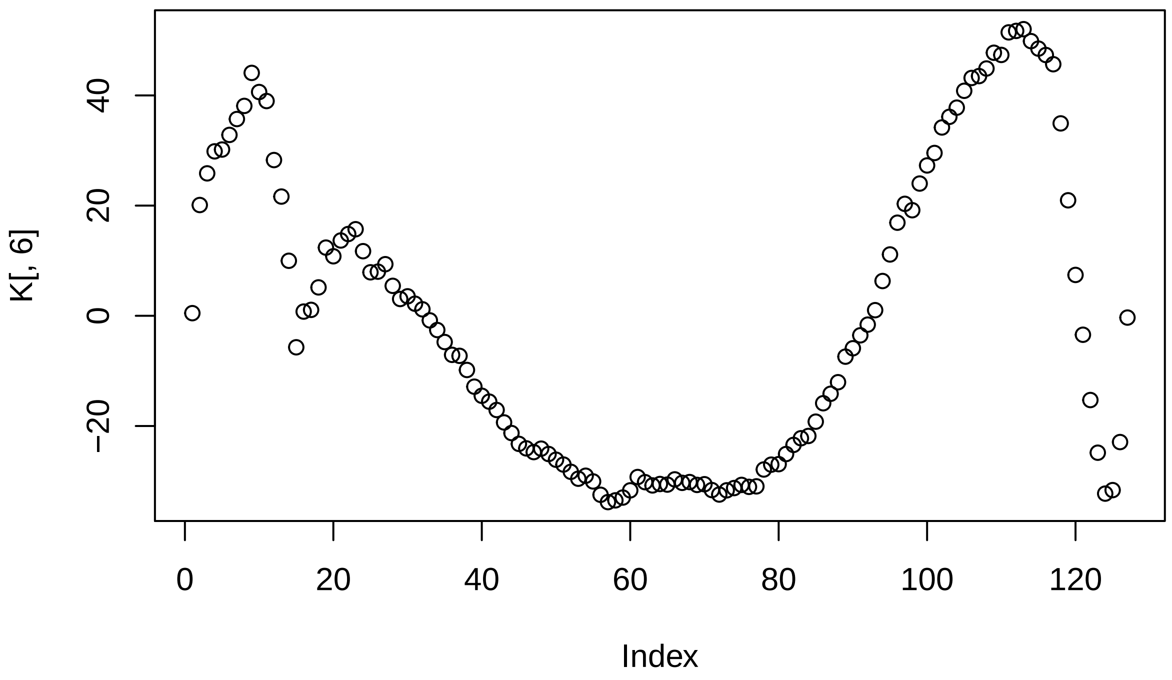
```r
plot(K[,4])
```



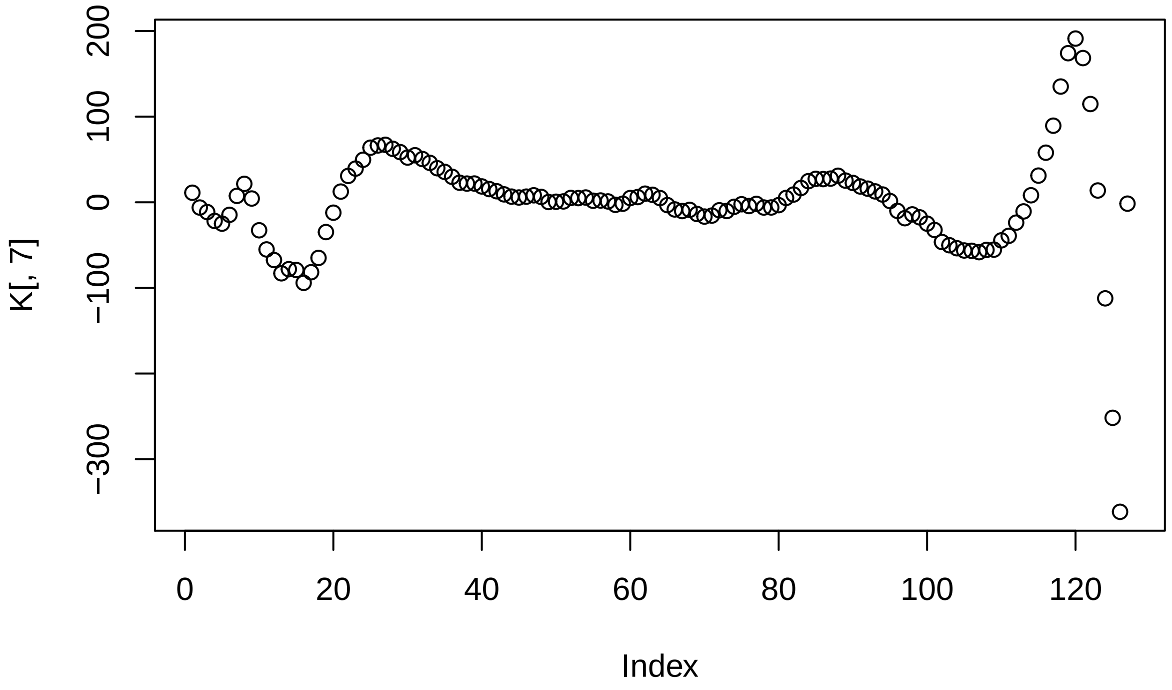
```r
plot(K[,5])
```



```r
plot(K[,6])
```



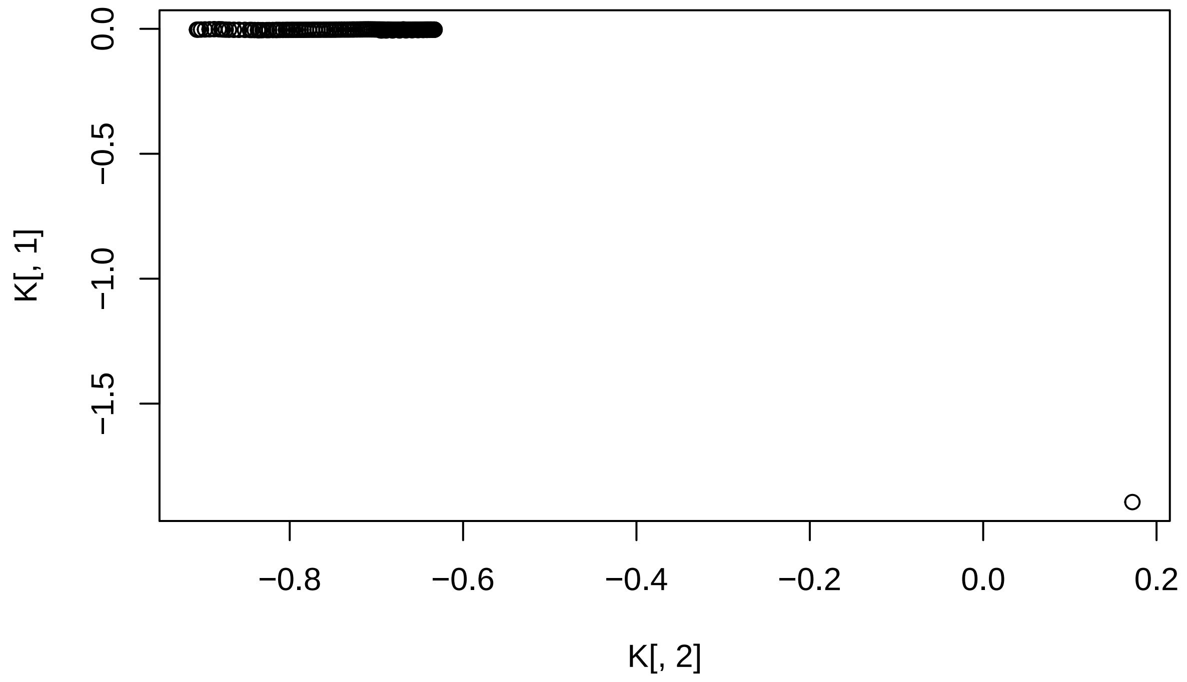
```r
plot(K[,7])
```



Again, traceplot 2 is formed by only a few features. After centring and and standardizing each column of the original datafram, the data is projected into principal components directions. Then, a matrix W is estimated which maximizes the so called neg-entropy approximation (source Rhelp, ?fastICA)

4.3

```r
plot(K[,2], K[,1])
```



Plot is rather similar to the one in step1, however one outlier appears in both dimensions.