Computer lab 2

Phillip Hölscher

8 12 2018

Assignment 2. Analysis of credit scoring

2.2

Misclassification rate

- Train:

```
mcer_classifier_tree_dev
```

```
## [1] 0.2105263
```

```
mcer_classifier_tree_gi
```

```
## [1] 0.2368421
```

- Test:

```
mcr_tree_dev_test
```

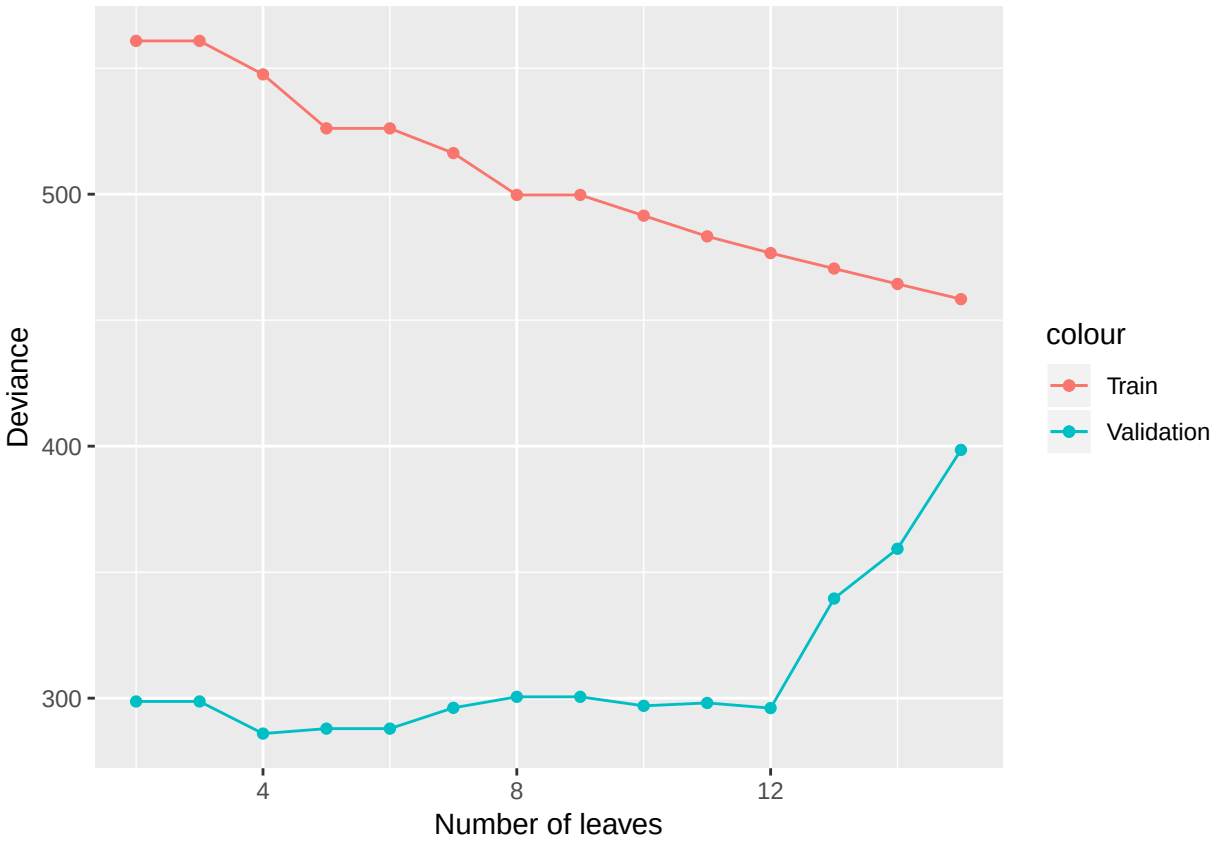
```
## [1] 0.384
```

```
mcr_tree_gi_test
```

```
## [1] 0.464
```

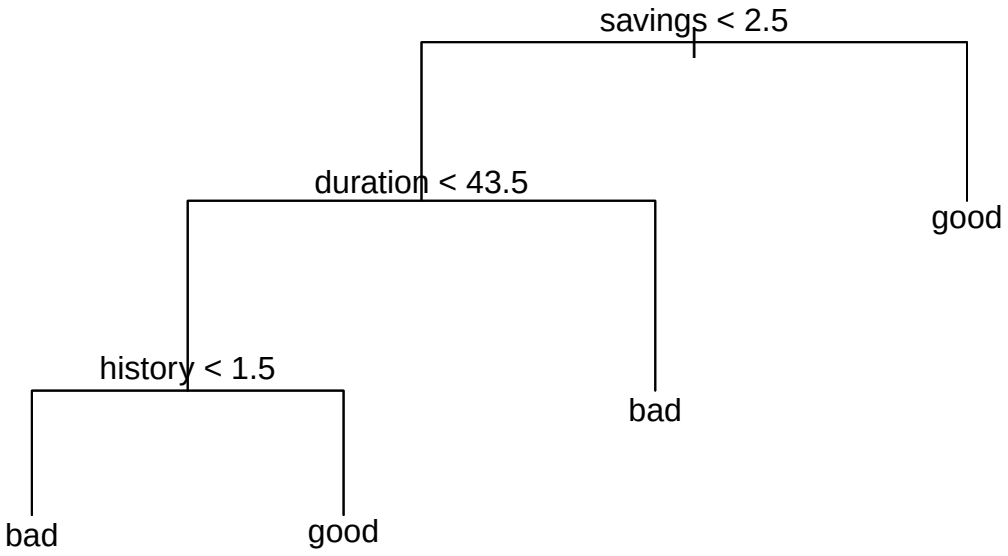
We can see that measures “deviance” is better both in training off and in the test case. Therefore the deviance is used in the further course and not the gini.

Deviance of the training and validation set



We can see that the deviation of the data set of train decreases continuously with increasing leaves. Furthermore, it can be seen that when validation set to leave twelve is continuous, with a small decrease between two and eight. The minimum can be seen at 4, which is the optimal number of leaves.

Report the optimal tree



Variables used

```
## [1] savings  duration history
## 20 Levels: <leaf> duration history purpose amount savings ... foreign
```

As already mentioned it can be seen that four is the optimal number for the leaves, savings duration and history were used as variables.

Misclassification optimal tree

```
## [1] 0.256
```

The misclassification has now been greatly reduced, from 38.4 to 25.6%.

2.4 Naïve Bayes

- Train: Confusion Matrix

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

Mislcassification

```
## [1] 0.3
```

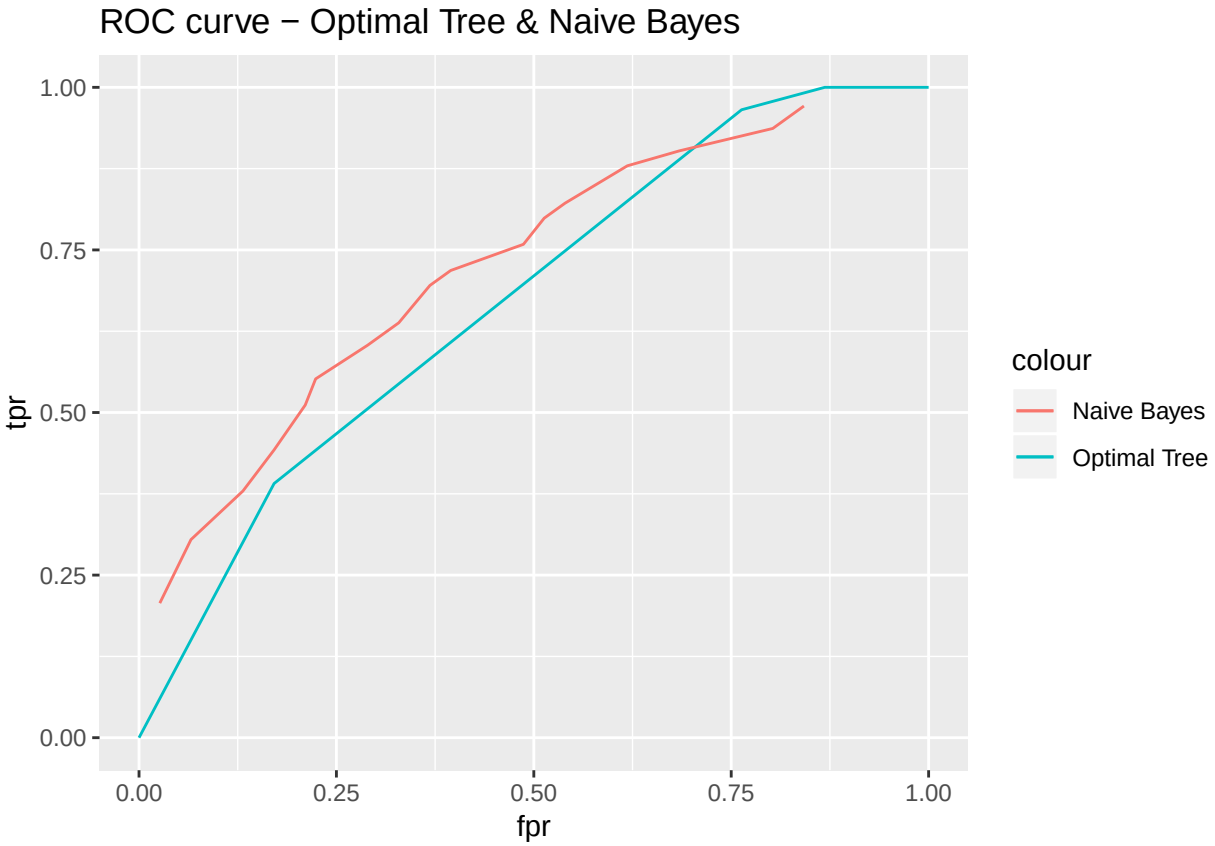
- Test: Confusion Matrix

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

Mislcassification

```
## [1] 0.316
```

2.5



In this plot it can be seen that the "Naive Bayes" method almost always performs better than the "Optimal Tree". The reason for this is that the red line ("Naive Bayes") lies above the blue line ("Optimal Tree"). Only between the fpr value of 0.65 and 0.75 the graphs cross. After that the performance of the "Optimal Tree" is better. Analytically expressed, the tpr value of the "Naive Bayes" is better at the same fpr value.

2.6

Confusion Matrix

- Train:

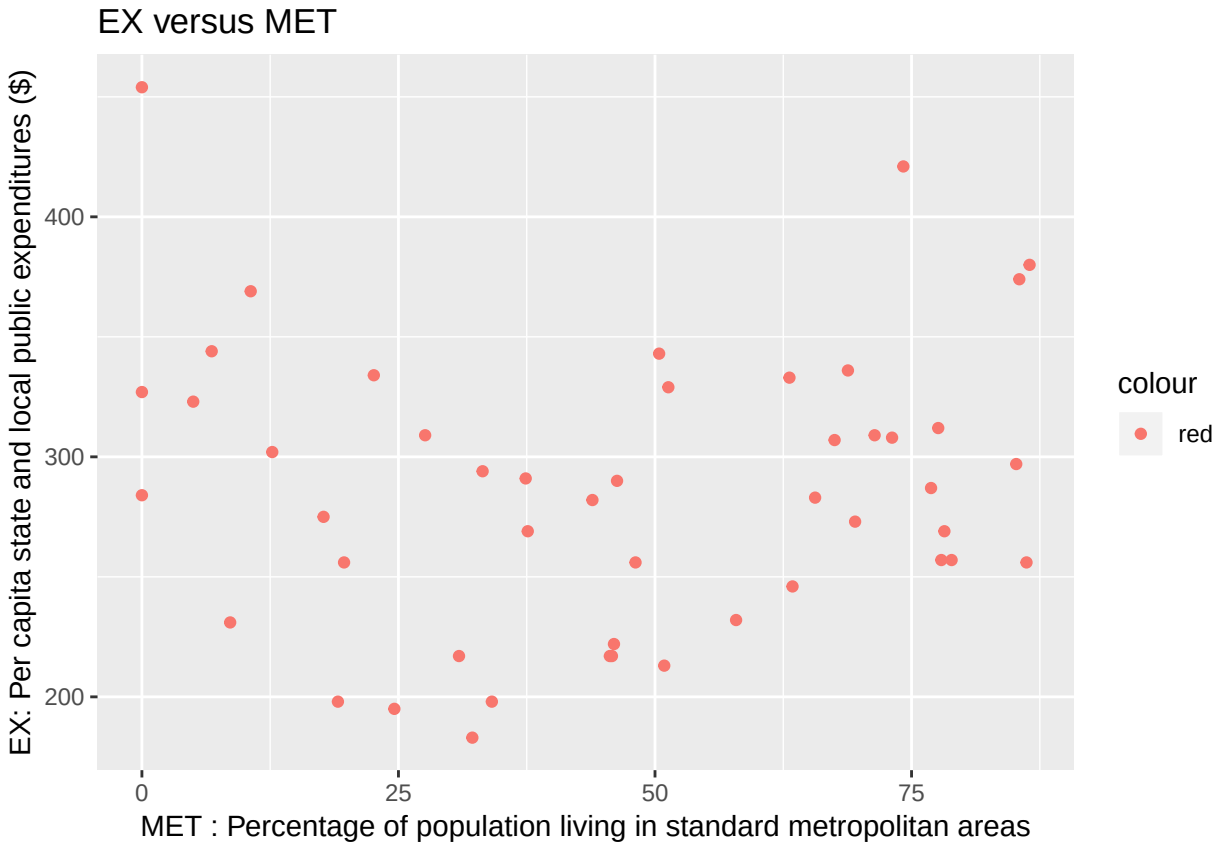
```
##    pred_nb_train_prob_loss
##       0   1
##   0 120  27
##   1 336  17
```

- Test:

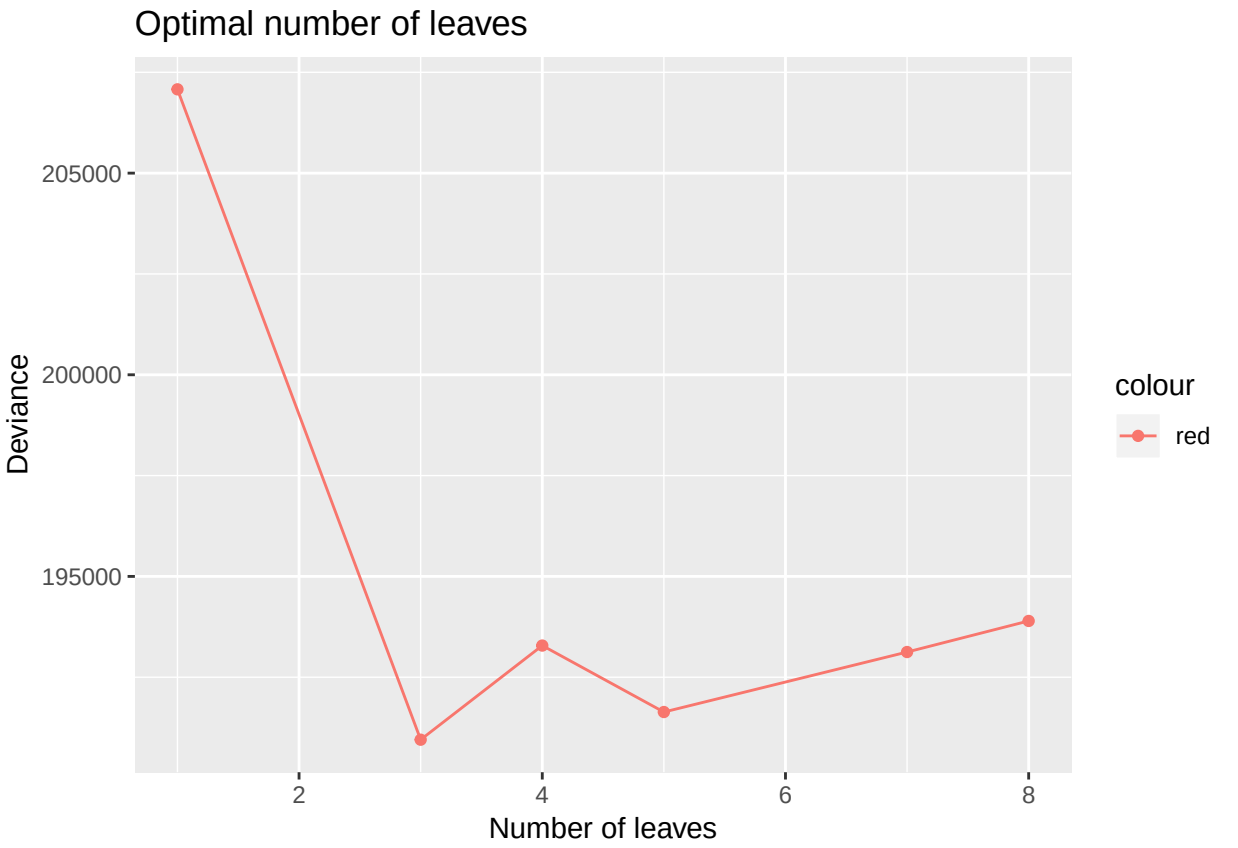
```
##    pred_nb_test_prob_loss
##       0   1
##   0  62  14
##   1 164  10
```

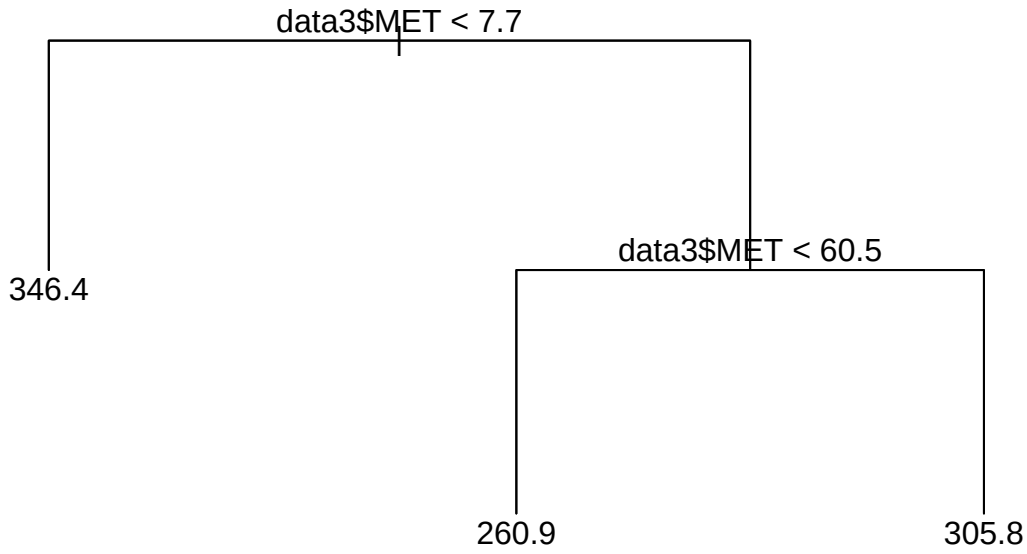
(0 = bad, 1 = good)

Assignment 3. Uncertainty estimation

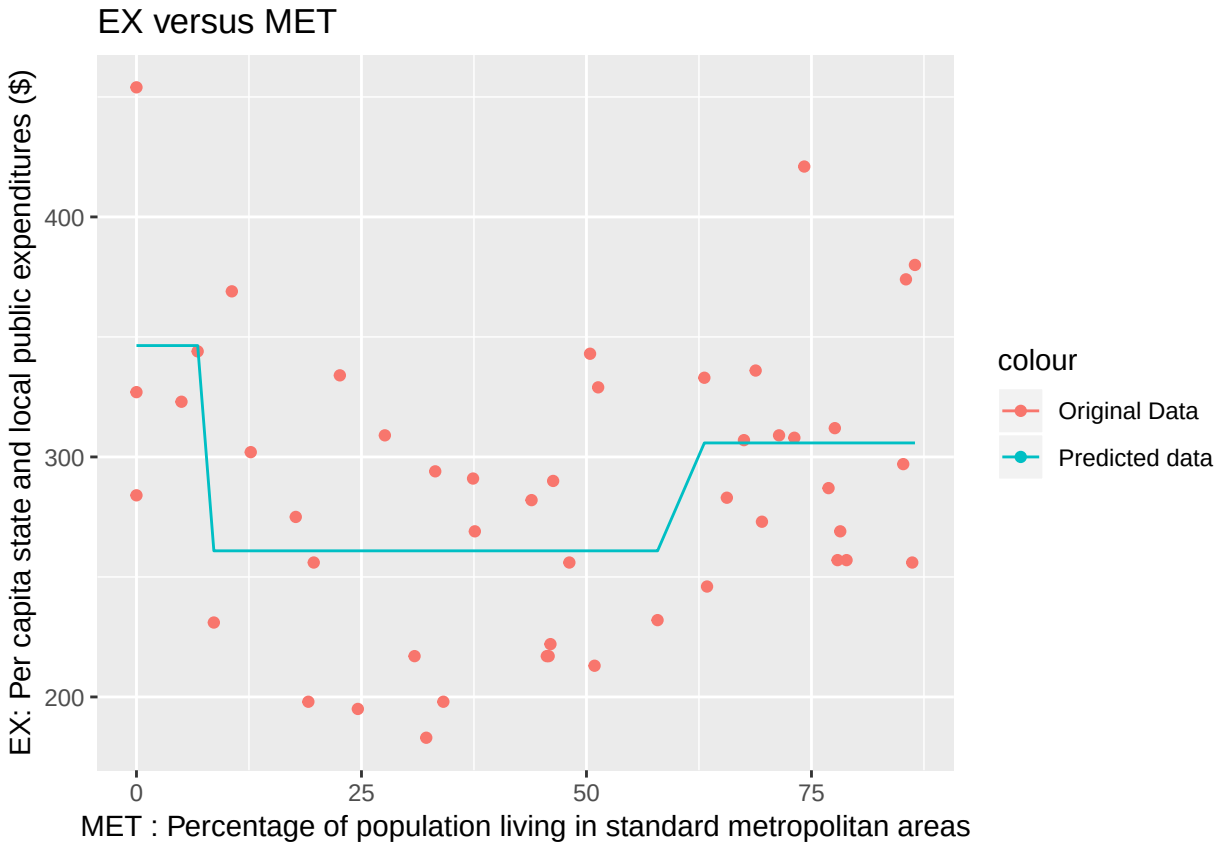


Different methods can be used for K classification. A linear or polinomial regression would be possible, but also a tree regression. We will investigate these further on.



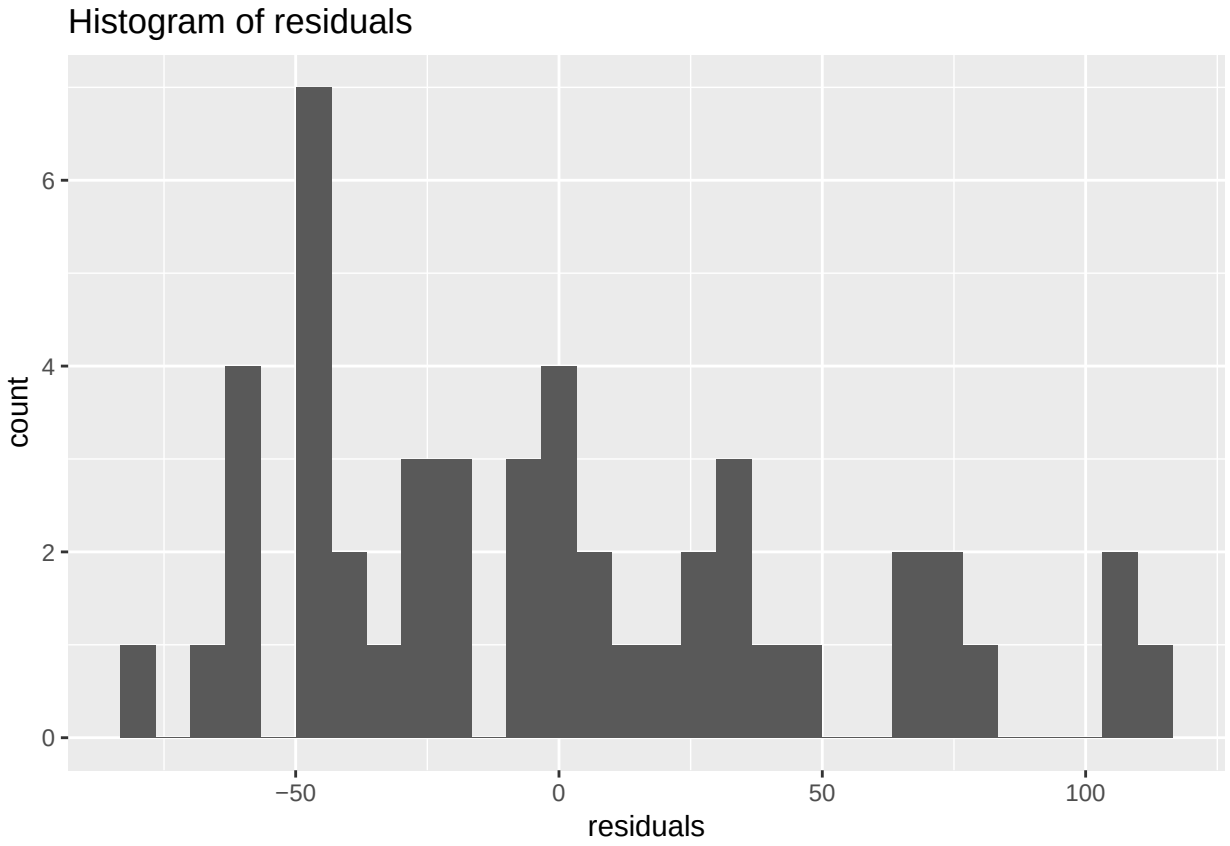


The optimal tree:

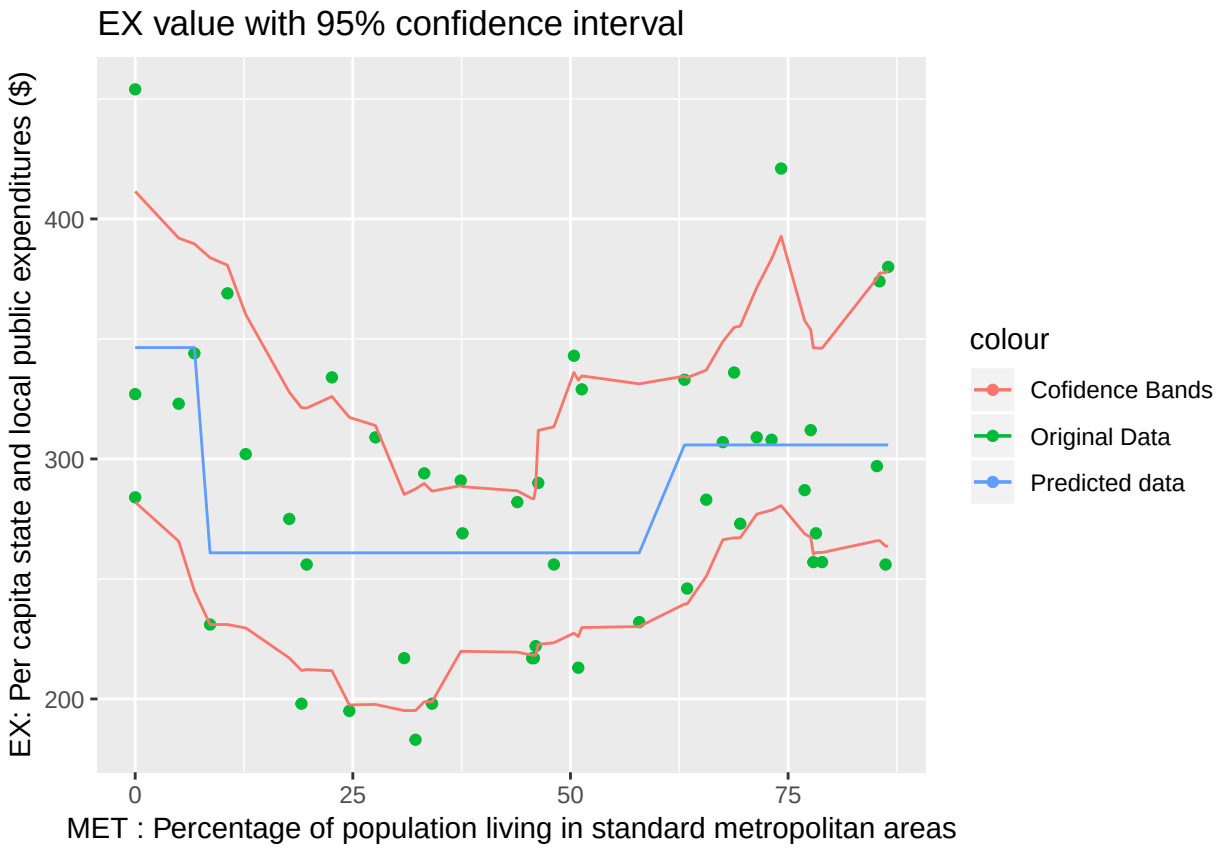


Histogram - Residuals of the fit

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

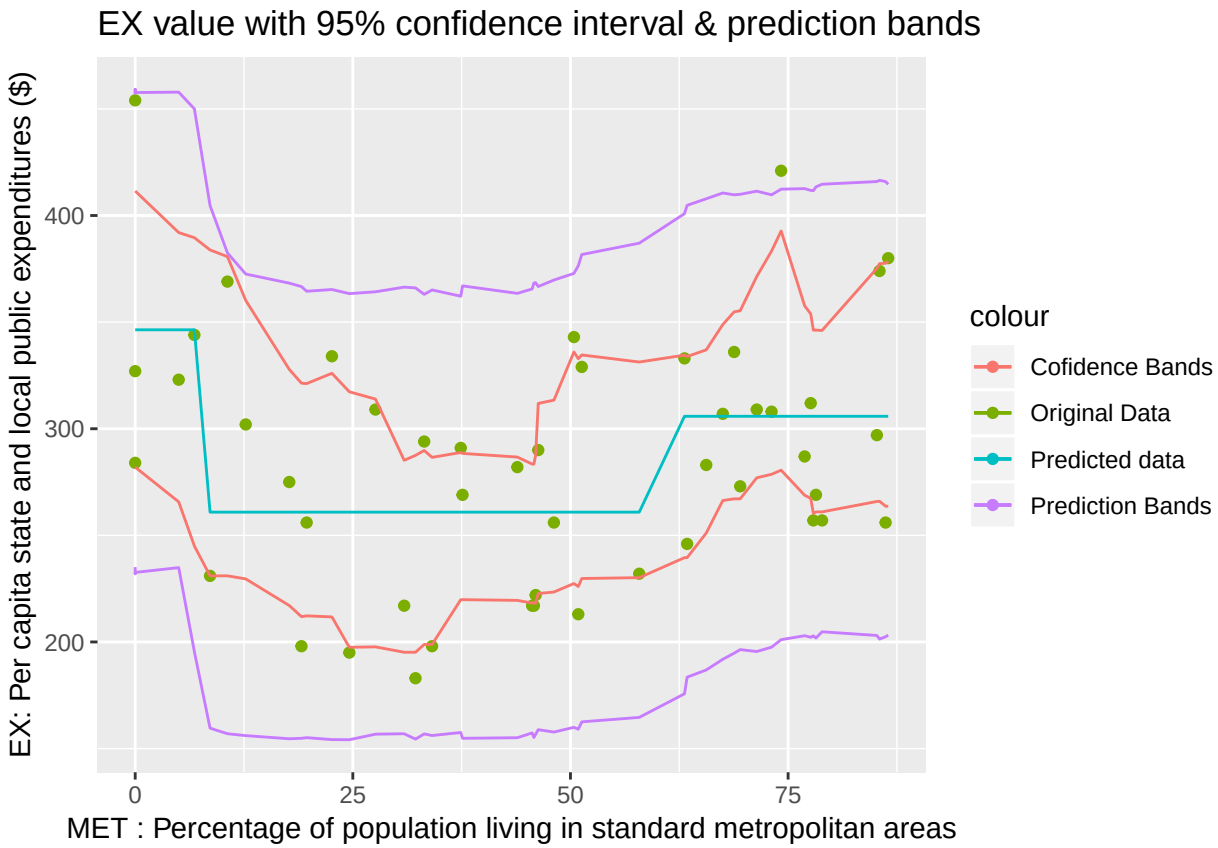


In this distribution we can see that it looks relatively normal, but with a right skewness. It can also be seen that the larger part of the residuals are in the negative range.



The confidence bands are not smooth, so rather bumpy. This can be explained by the fact that the forecast itself shows strong changes. This is partly due to the large gaps between confidence bands and prediction. An explanation for these strong changes could be the number of leaves, because this is a small number which can lead to strong changes.

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

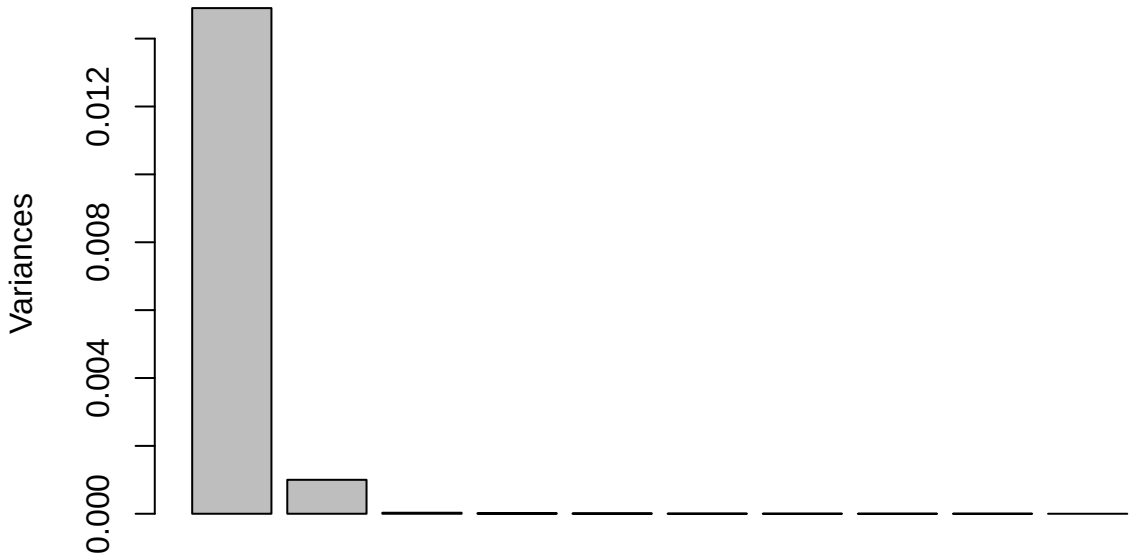


It can be seen that there is only one point outside the prediction bands. Thus it can be said that more than 95% of the predicted range is located. Accordingly, a distribution as given makes sense.

Since the prediction fits very well with the given distribution, it can be said that parametric booststrap fits well.

Assignment 4. Principal components

Result of principal components analysis



The components which are bigger or same as 0.001:

```
## [1] "93.332"
## [1] "6.263"
## [1] "0.185"
## [1] "0.101"
## [1] "0.068"
## [1] "0.025"
## [1] "0.009"
## [1] "0.003"
## [1] "0.003"
## [1] "0.002"
## [1] "0.001"
```

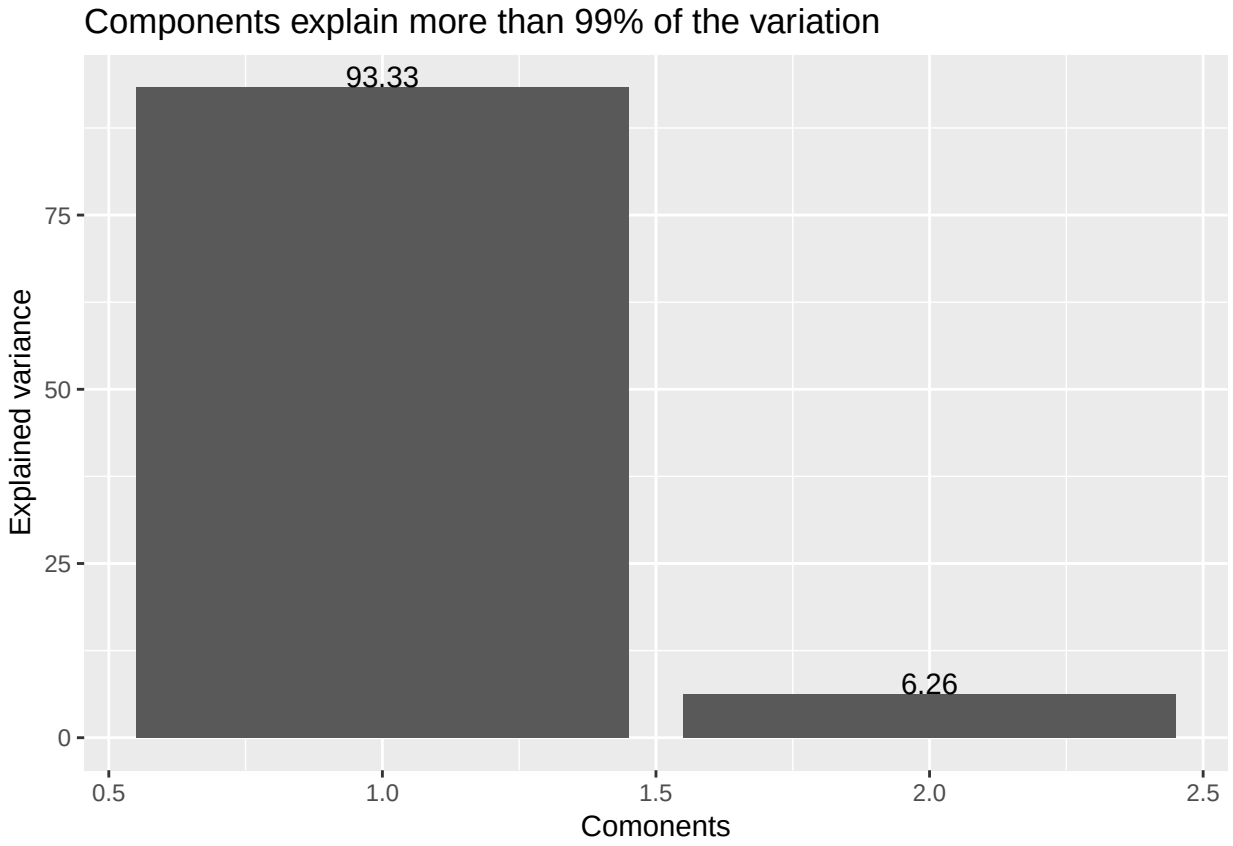
The minimum amount of components to explain 99% of the total variance

Amount of components:

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

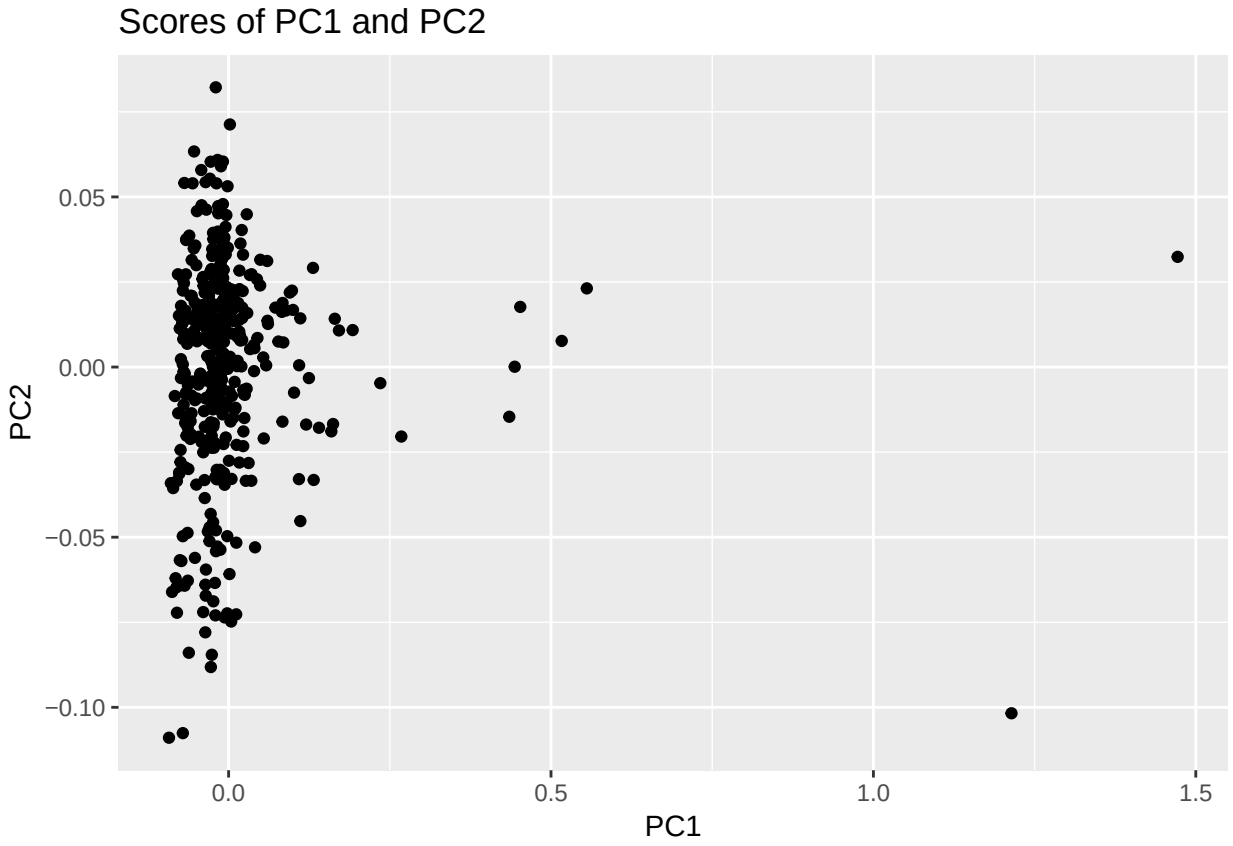
Total variance explained:

```
## [1] 99.596
```

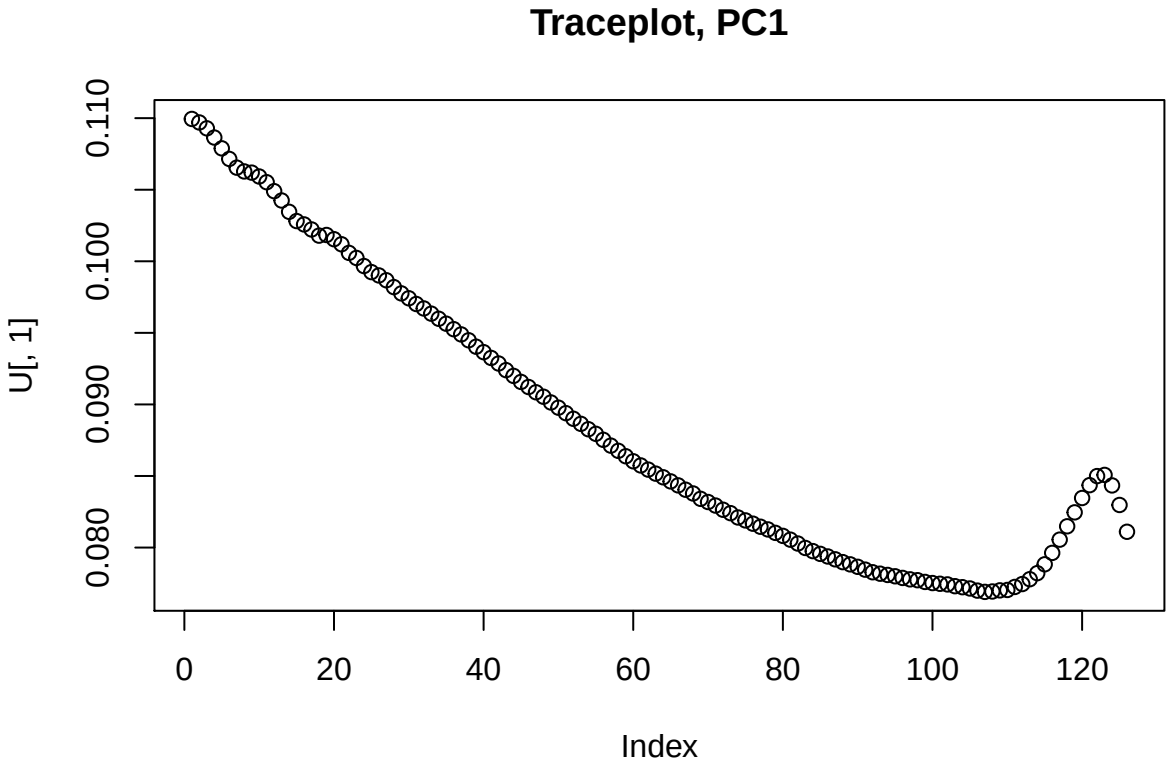


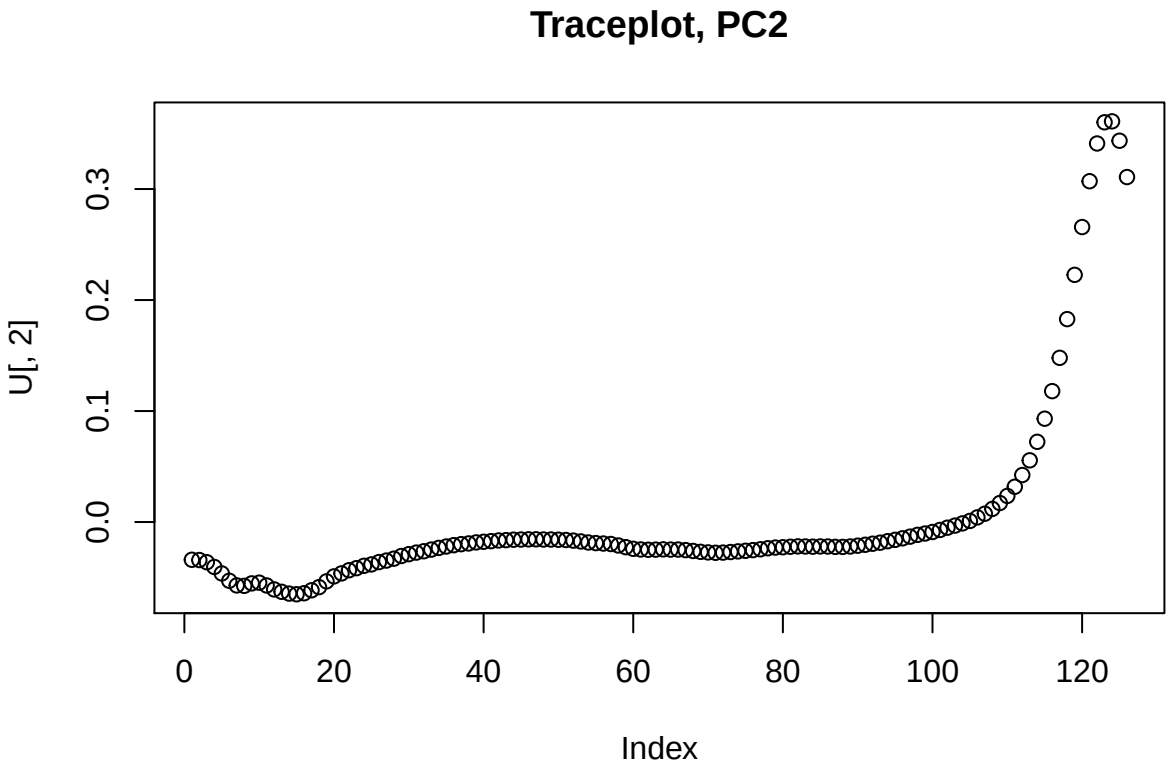
The minimum number of PC´s is 2. These two explain more than 99% of the variance, as can be seen in the visualizations.

Score of coordinates PC1 and PC2



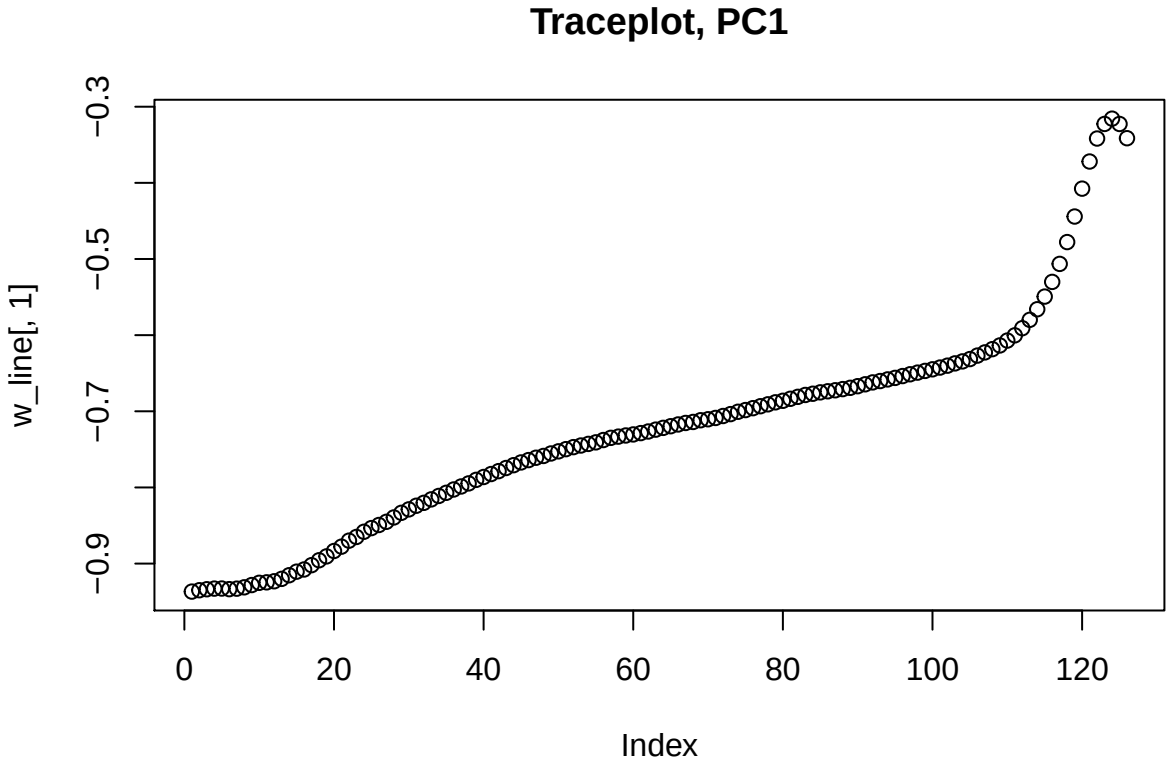
The trace plots

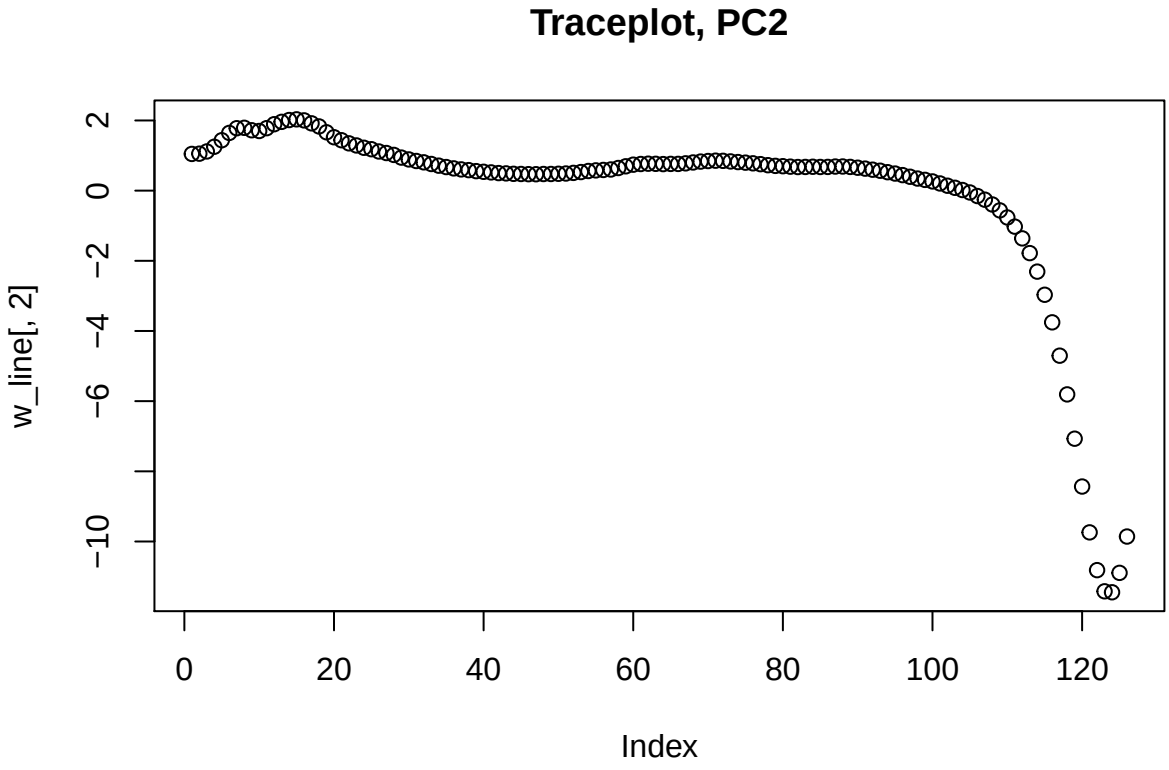


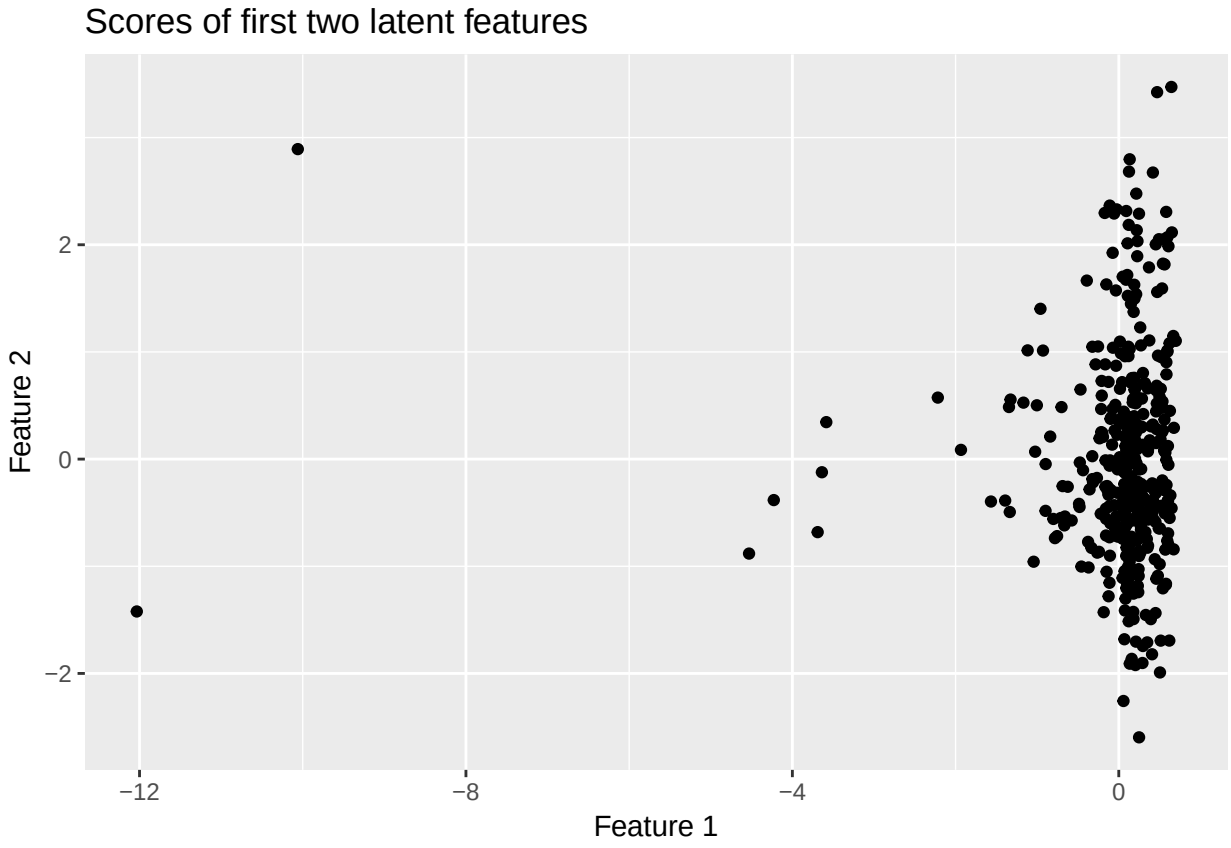


It can be seen that the feature index of the second plot is not as strong as that of the first, so the PC2 has just a few original features.

Independent component analysis







In all of the last three visualizations it can be seen that these are very similar to the one from the second task, they are only mirrored.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE)
#install.packages("tree")
library(tree)
# Naive Bayes classification
library(MASS)
#install.packages("e1071")
library(e1071)
#install.packages("SDMTools")
library(SDMTools)
#install.packages("ggplot2")
library(ggplot2)
#install.packages("boot")
library(boot)
#install.packages("fastICA")
library(fastICA)
#install.packages("gdata")
library(gdata)
# all the libraries used in assignment 2
# Assignment 2. Analysis of credit scoring ####

# 2.1 ----

# Import the data

# set working directory ------ set your working directory here !!!!
#setwd(" ")

# Import the data
data <- read.xls("creditscoring.xls")

# split the data into training, validation & test set
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 ----
# fit decision tree

# fit the model
classifier_tree_dev = tree(train$good_bad~. ,
                           data = train,
                           split = "deviance")

classifier_tree_gi = tree(train$good_bad~. ,
```

```
                          data = train,
                          split = "gini")

# misclassification - TRAIN
mcr_deviance <- misclass.tree(classifier_tree_dev) # 104
mcr_gini <- misclass.tree(classifier_tree_gi) # 117

# misclass error rate - TRAIN
mcer_classifier_tree_dev <- summary(classifier_tree_dev)
mcer_classifier_tree_dev <- mcer_classifier_tree_dev$misclass[1]/mcer_classifier_tree_dev$misclass[2]

mcer_classifier_tree_gi <- summary(classifier_tree_gi)
mcer_classifier_tree_gi <- mcer_classifier_tree_gi$misclass[1]/mcer_classifier_tree_gi$misclass[2]

# RESULT - misclassification - TRAIN data
# The - deviance - has a lower mcr !

# Missclassification - TEST data
#  --- deviance ---
# create prediction of TEST data
pred_tree__dev_test <- predict(classifier_tree_dev,
                           newdata = test,
                           type = "class")

# confusion matrix - TRAIN data - deviance
cm_tree_dev_test <- table(test$good_bad, pred_tree__dev_test)

# misclassification rate - TRAIN data - deviance
mcr_tree_dev_test <- (cm_tree_dev_test[1,2] + cm_tree_dev_test[1,2]) / sum(cm_tree_dev_test)

# --- gini ---
# create prediction of TEST data - gini
pred_tree_gi_test <- predict(classifier_tree_gi,
                             newdata = test,
                             type = "class")
# confusion matrix - TRAIN data - gini
cm_tree_gi_test <- table(test$good_bad, pred_tree_gi_test)

# misclassification rate - TRAIN data - gini
mcr_tree_gi_test <- (cm_tree_gi_test[1,2] + cm_tree_gi_test[1,2]) / sum(cm_tree_gi_test)

# RESULT - misclassification - TEST data
# The deviance has a lower mcr !

# plot(classifier_tree_dev)
# text(classifier_tree_dev, pretty=0)
# classifier_tree_dev
# summary(classifier_tree_dev)

mcer_classifier_tree_dev
mcer_classifier_tree_gi
mcr_tree_dev_test
mcr_tree_gi_test
```

```r
# 2.3 ----
#fit=tree(class~., data=train)
classifier_tree_dev2 = tree(good_bad~. ,
                            data = train,
                            split = "deviance")

trainScore=rep(0,15)
testScore=rep(0,15)

for(i in 2:15) {
  prunedTree = prune.tree(classifier_tree_dev2,best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree")
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred)
}

col <- c("Train" = "red", "Validation" = "blue")
train_valid_optimaltree <- ggplot() +
  geom_line(aes(2:15,trainScore[2:15], color = "Train")) +
  geom_point(aes(2:15,trainScore[2:15], color = "Train")) +
  geom_line(aes(2:15,testScore[2:15], color = "Validation")) +
  geom_point(aes(2:15,testScore[2:15], color = "Validation")) +
  ylab("Deviance") +
  xlab("Number of leaves")

# the deviance is at leave the lowest
# best tree depth is 4

# create the optimal tree
finalTree_2.3 <- prune.tree(classifier_tree_dev2,
                            best = 4)
# report the tree
# plot(finalTree_2.3)
# text(finalTree_2.3, pretty=0)

# variable used
finalTree_2.3_varused <- summary(finalTree_2.3)
finalTree_2.3_varused <- finalTree_2.3_varused$used

# misclassifcation - TEST data
pred_finalTree <- predict(finalTree_2.3,
                          newdata = test,
                          type = "class")
cm_finalTree <- table(test$good_bad, pred_finalTree)
mcr_finalTree <- (cm_finalTree[1,2] + cm_finalTree[2,1]) / sum(cm_finalTree)
# 0.256

train_valid_optimaltree
plot(finalTree_2.3)
text(finalTree_2.3, pretty=0)
finalTree_2.3_varused
mcr_finalTree
```

```r
# 2.4 ----
# classifier for naive bayes on train data
classifier_nb <- naiveBayes(formula = good_bad~.,
                            data = train)
# prediction of train data
pred_nb_train <- predict(classifier_nb,
                         newdata = train[-20])

# confusion matrix train data
cm_nb_train <- table(train$good_bad, pred_nb_train)
# misclassification rate
mcr_nb_train <- (cm_nb_train[1,2] + cm_nb_train[2,1])/ sum(cm_nb_train)

# prediction of test data
pred_nb <- predict(classifier_nb,
                   newdata = test[-20])

# confusion matrix test data
cm_nb <- table(test$good_bad, pred_nb)

# misclassification rate
mcr_nb <- (cm_nb[1,2] + cm_nb[2,1])/ sum(cm_nb)

# compare the results to step 2.3
# The result from step 2.3 is better
# 2.3 => 0.256 & 2.4 => 0.316

cm_nb_train
mcr_nb_train
cm_nb
mcr_nb
# 2.5 ----
# create the pi vector
pi_inc <- seq(from = 0.05,
              to = 0.95,
              by = 0.05)

# optimal tree - classify TEST data
# create prediction
pred_finalTree_prob <- predict(finalTree_2.3,
                           newdata = test)

#pred_finalTree_prob[,2] # prob of 'good' classifications
pred_finalTree_prob_good <- pred_finalTree_prob[,2]

# transfare "good" = 1 and "bad" = 0
test_gb_numb <- test
test_gb_numb$good_bad <- ifelse(test_gb_numb$good_bad == "good", 1, 0)

# cm optimal tree pred - TEST data

# create empty matrix for TPR & FPR values
tpr_fpr_ot <- matrix(ncol = 2, nrow = length(pi_inc))
```

```r
colnames(tpr_fpr_ot) <- c("TPR", "FPR")

for (i in 1:length(pi_inc)) {
  test_ot <- ifelse(pred_finalTree_prob_good > pi_inc[i], 1, 0)
  cm_test <- confusion.matrix(test_ot, test_gb_numb$good_bad)

  tpr <- cm_test[2,2]/(cm_test[2,1] + cm_test[2,2])
  fpr <- cm_test[1,2]/(cm_test[1,1] + cm_test[1,2])

  tpr_fpr_ot[i,] <- c(tpr,fpr)
}

# naive bayes to classify TEST data

# classifier for naive bayes on train data
classifier_nb2 <- naiveBayes(good_bad~.,
                             data = train)

# prediction of test data
pred_nb_prob <- predict(classifier_nb2,
                  newdata = test,
                  type = "raw") # type raw - creates a probability prediction

#pred_nb_prob[,2] # prob of 'good' classifications
pred_nb_prob <- pred_nb_prob[,2]

# create empty matrix for TPR & FPR values
tpr_fpr_nb <- matrix(ncol = 2, nrow = length(pi_inc))
colnames(tpr_fpr_nb) <- c("TPR", "FPR")

for (i in 1:length(pi_inc)) {
  test_nb <- ifelse(pred_nb_prob > pi_inc[i], 1, 0)
  cm_test <- confusion.matrix(test_nb, test_gb_numb$good_bad)

  tpr <- cm_test[2,2]/(cm_test[2,1] + cm_test[2,2])
  fpr <- cm_test[1,2]/(cm_test[1,1] + cm_test[1,2])

  tpr_fpr_nb[i,] <- c(tpr,fpr)
}

# plot ROC (Receiver operating characteristics)

col <- c("Optimal Tree" = "red", "Naive Bayes" = "blue")

# ROC Optimal Tree & Naive Bayes
roc_op_nb_plot <- ggplot() +
  geom_line(aes(x = tpr_fpr_ot[,2], y = tpr_fpr_ot[,1], col = "Optimal Tree")) +
  geom_line(aes(x = tpr_fpr_nb[,2], y = tpr_fpr_nb[,1], col = "Naive Bayes")) +
  xlab("fpr") +
  ylab("tpr") +
  ggtitle("ROC curve - Optimal Tree & Naive Bayes")

roc_op_nb_plot
```

```r
# 2.6 ----
# use the classifier from 2.4 & pred probabilities
set.seed(12345)

# classifier for naive bayes on train data
classifier_nb <- naiveBayes(formula = good_bad~.,
                            data = train)
# prediction of train data
pred_nb_train_prob <- predict(classifier_nb,
                          newdata = train[-20],
                          type = "raw") # creates  the probabilites

# create L_Observed
l_observed <- matrix(c(0,1,10,0), nrow = 2, byrow = TRUE)

# crate a classification - TRAIN data - implement loss function
pred_nb_train_prob_loss <- ifelse(l_observed[1,2] * pred_nb_train_prob[,1] > l_observed[2,1] * pred_nb_

# confusion matrix - TRAIN data
cm_nb_loss_train <- table(ifelse(train$good_bad == "good",1,0), pred_nb_train_prob_loss)

# prediction of test data
pred_nb_test_prob <- predict(classifier_nb,
                             newdata = test[-20],
                             type = "raw")

# crate a classification - TEST data - implement loss function
pred_nb_test_prob_loss <- ifelse(l_observed[1,2] * pred_nb_test_prob[,1] > l_observed[2,1] * pred_nb_te

# # confusion matrix - TRAIN data
cm_nb_loss_test <- table(ifelse(test$good_bad == "good",1,0), pred_nb_test_prob_loss)
cm_nb_loss_train
cm_nb_loss_test
# Assignment 3. Uncertainty estimation ####

# 3.1 ----
# read the data
data3 <- read.csv2("State.csv")

# order MET increasing
data3 <- data3[order(data3$MET),]

# plot EX versus MET

plot_EX_versus_MET <- ggplot(data = data3) +
  #geom_line(aes(x = data3$MET, y = data3$EX, color = "red")) +
  geom_point(aes(x = data3$MET, y = data3$EX, color = "red")) +
  xlab("MET : Percentage of population living in standard metropolitan areas") +
  ylab("EX: Per capita state and local public expenditures ($)") +
  ggtitle("EX versus MET")
plot_EX_versus_MET
# fit regression tree
regressor_tree <- tree(data3$EX ~ data3$MET,
```

```
                      data = data3,
                      control = tree.control(nobs = nrow(data3), minsize = 8))
# number of leaves
set.seed(12345)
regressor_tree_cv <- cv.tree(regressor_tree,  FUN  = prune.tree)

optimal_tree_nr <- ggplot() +
  geom_line(aes(x = regressor_tree_cv$size, y = regressor_tree_cv$dev, color = "red")) +
  geom_point(aes(x = regressor_tree_cv$size, y = regressor_tree_cv$dev, color = "red")) +
  xlab("Number of leaves") +
  ylab("Deviance") +
  ggtitle("Optimal number of leaves")

# we see from this plot the optimal number of leaves is 3
optimal_tree_nr
# report the selected tree
# fit the best tree
regressor_tree_best <- prune.tree(tree = regressor_tree,
                                  best = 3)
# report the selected tree
plot(regressor_tree_best)
text(regressor_tree_best, pretty=0)
# regressor_tree_best
# summary(regressor_tree_best)
# plot - original data and fitted data

col <- c("Original Data" = "blue", "Predicted data" = "red")
ggplot(data = data3) +
  geom_point(aes(x = data3$MET, y = data3$EX, color = "Original Data")) +
  xlab("MET : Percentage of population living in standard metropolitan areas") +
  ylab("EX: Per capita state and local public expenditures ($)") +
  ggtitle("EX versus MET") +
  geom_line(aes(x = data3$MET, y = predict(regressor_tree_best), color = "Predicted data"))
  # + geom_point(aes(x = data3$MET, y = predict(regressor_tree_best), color = "Predicted data"))

hist_of_reso <- ggplot() +
  geom_histogram(aes(x = resid(regressor_tree_best))) +
  ggtitle("Histogram of residuals")+
  xlab("residuals")
hist_of_reso
set.seed(12345)

# order MET increasing
#data3 <- data3[order(data3$MET),]

# Nonparametric bootstrap
# computing bootstrap samples
f <- function(data, ind){
databoosttramp <- data[ind,] # extract bootstrap sample
tree_23 <- tree(EX~MET,
                data=databoosttramp,
                control = tree.control(nobs = nrow(data3), minsize = 8))
```

```
res <- prune.tree(tree_23,best = 4)
#predict values for the original data
y_pred <- predict(res, newdata=data3) # take the reordered data
return(y_pred)
}
# make boost
make_boost <- boot(data3,f,R=1000)
# compute confidence bands
e <- envelope(make_boost, level = 0.95)
# regression tree step 2
regressor_tree <- tree(data3$EX ~ data3$MET,
                       data = data3,
                       control = tree.control(nobs = nrow(data3), minsize = 8))
# the best tree - step 2
regressor_tree_best <- prune.tree(tree = regressor_tree,
                                  best = 3)
# prediction - best tree
regressor_tree_best_pred <- predict(regressor_tree_best)

# create the plot of data with prediction and bootstrap
col <- c("Original Data" = "blue", "Predicted data" = "red", "Cofidence Bands" = "green")
booststrap_plot = ggplot(data = data3) +
  geom_point(aes(x = data3$MET, y = data3$EX, color = "Original Data")) +
  xlab("MET : Percentage of population living in standard metropolitan areas") +
  ylab("EX: Per capita state and local public expenditures ($)") +
  ggtitle("EX value with 95% confidence interval") +
  geom_line(aes(x = data3$MET, y = regressor_tree_best_pred, color = "Predicted data")) +
  geom_line(aes(x = data3$MET, y = e$point[2,], color = "Cofidence Bands")) +
  geom_line(aes(x = data3$MET, y = e$point[1,], color = "Cofidence Bands"))

booststrap_plot
# Assignment 3 Exercise 4
# parametric bootstrap
set.seed(12345)

# regression tree step 2
mle <- regressor_tree_best

rng=function(data, mle) {
  data1=data.frame(EX=data$EX, MET=data$MET)
  n=length(data$EX)
  pred <- predict(mle, newdata = data3)
  residual <- data3$EX - pred
  #generate new Price
  data1$EX=rnorm(n, pred, sd(residual))
  return(data1)
}

f1 = function(data){
  res = tree(data = data,
             EX~MET,
             control = tree.control(nobs=NROW(data3),minsize = 8))
  res_opti = prune.tree(res, best = 3)
```

```
  n = length(data3$EX)
  res_opti_pred = predict(res_opti, newdata = data3)
  predicted = rnorm(n,res_opti_pred,sd(residuals(mle)))
  return(predicted)
}

res = boot(data3, statistic=f1, R=10000, mle=mle, ran.gen=rng, sim="parametric")
e1 <- envelope(res, level = 0.95)

# create the plot of data with prediction and bootstrap
col <- c("Original Data" = "blue", "Predicted data" = "red", "Prediction Bands" = "green", "Cofidence Ba
pred_booststrap = ggplot(data = data3) +
  geom_point(aes(x = data3$MET, y = data3$EX, color = "Original Data")) +
  xlab("MET : Percentage of population living in standard metropolitan areas") +
  ylab("EX: Per capita state and local public expenditures ($)") +
  ggtitle("EX value with 95% confidence interval & prediction bands") +
  geom_line(aes(x = data3$MET, y = predict(regressor_tree_best), color = "Predicted data")) +
  geom_line(aes(x = data3$MET, y = e$point[2,], color = "Cofidence Bands")) +
  geom_line(aes(x = data3$MET, y = e$point[1,], color = "Cofidence Bands")) +
  geom_line(aes(x = data3$MET, y = e1$point[2,], color = "Prediction Bands")) +
  geom_line(aes(x = data3$MET, y = e1$point[1,], color = "Prediction Bands"))
pred_booststrap

# read the data
NIRSpectra <- read.csv2("NIRSpectra.csv", header=TRUE,stringsAsFactors=FALSE)
pca_data = NIRSpectra
pca_data$Viscosity = c()
res = prcomp(pca_data) # principal components analysis
#eigenvalues
lambda = res$sdev^2
#proportion of variation
#sprintf("%2.3f",lambda/sum(lambda)*100)
variation_proportion = lambda/sum(lambda)*100
screeplot(res)
i = 1
while(variation_proportion[i] >= 0.001){
  print(sprintf("%2.3f",lambda/sum(lambda)*100)[i])
  i = i+1
}

# values until 99%
value_99_sum = 0
value_99 = c()
i = 1
while (value_99_sum < 99) {
  value_99_sum = value_99_sum + variation_proportion[i]
  value_99[i] = variation_proportion[i]
  i = i +1
}
length(value_99)
round(value_99_sum,3)

# barplot
```

```r
value_99_df = as.data.frame(value_99)
value_99_df$nr_component = 1:length(value_99)
value_99_df$lab = c(as.character(round(value_99[1],2)),as.character(round(value_99[2],2)))

pc1_pc2_plot = ggplot(data = value_99_df, aes(x = value_99_df$nr_component, y = value_99_df$value_99), l
  geom_bar(stat="identity") +
  ggtitle("Components explain more than 99% of the variation")+
  xlab("Comonents") +
  ylab("Explained variance") +
  geom_text(data=value_99_df,aes(x=value_99_df$nr_component,y=value_99_df$value_99,label=value_99_df$lab

# create the plot
pc1_pc2_plot
# pca loadings (U)
U = res$rotation


# Score of PC1 and PC2
# create a data frame
pc1_pc2 = as.data.frame(res$x[,1])
pc1_pc2 = cbind(pc1_pc2,res$x[,2])
colnames(pc1_pc2) = c("pc1","pc2")

pc1_pc2_score = ggplot(data = pc1_pc2) +
  geom_point(aes(x = pc1_pc2$pc1, y = pc1_pc2$pc2)) +
  xlab("PC1") +
  ylab("PC2") +
  ggtitle("Scores of PC1 and PC2")

# create the plot
pc1_pc2_score
# plot trace plot
plot(U[,1], main="Traceplot, PC1")
plot(U[,2],main="Traceplot, PC2")
set.seed(12345)
# https://rdrr.io/cran/ica/man/icafast.html
# S - Matrix of source signal estimates (S=Y%*%R).
# M - Estimated mixing matrix.
# W - Estimated unmixing matrix (W=crossprod(R,Q)).
# Y - Whitened data matrix.
# Q - Whitening matrix.
# R - Orthogonal rotation matrix.

X = as.matrix(pca_data)
ica = fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
              method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001, verbose = TRUE) #ICA

# compute W´ = K * W
w_line = ica$K %*% ica$W

# plot trace - step 2
plot(w_line[,1], main="Traceplot, PC1")
plot(w_line[,2], main="Traceplot, PC2")
```

```r
# plot of frist 2 latent features - compare with step 1
latent_features = as.data.frame(ica$S[,1]) # ica$s = score values
latent_features = cbind(latent_features,ica$S[,2])
colnames(latent_features) = c("feature1", "feature2")


ggplot(data = latent_features) +
  geom_point(aes(x = latent_features$feature1, y = latent_features$feature2)) +
  xlab("Feature 1") +
  ylab("Feature 2") +
  ggtitle("Scores of first two latent features")
```