Final Exam

The file `nba_logreg.csv` contains the statistics that `1329` players got during their first year in the NBA. The goal of this review is to develop a system that, in the future, will discover those players who will play more than 5 years in the NBA.

Important:

- As some methods may be time consuming, make a logical selection of the possible parameter values.
- Exercises should be solved using only the R packages that have been seen either in master class or in a small group. The use of any other package will have a penalty.

The file contains the following variables:

`Name`: Name of the player	`FTM`: Free Throw Made
`GP`: Games Played	`FTA`: Free Throw Attempts
`MIN`: Minutes Played	`FT%`: Free Throw Percent
`PTS`: Points Per Game	`OREB`: Offensive Rebounds
`FGM`: Field Goals Made	`DREB`: Defensive Rebounds
`FGA`: Field Goal Attempts	`REB`: Rebounds
`FG%`: Field Goal Percent	`AST`: Assists
`3P Made`: 3 Points Made	`STL`: Steals
`3PA`: 3 Points Attempts	`BLK`: Blocks
`3P%`: 3 Point Attempts	`TOV`: Turnovers

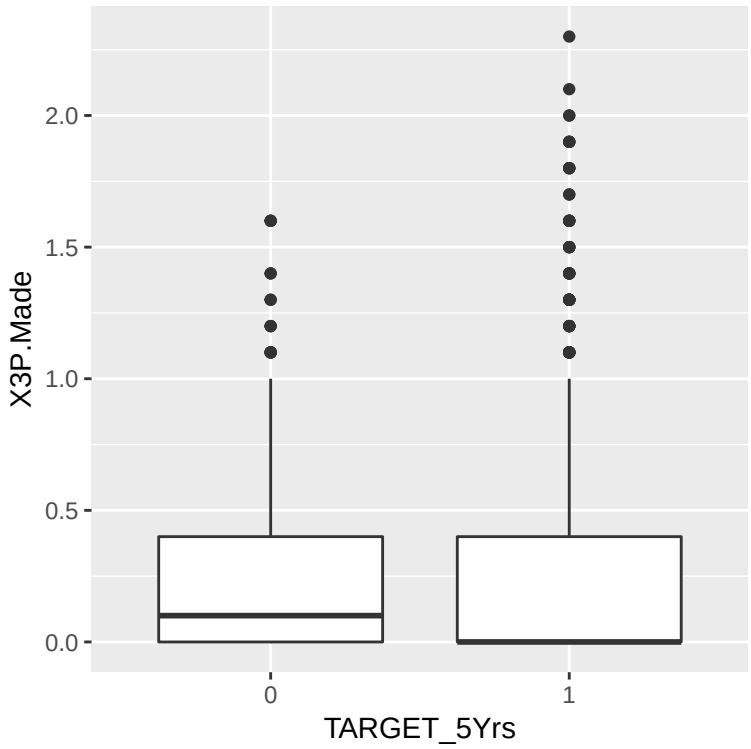
`TARGET_5Yrs` (dependent variable): 1 if career >= 5 years, 0 otherwise

Question 1

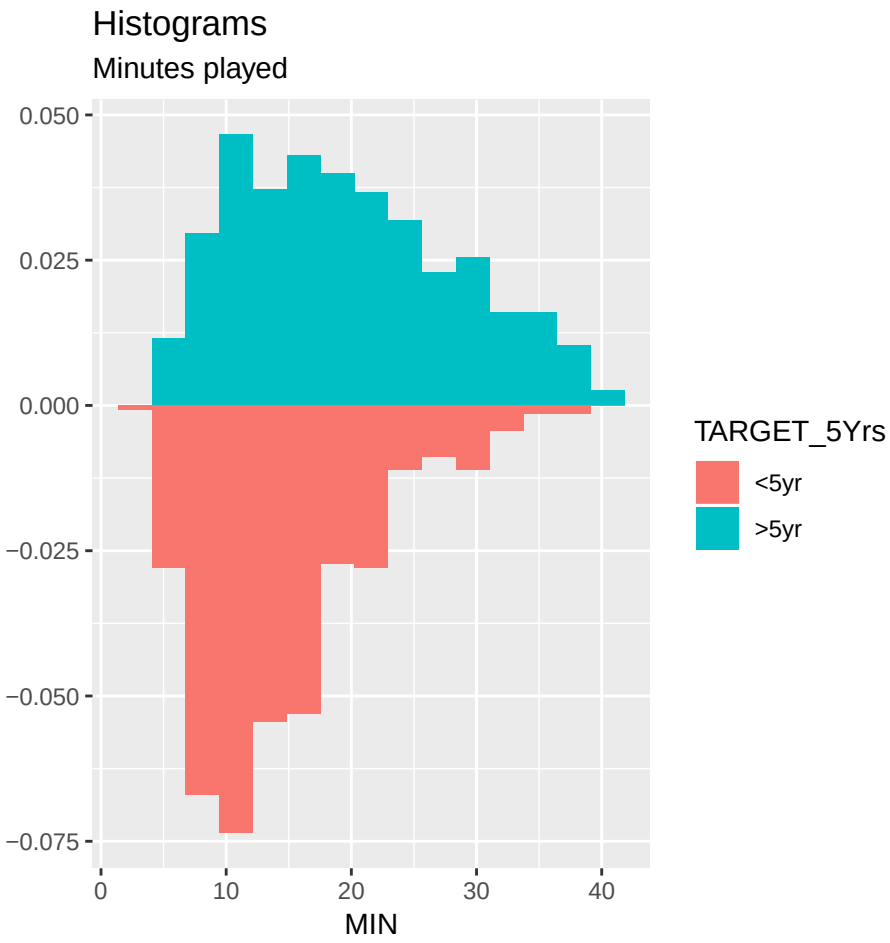
Write a code in R, using the `ggplot` package that produces the following displays. (1.5 points)

```
library(ggplot2)
data = read.csv("nba_logreg.csv", sep = ";", dec = ".")
data$TARGET_5Yrs = factor(data$TARGET_5Yrs)

ggplot(data=data)+aes(x=as.factor(TARGET_5Yrs),y=X3P.Made)+geom_boxplot()+
  xlab("TARGET_5Yrs")
```




```
ggplot()+
  geom_histogram(aes(x = data$MIN[data$TARGET_5Yrs==1],
                     y = ..density.., fill = ">5yr"), bins = 15)+
  geom_histogram(aes(x = data$MIN[data$TARGET_5Yrs==0],
                     y = -..density.., fill = "<5yr"), bins = 15)+
  labs(title = "Histograms", subtitle = "Minutes played",
       fill = "TARGET_5Yrs", x = "MIN", y = "")
```




```
ggplot(data) +
  aes( x = OREB, y = DREB, color = TARGET_5Yrs)+
  geom_point()
```

Question 2

Describe the Principal Component Analysis technique, providing as much detail as you can and emphasizing one of its main applications (i.e., an example in which it is often used). (1 point)

Solution

Principal component analysis is a ***dimension reduction*** *technique, very effective in reducing a large number of variables related to each other to a few latent variables, trying to lose the minimum amount of information. The new latent variables obtained (the principal components), which are a* ***linear transformation*** *of the original variables, are uncorrelated (****orthogonal****) and ordered in such a way that the first components capture* ***most of the variation*** *present in all the original variables.*

Question 3

a. Build a training data set, named `data_tr`, that contains the first `1000` observations and a test set, `data_tst`, that contains the remaining `329` observations. Hint: Remember that variable `TARGET_5Yrs` must be a factor. (0 points)

```
data = read.csv("nba_logreg.csv", sep = ";", dec = ".")
data$Name = NULL
data$TARGET_5Yrs = factor(data$TARGET_5Yrs)
```

```
data_tr = data[1:1000,]
data_tst = data[1001:1329,]
```

b. Implement a decision tree, using `data_tr`, that maximizes the correct classification rate (`ccr`) of the set `data_tst`. Choose the parameters that maximize this `ccr`. (1 point)

```
library(rpart)

# params:
minsplit = c(2, 5, 10, 20, 50, 100, 200)
minbucket = c(2, 5, 10, 20, 50, 100, 200)
params = expand.grid(minsplit=minsplit, minbucket=minbucket)

result = data.frame(params, ccr = 0)
for (run in 1:nrow(result)) {
      tree = rpart(TARGET_5Yrs~., data = data_tr,
                   control = rpart.control(minsplit = result[run,1],
                                           minbucket = result[run,2]))
      pred = predict(tree, data_tst, type = "class")
      result$ccr[run] = mean(pred == data_tst$TARGET_5Yrs)
}

best = which.max(result$ccr)
result[best,]
```

c. Visualize the decision tree that you have built. (0.5 points)

```
library(rpart.plot)
tree = rpart(TARGET_5Yrs~., data = data_tr,
                   control = rpart.control(minsplit = result[best,1],
                                           minbucket = result[best,2]))
rpart.plot(tree)
```

d. Implement a random forest, using `data_tr`, that maximizes the `ccr` of the set `data_tst`. Choose the parameters that maximize this `ccr`. (1 point)

```
library(randomForest)

# params:
ntree = c(20, 50, 100, 500)
mtry = c(3, 5, 10, 15)
params = expand.grid(ntree= ntree, mtry = mtry)

result = data.frame(params, ccr = 0)
for (run in 1:nrow(result)) {
      forest = randomForest(TARGET_5Yrs~., data = data_tr,
                            ntree = result[run,1],
                            mtry = result[run,2])
      pred = predict(forest, data_tst, type = "class")
      result$ccr[run] = mean(pred == data_tst$TARGET_5Yrs)
}

best = which.max(result$ccr)
result[best,]
```

e. Implement a k-nearest neighbors, using `data_tr`, that maximizes the `ccr` of the set `data_tst`. Choose

the parameters that maximize this `ccr`. (1 point)

```
data_trX = data_tr
class_tr = data_trX$TARGET_5Yrs
data_trX$TARGET_5Yrs = NULL
data_trX = scale(data_trX)

data_tstX  = data_tst
class_tst = data_tstX$TARGET_5Yrs
data_tstX$TARGET_5Yrs =NULL
data_tstX= scale(data_tstX, center = attr(data_trX, "scaled:center"),
                            scale =  attr(data_trX, "scaled:scale"))

library(class)
# params
k = c(3, 5, 7, 10, 20, 50)
params = expand.grid(k= k)

result = data.frame(params, ccr = 0)
for (run in 1:nrow(result)) {
      pred = knn(data_trX, data_tstX, class_tr, k = result[run,1])
      result$ccr[run] = mean(pred == class_tst)
}

best = which.max(result$ccr)
result[best,]
```

f. Implement a support vector machine, using `data_tr`, that maximizes the `ccr` of the set `data_tst`. Choose the parameters that maximize this `ccr`. (1 point)

```
data_tr_sc = data.frame(data_trX, TARGET_5Yrs = class_tr)
data_tst_sc = data.frame(data_tstX, TARGET_5Yrs = class_tst)

library(e1071)
# params:
kernel = c("linear", "radial")
gamma = c(0.001, 0.01, 0.1, 0.2, 0.5, 1)
cost = c(0.01, 0.1, 0.5, 1, 2, 5, 10)

params = expand.grid(kernel = kernel, gamma = gamma, cost = cost)

result = data.frame(params, ccr = 0)
for (run in 1:nrow(result)) {
      fit = svm(TARGET_5Yrs~., data_tr_sc, kernel = result[run, 1],
                                           gamma = result[run, 2],
                                           cost = result[run, 3])
      pred = predict(fit, data_tst_sc, type = "class")
      result$ccr[run] = mean(pred == data_tst_sc$TARGET_5Yrs)
}

best = which.max(result$ccr)
result[best,]
```

Question 4

a. Explain in detail the k-means algorithm. That is, what problems does it tackle, how does it work, what parameters have to be supplied in order to obtain an optimal solution. (1 point)

Solution

*K-means is an **unsupervised clustering method** that partitions a set of N observations in groups such that each observation belongs to the group whose center is closest, in terms of the squared euclidean dissimilarity. The idea behind k-means clustering is obtaining clusters with the minimum WSS (within cluster sum of squares), which measures how compact clusters are. The algorithm is iterative, and the final solution depends on the **initial centers**. That is why it is often initialized with different seeds, and the best result is chosen.*

b. Write a script in R to perform color segmentation using k-means on the image `elektra.jpg` and visualize the result, as we did in Lab.4 with the Altamira picture. Explain why you have selected that number of centers. (1 point)

```
library(OpenImageR)

img = readImage("elektra.jpg")
imageShow(img)

img.vect = apply(img, 3, as.vector)

wss = rep(NA, 6)
for (c in 1:6) {
  km = kmeans(img.vect, centers = c, nstart = 25)
  wss[c] = km$tot.withinss
}
library(ggplot2)
ggplot() + aes( x = 1:6, y = wss) + geom_point() + geom_line() +
  labs(title="Elbow method")

# Could be 2 or 3 centers

km = kmeans(img.vect, centers = 3, nstart = 25)

img2 = km$centers[km$cluster,]
dim(img2) = dim(img)
imageShow(img2)
```

Question 5

Replace the following **for** loop to execute it using parallel programming to speed it up. Note: you cannot remove the `stupid_function` from the loop. (1 point)

```
  if (!require(tictoc))
    install.packages("tictoc")
  library(tictoc)

  stupid_function=function(){
   Sys.sleep(0.5)
  }
```

```
  tic()
  a=0
  for (i in 1:21){
    a[i]=sqrt(i)
    stupid_function()
  }
  toc()
```

Solution

```
library(foreach)
library(doParallel)
library(tictoc)

nodes = detectCores();
cl = makeCluster(ifelse(nodes>1, nodes - 1, 1))
registerDoParallel(cl)

tic()
centers = 1:21
result.parallel = foreach(i = centers, .combine = 'c')%dopar%{
  temp = sqrt(i)
  stupid_function()
  temp
}
toc()

stopCluster(cl)
```