Homework 2

Alberto De Benedittis

25/4/2021

```
library(plotmo)
library(glmnet)
library(FactoMineR)
library(corrplot)
library(psych)
library(factoextra)
library(tree)
library(randomForest)
library(gbm)
```

Exercise 1

Consider the “fat” dataset provided for this Homework (tab-separated fat.tsv). It contains percent body fat, age, weight, height and body circumference measurements for 252 male subjects. Our goal is to predict body fat (variable y in the dataset) from the other explanatory variables. Load the data and perform a first exploratory analysis

The first thing to is to import the the data set and analyzing it.

```
#grasso <-  read_tsv('fat.tsv')
grasso <-  read.csv('fat.tsv', sep = '\t')

dim(grasso)
```

```
## [1] 252  16
```

```
head(grasso)
```

```
##      y siri density age weight height neck chest abdomen   hip thigh knee ankle
## 1 12.6 12.3  1.0708  23 154.25  67.75 36.2  93.1    85.2  94.5  59.0 37.3  21.9
## 2  6.9  6.1  1.0853  22 173.25  72.25 38.5  93.6    83.0  98.7  58.7 37.3  23.4
## 3 24.6 25.3  1.0414  22 154.00  66.25 34.0  95.8    87.9  99.2  59.6 38.9  24.0
## 4 10.9 10.4  1.0751  26 184.75  72.25 37.4 101.8    86.4 101.2  60.1 37.3  22.8
## 5 27.8 28.7  1.0340  24 184.25  71.25 34.4  97.3   100.0 101.9  63.2 42.2  24.0
## 6 20.6 20.9  1.0502  24 210.25  74.75 39.0 104.5    94.4 107.8  66.0 42.0  25.6
##   biceps forearm wrist
## 1   32.0    27.4  17.1
## 2   30.5    28.9  18.2
## 3   28.8    25.2  16.6
## 4   32.4    29.4  18.2
```

```
## 5   32.2    27.7  17.7
## 6   35.7    30.6  18.8
```

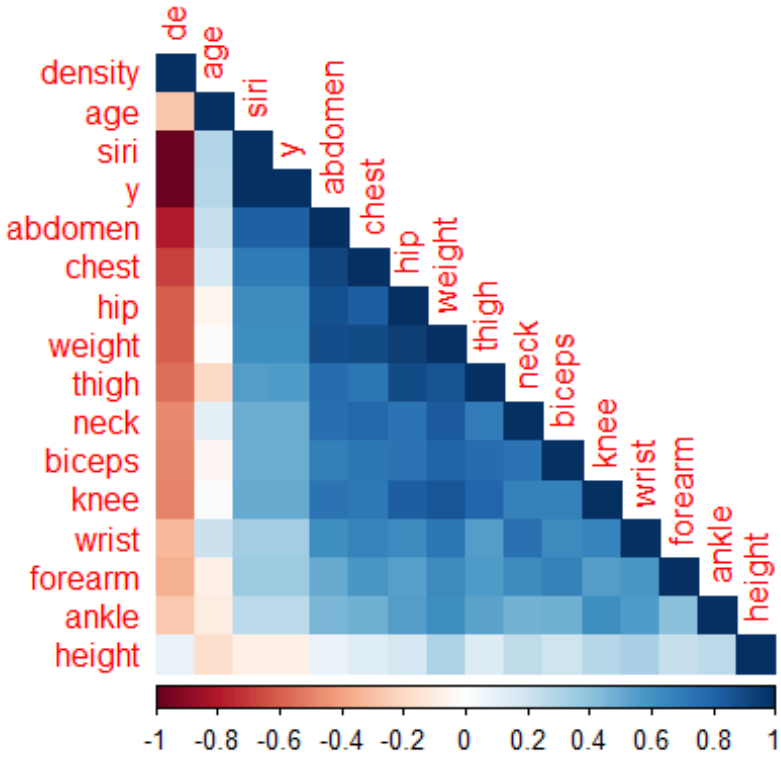
```
summary(grasso)
```

```
##        y              siri          density           age
##  Min.   : 0.00   Min.   : 0.00   Min.   :0.995   Min.   :22.00
##  1st Qu.:12.80   1st Qu.:12.47   1st Qu.:1.041   1st Qu.:35.75
##  Median :19.00   Median :19.20   Median :1.055   Median :43.00
##  Mean   :18.94   Mean   :19.15   Mean   :1.056   Mean   :44.88
##  3rd Qu.:24.60   3rd Qu.:25.30   3rd Qu.:1.070   3rd Qu.:54.00
##  Max.   :45.10   Max.   :47.50   Max.   :1.109   Max.   :81.00
##      weight          height          neck           chest
##  Min.   :118.5   Min.   :29.50   Min.   :31.10   Min.   : 79.30
##  1st Qu.:159.0   1st Qu.:68.25   1st Qu.:36.40   1st Qu.: 94.35
##  Median :176.5   Median :70.00   Median :38.00   Median : 99.65
##  Mean   :178.9   Mean   :70.15   Mean   :37.99   Mean   :100.82
##  3rd Qu.:197.0   3rd Qu.:72.25   3rd Qu.:39.42   3rd Qu.:105.38
##  Max.   :363.1   Max.   :77.75   Max.   :51.20   Max.   :136.20
##     abdomen            hip            thigh            knee
##  Min.   : 69.40   Min.   : 85.0   Min.   :47.20   Min.   :33.00
##  1st Qu.: 84.58   1st Qu.: 95.5   1st Qu.:56.00   1st Qu.:36.98
##  Median : 90.95   Median : 99.3   Median :59.00   Median :38.50
##  Mean   : 92.56   Mean   : 99.9   Mean   :59.41   Mean   :38.59
##  3rd Qu.: 99.33   3rd Qu.:103.5   3rd Qu.:62.35   3rd Qu.:39.92
##  Max.   :148.10   Max.   :147.7   Max.   :87.30   Max.   :49.10
##      ankle          biceps          forearm          wrist
##  Min.   :19.1   Min.   :24.80   Min.   :21.00   Min.   :15.80
##  1st Qu.:22.0   1st Qu.:30.20   1st Qu.:27.30   1st Qu.:17.60
##  Median :22.8   Median :32.05   Median :28.70   Median :18.30
##  Mean   :23.1   Mean   :32.27   Mean   :28.66   Mean   :18.23
##  3rd Qu.:24.0   3rd Qu.:34.33   3rd Qu.:30.00   3rd Qu.:18.80
##  Max.   :33.9   Max.   :45.00   Max.   :34.90   Max.   :21.40
```

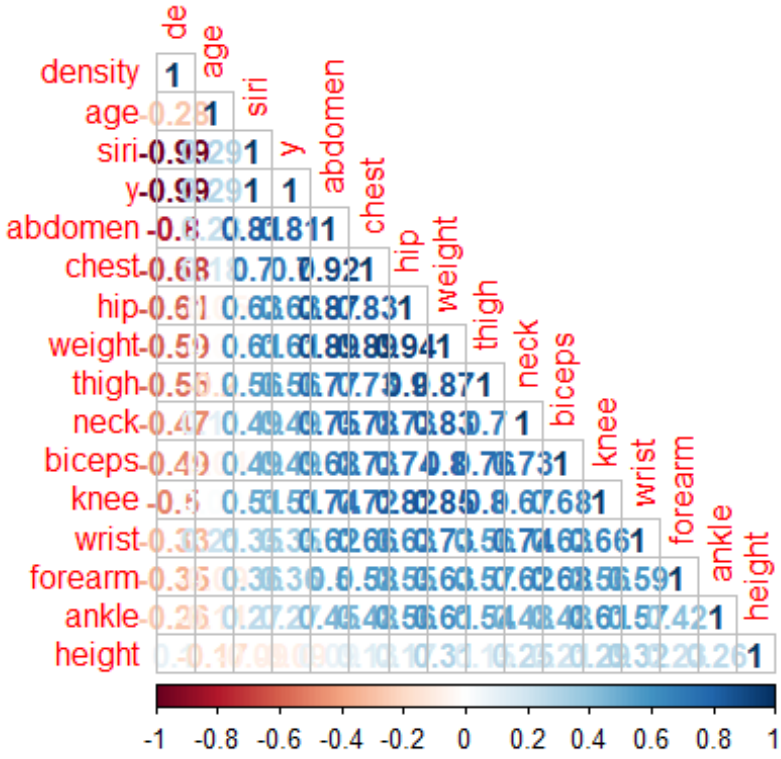
The data set is composed by 252 observation and 16 variables. Thanks to the command `summary()` we are able to see the main characteristic of our data set. The first good thing that we notice is that we do not have missing information (no *NAs*).

Another interesting thing that we can spot from this summary is that all the variables are continuous, and they have different scales. For example, the values of the category `density` range between 0 and 1 while the values of the category `weight` range between almost 100 and 300. Hence, we will probably need to standardize our variables in order to make better predictions. Now we want to investigate if some of the variables are correlated among each other. We expect that there will be an high level of correlation between the variable because all the predictors tend to spot characteristic related to people with an high percentage of body fat.

```
corp1 <- cor(grasso, use="complete", method = "pearson")
corrplot(corp1, method = "color", order = "AOE", type="lower")
```




```
corrplot(corp1, method = "number", order = "AOE", type="lower")
```



Here we have represented two correlation plots, they represent the same information but we have decided to plot the first graph because it is aesthetically pleasant while the second is more informative since it provides the actual value of the correlation between variables. These two plots give us useful and interesting information:

1) The variable `siri` is perfectly and positively correlated to the response variable `y`;

2) The variable `density` is almost perfectly and negatively correlated to the response variable `y`;

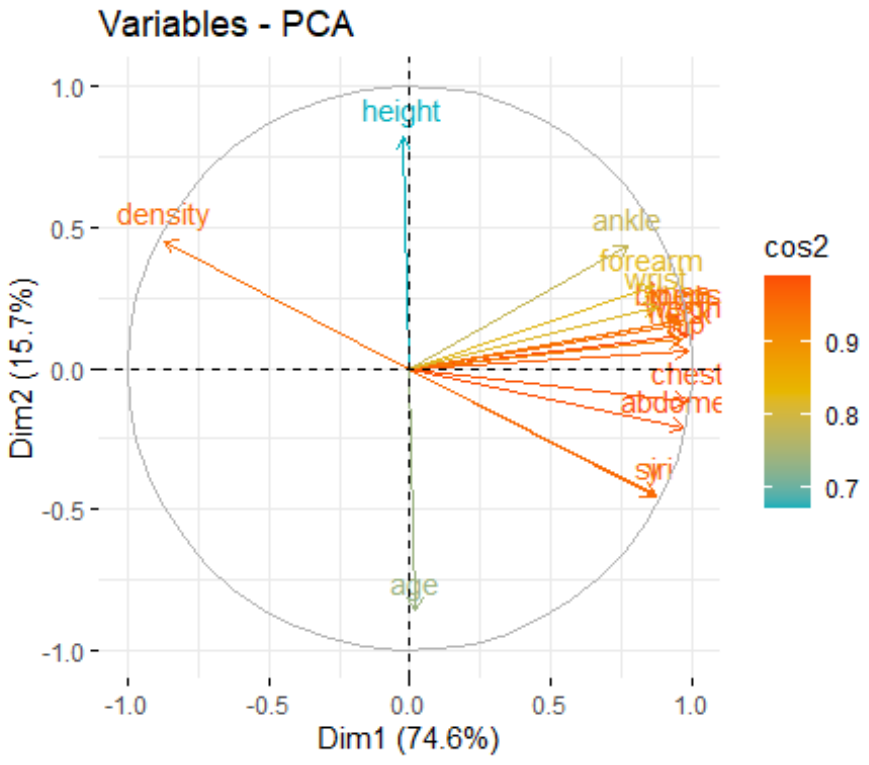
3) The variables `abdomen` and `chest` are also highly correlated to the response variable.

4) In general, there is a high degree of correlation among the variables.

This info is really useful and allows us to start to hypothesize which are the determinant variable for the prediction of y. We are quite sure that the variable `siri` will play an important role.

Now since we have many predictors we want to investigate if we can group some of the variables in some macro-categories. This is useful also because in this way we can better understand the previous statements regarding the high degree of correlation among the variables.

```
PCAdf1 <- PCA(corp1, scale =TRUE)

fviz_pca_var(PCAdf1, col.var = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07")
)
```



Here we have decided to plot our variables as we have to do a PCA. We did so because this helps us to visualize how the variables and their directions. Indeed, we confirm that `siri` and `y` go in the same direction (and have the same 'intensity') while `density` goes in the exact opposite direction. On the other hand, all the other variables tend to be concentrated in the upper right corner and so we can assume summarize the effect of all these variables as a unique one. Then there are the variables `age` and `height` that are less correlated to `y`.

Split the data into train/test

Now we have to create two new data sets: one will be the training data set that will be used to create our model while the other will be the validation set which will be used to test our models and understand how good are the predictions. To do so we need to call the function `sample()` to choose randomly half of the observations of the original data set (we choose half of the observations to constitute the train set and the validation set but we could have also consider a different proportion for the train set and the validation set, but split in two halves is a convention so we decide to follow it). When we call the function `sample()` we have to be careful because as said before it choose *randomly*. Hence, we need to set a seed in order to make this random operation repeatable.

```
set.seed(99)
train_ind <-  sample(nrow(grasso), nrow(grasso)*0.5)
train <-  grasso[train_ind,]
test <-  grasso[-train_ind,]
```

Perform least squares regression to predict y from the other variables. Discuss the results of the model and compute the test MSE.

Now, we want to make our first model: a linear regression model.

```
lm.fit <-  lm(y~., data = train)
summary(lm.fit)

##
## Call:
## lm(formula = y ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.05271 -0.05291  0.00326  0.04516  1.05250
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.276e+01  5.018e+00   2.542   0.0124 *
## siri         9.025e-01  1.083e-02  83.322   <2e-16 ***
## density     -1.093e+01  4.474e+00  -2.443   0.0162 *
## age          1.119e-04  1.933e-03   0.058   0.9540
## weight       1.404e-03  3.527e-03   0.398   0.6912
## height       3.678e-03  1.002e-02   0.367   0.7142
## neck         9.756e-03  1.396e-02   0.699   0.4861
## chest        2.393e-03  5.615e-03   0.426   0.6708
```

```
## abdomen     -8.012e-03  6.869e-03  -1.166   0.2460
## hip          3.315e-03  8.387e-03   0.395   0.6934
## thigh        8.233e-03  8.626e-03   0.954   0.3420
## knee        -3.052e-02  1.513e-02  -2.018   0.0461 *
## ankle        6.600e-04  1.291e-02   0.051   0.9593
## biceps      -2.142e-02  9.057e-03  -2.366   0.0198 *
## forearm      1.111e-02  1.019e-02   1.090   0.2782
## wrist        4.181e-02  2.998e-02   1.395   0.1659
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1722 on 110 degrees of freedom
## Multiple R-squared:  0.9996, Adjusted R-squared:  0.9995
## F-statistic: 1.849e+04 on 15 and 110 DF,  p-value: < 2.2e-16
```

The result of this linear model looks quite good. Indeed, we can consider the **Adjusted R-Squared** which is one of the criteria used to assess the quality of a model. Hence, we both recall the formulas of the **R-squared** and the one of the **Adjusted R-Squared**:

$$R^2 = 1 - RSS/TSS$$

(where TSS is the total sum of squares for the response);

$$AdjustedR^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$$

Looking at the formula we easily notice that having an $AdjustedR^2 = 0.9995$ means having a really good model. Looking more in the details of the summary we look at the coefficients and we notice that many of them are not significant.

Indeed, the most important predictors according to the p-value is `siri` (that should be a *body composition equation* that is used for calculating the percentage of body fat, indeed `siri` could be also defined by the following equation $\%bodyfat = (495/density) - 450$).

The second most important predictor is `density`. As we have seen in the PCA plot and during the analysis of the correlation among variables, density is negatively correlated to the body fat and hence it does not look strange to see it between the relevant predictors. Moreover, as we have just discovered, it is also used to estimate the percentage of body fat in the `siri` formula.

The third predictor with a low p-value is `biceps` that should indicate the circumference of the biceps. The last relevant predictor for our analysis is `knee` (which should refer to knee pain. Indeed, this look quite reasonable since people with an high percentage of fat typically has this kind of problem).

However, we know that the importance of the predictors could be biased due to the the way we have created our train set and validation set. This could be the case also because we have relatively few observation and an high number of predictors. Indeed, we won't be surprised if changing the seed will also change the relevant predictors (with the exception of `siri` which has a really low p-value). Hence, we could consider a different seed and see how the p-values of the predictors change. It would be interesting doing some hypothesis based on the data that we have. For example, many of the variables that we have refers to the circumferences of different

body parts such as chest and biceps. These predictors could be reasonably good estimators for detecting people with an high percentage of bodyfat. However, there could also be some outliers that could soften the predictive power of these variables. For example, athletes such as bodybuilders or football players could have really high values for the above mentioned variables but also a really low level of body fat but, we imagine that in a small sample as `train` it is rare to find an observation with these characteristic (high values for chest and biceps but low level of body fat).

```
bbt <-  train[train$chest > mean(train$chest) & train$biceps > mean(train$biceps)
& train$y < mean(train$y), ]
bb <- grasso[grasso$chest > mean(grasso$chest) & grasso$biceps > mean(grasso$bicep
s) & grasso$y < mean(grasso$y), ]
```

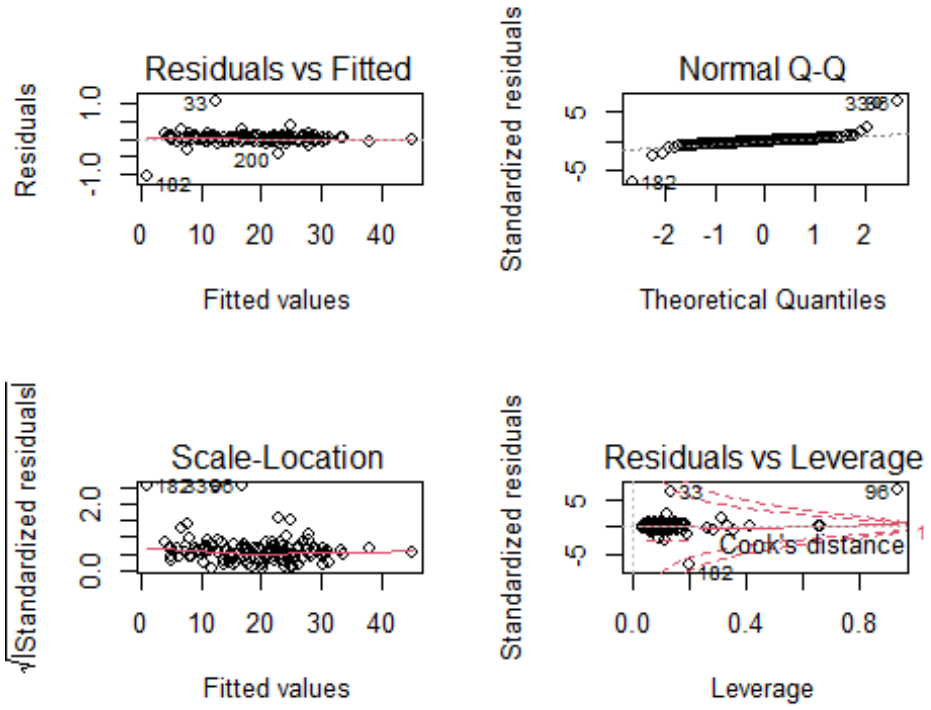
Although the previous one was just a reflection, we have found that actually there exist a small class of individuals with the following characteristic: circumferences of biceps and chest over the average and body fat under the average (8% of the people in the data set). This could be the reason why these predictors could be sometimes not very accurate to predict if a person given some characteristic has or not an high percentage of fat. Now we want to test some of the previous assumptions creating a new linear model with another random sample.

```
set.seed(11)
train_ind2 <-  sample(nrow(grasso), nrow(grasso)*0.5)
train2 <-  grasso[train_ind2,]
test2 <-  grasso[-train_ind2,]
lm.fit2 <-  lm(y~., data = train2)
summary(lm.fit2)

##
## Call:
## lm(formula = y ~ ., data = train2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.50175 -0.02582  0.01310  0.05122  0.11971
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.1023837  4.8882580   0.430   0.6680
## siri         0.9213026  0.0096314  95.656   <2e-16 ***
## density     -0.5578471  4.3564853  -0.128   0.8983
## age          0.0001430  0.0009773   0.146   0.8839
## weight       0.0018039  0.0021317   0.846   0.3993
## height      -0.0006204  0.0054039  -0.115   0.9088
## neck        -0.0049569  0.0081753  -0.606   0.5455
## chest       -0.0044850  0.0034874  -1.286   0.2011
## abdomen      0.0017432  0.0035656   0.489   0.6259
## hip          0.0067708  0.0055967   1.210   0.2290
## thigh       -0.0067091  0.0049025  -1.369   0.1739
## knee        -0.0108372  0.0079168  -1.369   0.1738
## ankle        0.0017709  0.0055564   0.319   0.7505
## biceps      -0.0122605  0.0064337  -1.906   0.0593 .
```

```
## forearm      0.0189737  0.0095830   1.980   0.0502 .
## wrist       -0.0048173  0.0169651  -0.284   0.7770
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09257 on 110 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9998
## F-statistic: 5.391e+04 on 15 and 110 DF,  p-value: < 2.2e-16
```

As we expected there are some differences between this linear model and the previous one. Firstly, here the only important predictor is `siri` and the others have high p-values or as for `biceps` and `forearm` close to the 5%. To conclude, this *excursus* was meant to explain how the results of a simple linear model could be affected by randomness and also to try to give some explanation regarding the results and to explain why some variables could not be considered or considered with caution. Now we briefly continue the analysis of the linear model plotting it.

```
par(mfrow = c(2,2))
plot(lm.fit)
```



Let's start by analyzing the *Residual vs Fitted*: this plot represents the residuals for each predicted value. The y values are spread around 0 for each value of x without showing any trend hence we can conclude that the model structure is probably correct.

The next plot is the *Normal Q-Q*. This is the QQ plot of residuals vs normal distribution. The points lie on a line so we can conclude that the assumption of normal distribution is not debatable.

Then we have the *scale location* plot. Here we have the standardized residuals vs the predicted values. From this plot we can deduce that the hypothesis of homoscedasticity may hold.

Lastly, we have the *residuals vs leverage*. From this plot we can detect if there are some dangerous outliers or not. We do not spot any warning situation

Now, we are curious to plot the regression line and together with the plot of `y` and `siri`.

```
plot(train$y ~ train$siri, main = '% of body fat Vs SIRI index', ylab = '% of body
fat', xlab = 'SIRI')
abline(lm.fit, col = 'red', lwd = 3)
```

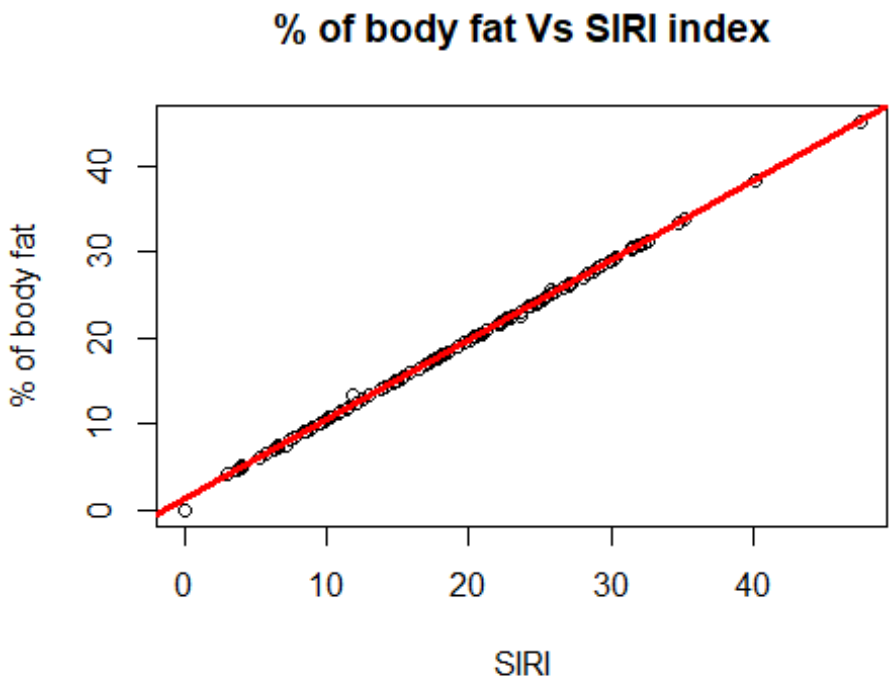
Although we are using only one out of the 16 predictors, we notice that there is a strong relation between `SIRI` and the percentage of body fat. Indeed, the most interesting thing is that the regression line is close and parallel to the distribution of the values of `y` in relation to `Siri`. Hence, we can justify the small shift with the presence of the other relevant predictors. Now we are curious and we want to further investigate the relation between `siri` and the percentage of body fat `y`.

```
lm.fit_only_siri <-  lm(y~ siri, data = train)
summary(lm.fit_only_siri)

##
## Call:
## lm(formula = y ~ siri, data = train)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -1.24367 -0.02487  0.00710  0.03413  1.25481
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.243669   0.038090   32.65   <2e-16 ***
## siri        0.923857   0.001792  515.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1759 on 124 degrees of freedom
## Multiple R-squared:  0.9995, Adjusted R-squared:  0.9995
## F-statistic: 2.659e+05 on 1 and 124 DF,  p-value: < 2.2e-16
```

Here we see the results of the linear regression when we consider only `siri` as predictor. The result confirms our hypothesis. Indeed, we notice that there is almost no change in the **Adjusted R-squared**. We now decide to plot this new regression line with the plot of `y` against `siri`.

```
plot(train$y ~ train$siri, main = '% of body fat Vs SIRI index', ylab = '% of body
fat', xlab = 'SIRI')
abline(lm.fit_only_siri, col = 'red', lwd = 3)
```



The result is impressive, the two lines are the same.

Now we are ready to see how the two models perform.

```
lm.pred <-  predict(lm.fit, test, type = 'response')
MSE.lm <- mean((lm.pred-test$y)^2)
MSE.lm

## [1] 0.03234367

lm.pred2 <-  predict(lm.fit_only_siri , test, type = 'response')
MSE.lm2 <- mean((lm.pred2-test$y)^2)
MSE.lm2

## [1] 0.03135333
```

In both cases we have obtained a quite low value for the mean standard error. We expected a result like this since we have previously obtained two models with high values for the **Adjusted R-squared**. We also notice that the model with only `siri` as predictor performs better than the one with all the predictors.

Fit a ridge regression model on the training set, choosing λ by cross-validation. Report the test error.

As we have seen so far, the linear model has distinct advantages in terms of inference and it is often competitive in relation to non linear methods. However, we want to try some methods that should help to improve the performance of the linear model by replacing plain least squares with some alternative fitting procedures. Indeed, we want to use the shrinkage methods. They reduce the prediction error by modifying the least squares criterion by constraining/shrinking the coefficients. This lead to an improvement in the model because in this way it is possible to reduce the variance. Let's first consider the **ridge regression**.

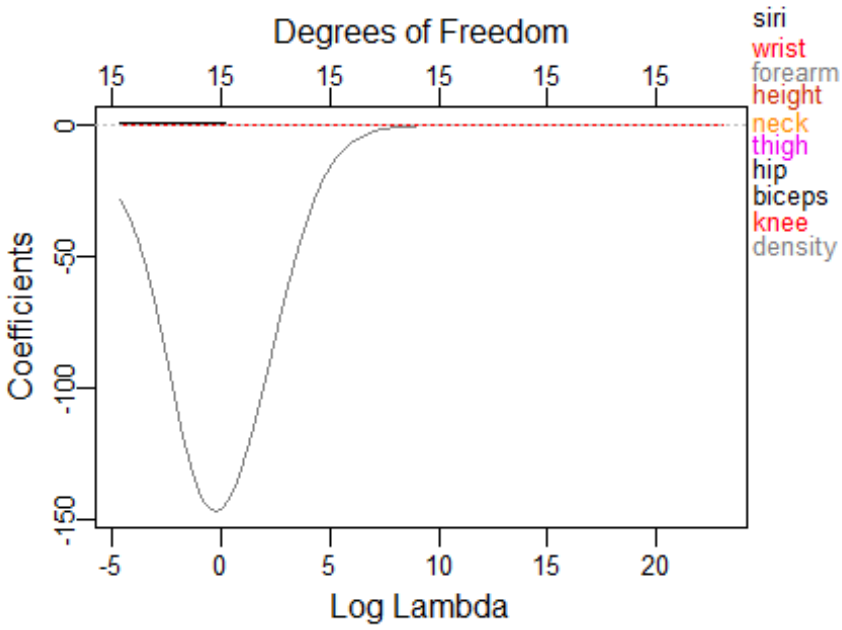
Ridge regression is similar to least squares but the coefficients are estimated by minimizing a slightly different quantity. Indeed, the *ridge regression coefficients estimates* $\widehat{\beta^R}$ are the values that minimize the following quantity:

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

where λ is our tuning parameter and the right right part of the formula is called *shrinkage penalty*. Obviously, when $\lambda = 0$ the ridge will simply produce the least squares estimates. Hence, the main task when we do a ridge regression model is to find the best value of λ.

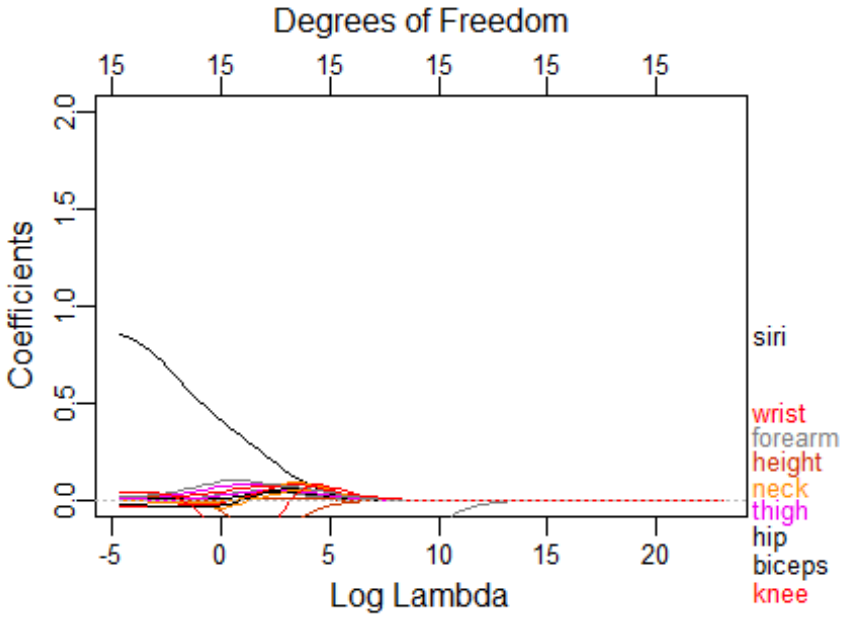
```
# ridge regression convert to model matrices
x_train <-  model.matrix(y ~., train)[,-1] # the shrinkage penalty is applied to b
eta1,....betap but not to the intercept beta 0
x_test <-  model.matrix(y ~., test)[,-1]
# labels
y_train <- train$y
y_test <-  test$y

library(glmnet)
grid <-  10^seq(10, -2, length =100) # We have chosen to implement the function ov
er a grid of values ranging from lambda = 10^10 to lambda = 10^-2, essentially cov
```

ering the full range of scenarios from the null model containing only the intercep
t, to the least squares fit.

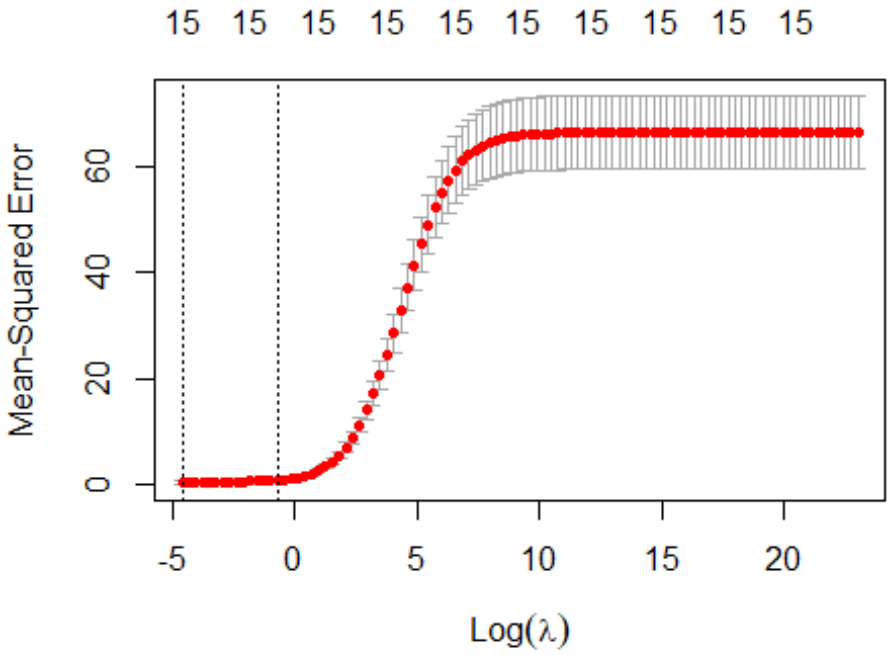



```
ridge.mod <-  glmnet(x_train,y_train,alpha = 0, lambda = grid)
plot_glmnet(ridge.mod, xvar = 'lambda')
```

```r
plot_glmnet(ridge.mod, xvar = 'lambda', ylim = c(0,2)) # zooming to better visuali
ze the different variables
```



```
rig.fit <-  cv.glmnet(x_train, y_train, alpha = 0, lambda = grid)
plot(rig.fit)
```




```
# optimal lambda
lambda <-  rig.fit$lambda.min
pred_ridge <-  predict(ridge.mod, s = lambda, newx = x_test)
MSE_ridge <- mean((y_test - pred_ridge)^2)
print(MSE_ridge)

## [1] 0.03744364
```

In this step of our analysis we have computed a ridge regression model. In a first instance, we have computed the ridge model considering different values of lambda in order to cover the full range of scenarios. Then we have plotted the first graph where we visualize the coefficients as a function of λ. From this plot we see that almost all the variables are close to zero expect `density`. Indeed, the value of this coefficients tends to decrease reaching is maximum at almost -150 and only when the value of λ is big it becomes close to zero. It is also interesting zooming the upper left corner of the plot. Here we can see all the variables in details. Firstly, we notice that no one of the variable is zero (as we expected). Secondly, we see that for small values of λ, the value of `siri`'s coefficient is bigger than the other predictors, a part from the already mentioned `density`. The following step is choosing the tuning parameter λ through cross-validation. We have created the third plot where we have the MSE against the log of lambda and in the upper part of the graph we have the number of predictors considered for each value of λ, this means that none of the predictors has been shrunken towards zero.

From the above plot we notice the the smallest value of the MSE is reached when $log(\lambda) = -2$ so, when $\lambda = 0.01$ which is the optimal value of the tuning parameter. Then we recompute the model with the best value for λ and we compute the MSE which is equal to 0.03744364 which is slightly higher than the one obtained with the prediction of the linear model.

Now that we have estimated the best lambda we refit our regression model on the full data set and we examine the coefficients estimated

```
x <-  model.matrix(y ~., grasso)[,-1]
y <-  grasso$y
out_full_mod <-  glmnet(x,y,alpha = 0)
predict(out_full_mod, type = 'coefficients', s = lambda)

## 16 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)  1.712479e+02
## siri         4.403569e-01
## density     -1.572258e+02
## age          1.510144e-02
## weight       1.274056e-03
## height      -2.101167e-02
## neck        -3.584354e-02
## chest        2.319826e-02
## abdomen      5.891176e-02
## hip          7.756010e-03
## thigh        2.248492e-02
## knee        -1.367942e-02
## ankle       -2.700741e-02
## biceps      -1.304734e-02
## forearm      4.818670e-02
## wrist       -1.404157e-01
```

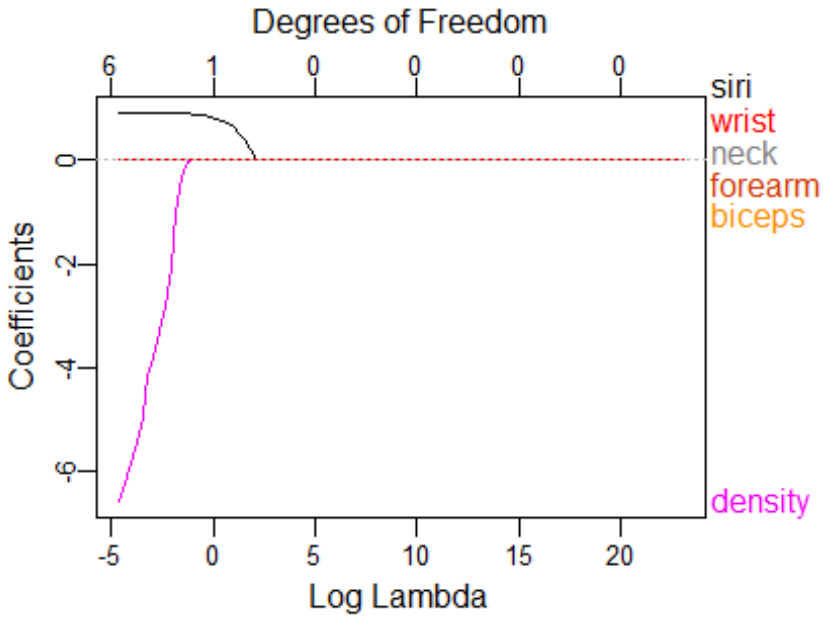
Looking at the different estimates of the coefficients as first thing we notice, as expected, that none of the coefficients are zero. Indeed, ridge regression does not perform variables selection. The second thing that we notice is that the coefficient with the highest value is `density`. This result confirms what we have discovered before: `density` is definitely correlated with `y`. The other predictor with an high value for its coefficient is `siri`, the other predictor that seems to reflect very well the response variable `y`. However, as a general comment we are satisfied because the most important predictors are still the same and this confirms our intuition until now. In addition we may think that for what concerns the other variables there could be an influence due to the seed.

Now we want to check whether the **lasso** can yield a more accurate or more interpretable model than ridge regression. Indeed, the **Lasso** is an alternative to the ridge. The lasso coefficients $\widehat{\beta^L}$ are the values that minimize the following formula:

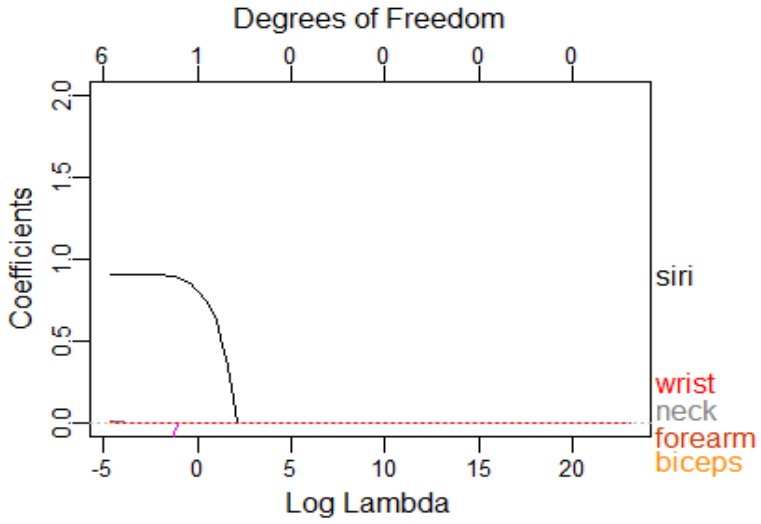
$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

As we can see the only difference between the lasso and the ridge is in the penalty. In the case of the lasso the penalty has the effect of forcing some of the coefficients estimates to be exactly equal to zero when λ is sufficiently large. From this we get two important information: firstly, the lasso does variable selection, secondly, depending on the value of lambda the lasso can produce a model involving any number of variables. Now we start creating our model.

```
lasso.mod <- glmnet(x_train, y_train, alpha = 1 , lambda = grid)
plot_glmnet(lasso.mod, xvar = 'lambda', label = T)
```

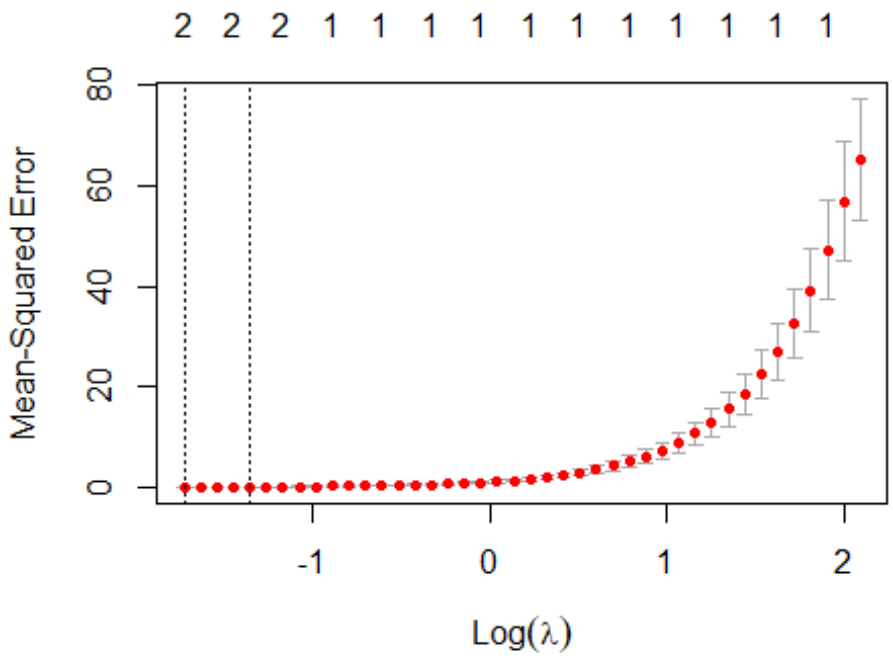


```r
plot_glmnet(lasso.mod, xvar = 'lambda', label = T, ylim = c(0,2)) # Zooming
```



As for the ridge we decide to plot the values of the coefficients against the value of $log\lambda$. We can see from the coefficient plot that depending on the choice of the tuning parameter, some of the coefficients will be exactly equal to zero. More specifically, we notice that just two variables are different from zero: `siri` and `density`. Also in this case we want ot find the best value of λ via cross-validation.

```
cv.out  <-  cv.glmnet(x_train, y_train, alpha = 1)
plot(cv.out)
```




```
bestlam <-  cv.out$lambda.min
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x_test)
MSE_lasso <- mean((lasso.pred - y_test)^2)
MSE_lasso
```

```
## [1] 0.06854537
```

From the plot we can see that the least value of the Mean-squared error is obtained with a value of $log\lambda = -0.8$ so when the value of $\lambda = 0.1474984$. Another interesting thing that we can spot from the plot is that we the lasso model considers one or two predictors at max which we expect to be `siri` and `density`. Lastly, we notice that the value of the MSE is increased. This is quite unexpected because when we apply the the shrinkage method we usually expect an increase in the performance and so a reduction of the MSE.

```
out <-  glmnet(x,y, alpha = 1, lambda = grid)
lasso.coef <-  predict(out, type = 'coefficients', s = bestlam)
lasso.coef
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  2.2007393
## siri         0.9036227
## density     -0.5374696
## age          .
## weight       .
```

```
## height       .
## neck         .
## chest        .
## abdomen      .
## hip          .
## thigh        .
## knee         .
## ankle        .
## biceps       .
## forearm      .
## wrist        .
```

From the analysis of the coefficients we notice the substantial difference between the ridge and the lasso: resulting coefficient are sparse. Indeed, the lasso model with λ chosen by cross validation contains only 2 variables `siri` and `density`, as expected.

Critically evaluate the results you obtained. If they look suspicious, think about a possible cause. For example, examine the coefficients of the least square regression model (estimate and sign), together with the R2 value; compute the pairwise correlations between the variables, . . . Think of a modification of the analysis in light of your findings and repeat steps 1-4 of your new analysis. Comment on the new results.

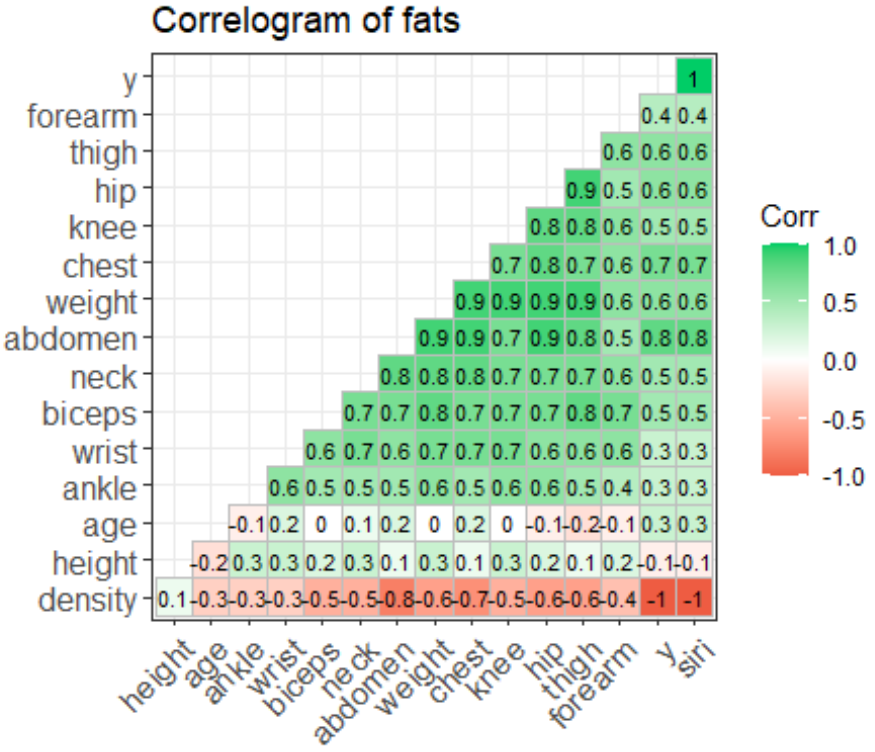
To recap we decide to create a small data frame where we store the results of our analysis to make it easier a comparison between the three different methods that we have used.

```
MSEs <- c(MSE.lm, MSE_ridge, MSE_lasso)
Model <-  c('Linear regression', 'Ridge regression', 'Lasso regression')
results <-  data.frame(Model, MSEs)
knitr::kable(results)
```

Model	MSEs
Linear regression	0.0323437
Ridge regression	0.0374436
Lasso regression	0.0685454

The results of the analysis look suspicious. Since the beginning of our analysis we were aware that something strange was going on. Indeed, what triggers us is that we expected an increase of the performance when we applied the shrinkage methods. Moreover, there is a relatively big difference between the MSE of the ridge and the one of the lasso and we typically expect that they will produce similar results (here the MSE of the lasso is almost twice the one of the ridge). In addition, from a theoretical point of view, the lasso typically outperforms the ridge when we have just few relevant predictors (and this should be the case). It looks reasonable believing that what causes all these problems was the variable `siri`. To better understand this we decide to produce a corrplot.

```
library(ggcorrplot)
data(grasso)
corr <- round(cor(grasso), 1)

# Plot
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
           method="square",
           colors = c("tomato2", "white", "springgreen3"),
           title="Correlogram of fats",
           ggtheme=theme_bw)
```



This beautiful plot shows us what we have already seen during our data exploration but, now we are more aware of what is going on. Indeed, we have an hypothesis:

the variable `siri` is equal to `y`.

As we said at the beginning `siri` is an indicator used to compute the percentage of body fat while `y` indicates the body fat. Hence, we can think that the dataset that we have analyzed could have been used by some nutritionists to see if `siri` is a good substitute for `y` or something like that, and according to the data it is. Thus, what looks reasonable is to redo our analysis but with a new dataset that does not include the variable `siri`. In this way we should be able to conduct an unbiased analysis. Lastly, before starting again we want to make an hypothesis based on our knowledge until now: we expect that the variable `density` would be an important one because it

was used to compute the value of `siri` and also because it should be an unbiased estimator (recalling the reasoning made at the beginning, detecting the % of body fat by measuring the circumferences of different body parts can be not reliable).

NEW START

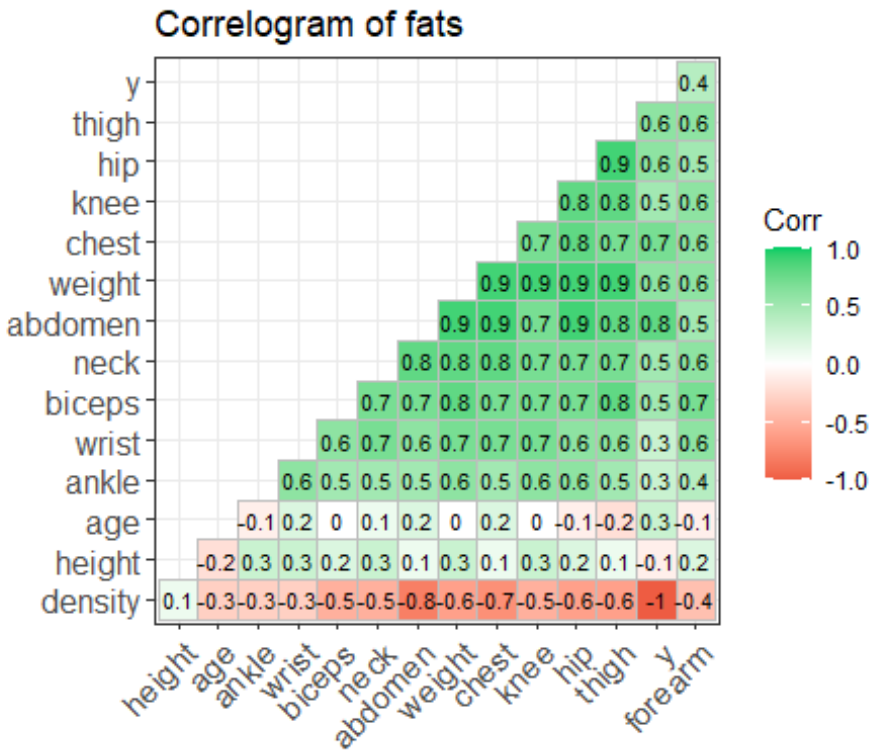
The first thing that we do is to create our new data set `grasso2` which is the same as before but without the column `siri`.

```
grasso2 <-  grasso[,c(-2)]
summary(grasso2)
```

```
##        y            density          age            weight
##  Min.   : 0.00   Min.   :0.995   Min.   :22.00   Min.   :118.5
##  1st Qu.:12.80   1st Qu.:1.041   1st Qu.:35.75   1st Qu.:159.0
##  Median :19.00   Median :1.055   Median :43.00   Median :176.5
##  Mean   :18.94   Mean   :1.056   Mean   :44.88   Mean   :178.9
##  3rd Qu.:24.60   3rd Qu.:1.070   3rd Qu.:54.00   3rd Qu.:197.0
##  Max.   :45.10   Max.   :1.109   Max.   :81.00   Max.   :363.1
##      height           neck           chest           abdomen
##  Min.   :29.50   Min.   :31.10   Min.   : 79.30   Min.   : 69.40
##  1st Qu.:68.25   1st Qu.:36.40   1st Qu.: 94.35   1st Qu.: 84.58
##  Median :70.00   Median :38.00   Median : 99.65   Median : 90.95
##  Mean   :70.15   Mean   :37.99   Mean   :100.82   Mean   : 92.56
##  3rd Qu.:72.25   3rd Qu.:39.42   3rd Qu.:105.38   3rd Qu.: 99.33
##  Max.   :77.75   Max.   :51.20   Max.   :136.20   Max.   :148.10
##       hip            thigh           knee            ankle          biceps
##  Min.   : 85.0   Min.   :47.20   Min.   :33.00   Min.   :19.1   Min.   :24.80
##  1st Qu.: 95.5   1st Qu.:56.00   1st Qu.:36.98   1st Qu.:22.0   1st Qu.:30.20
##  Median : 99.3   Median :59.00   Median :38.50   Median :22.8   Median :32.05
##  Mean   : 99.9   Mean   :59.41   Mean   :38.59   Mean   :23.1   Mean   :32.27
##  3rd Qu.:103.5   3rd Qu.:62.35   3rd Qu.:39.92   3rd Qu.:24.0   3rd Qu.:34.33
##  Max.   :147.7   Max.   :87.30   Max.   :49.10   Max.   :33.9   Max.   :45.00
##     forearm          wrist
##  Min.   :21.00   Min.   :15.80
##  1st Qu.:27.30   1st Qu.:17.60
##  Median :28.70   Median :18.30
##  Mean   :28.66   Mean   :18.23
##  3rd Qu.:30.00   3rd Qu.:18.80
##  Max.   :34.90   Max.   :21.40
```

Now we decide to show the corplot of this new data set.

```
corr <- round(cor(grasso2), 1)

# Plot
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
           method="square",
           colors = c("tomato2", "white", "springgreen3"),
           title="Correlogram of fats",
           ggtheme=theme_bw)
```



As we can see there is a strong negative relation between `y` and `density`. Now we want make some models and test them. Hence we create the train set and the validation set. To make the results as comparable as possible to the one made before we will use the same indexes to create the two sets.

```
train_gr2 <-  grasso2[train_ind,]
test_gr2 <-  grasso2[-train_ind,]
```

Now we create the linear model

```
lm_gr2 <-  lm(y~., data = train_gr2)
summary(lm_gr2)

##
## Call:
```

```
## lm(formula = y ~ ., data = train_gr2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5197 -0.6165 -0.2054  0.3351 12.1556
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  379.48858   19.20642  19.758   <2e-16 ***
## density     -356.78586   13.30199 -26.822   <2e-16 ***
## age            0.01397    0.01535   0.910    0.365
## weight        -0.01351    0.02808  -0.481    0.631
## height         0.08122    0.07950   1.022    0.309
## neck          -0.03393    0.11120  -0.305    0.761
## chest          0.06268    0.04439   1.412    0.161
## abdomen        0.07286    0.05420   1.344    0.182
## hip            0.03295    0.06679   0.493    0.623
## thigh         -0.02876    0.06867  -0.419    0.676
## knee          -0.02943    0.12058  -0.244    0.808
## ankle         -0.06543    0.10271  -0.637    0.525
## biceps        -0.07348    0.07202  -1.020    0.310
## forearm        0.09685    0.08082   1.198    0.233
## wrist          0.06266    0.23893   0.262    0.794
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.373 on 111 degrees of freedom
## Multiple R-squared:  0.9746, Adjusted R-squared:  0.9714
## F-statistic: 303.9 on 14 and 111 DF,  p-value: < 2.2e-16
```

The summary of the linear model shows exactly what we have supposed. Indeed, we see that the only reliable predictor is `density`. We also see that the value of the **Adjusted R Squared** is really high and this is a good indicator for our model. The next step of our analysis is to compute the MSE

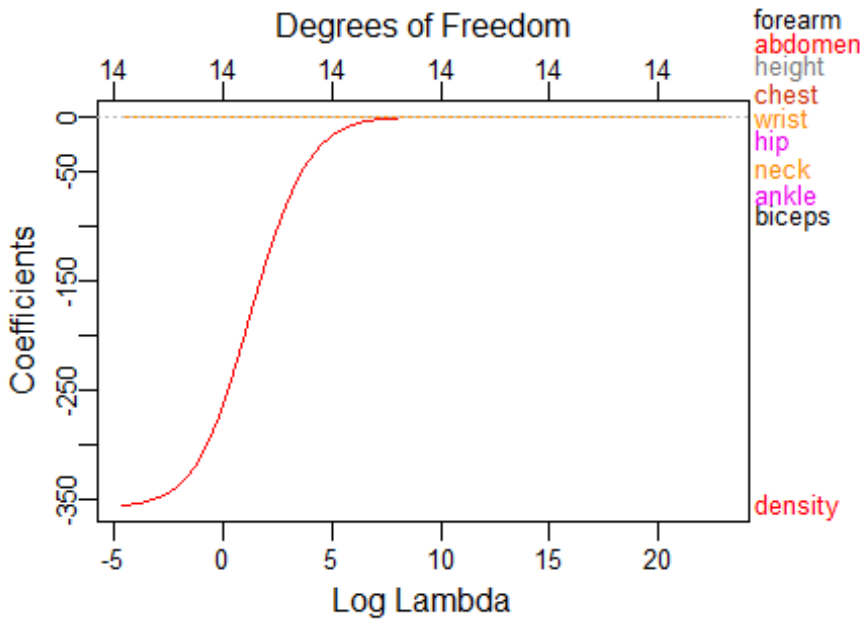
```
lm.pred2 <-  predict(lm_gr2, test_gr2, type = 'response')
MSE.lm2 <- mean((lm.pred2 - test_gr2$y)^2)
MSE.lm2
```

```
## [1] 1.293102
```

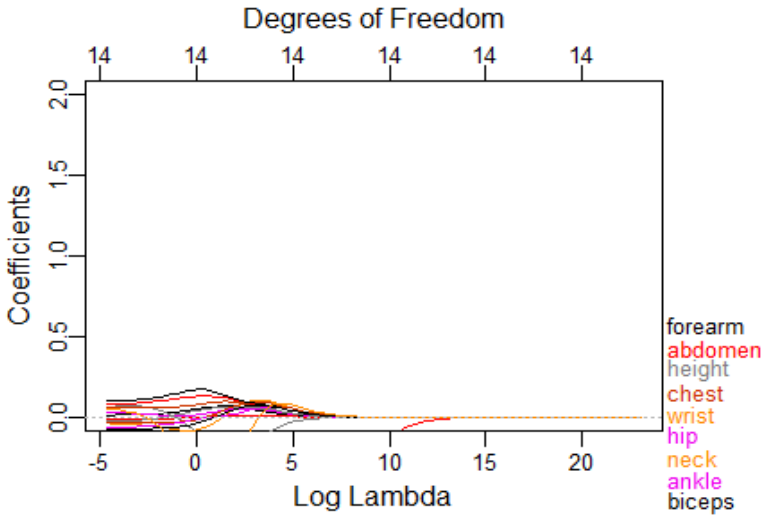
Now we want to compare this result with the one that we will obtain from the ridge regression and the lasso regression. This time we expect that there will be an improvement and so a reduction of the MSE with the shrinkage methods. We also hypothesize that the lasso will outperform the ridge. We think so because the lasso typically performs better in settings where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or equal to zero. Let's see if this intuition is right or not.

```
x_train2 <-  model.matrix(y ~., train_gr2)[,-1]
x_test2 <-  model.matrix(y ~., test_gr2)[,-1]
```

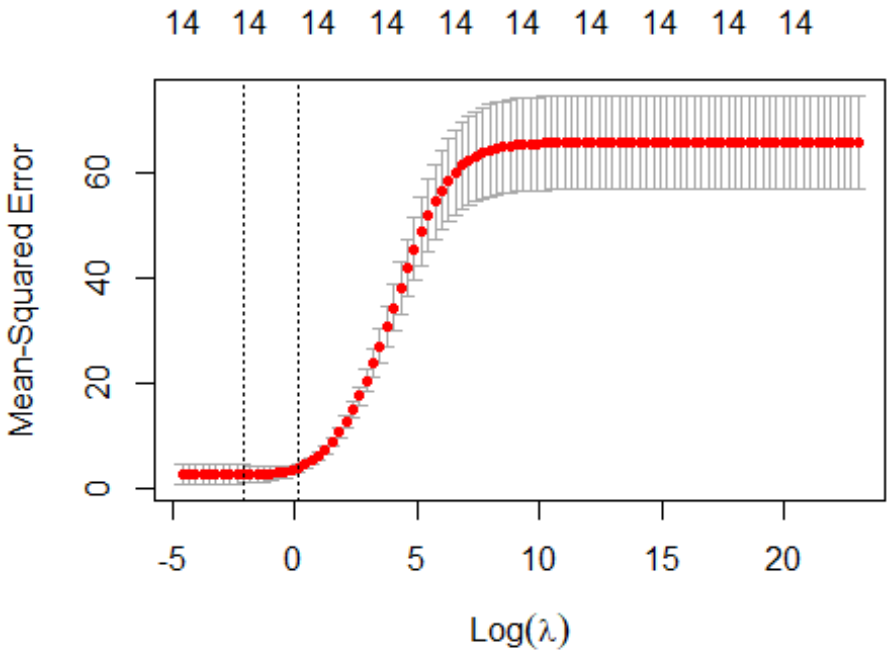
```r
y_train2 <- train_gr2$y
y_test2 <-  test_gr2$y

grid <-  10^seq(10, -2, length =100)
ridge.mod2 <-  glmnet(x_train2,y_train2,alpha = 0, lambda = grid)

plot_glmnet(ridge.mod2, xvar = 'lambda')
```




```r
plot_glmnet(ridge.mod2, xvar = 'lambda', ylim = c(0,2))
```



```
rig.fit2 <-  cv.glmnet(x_train2, y_train2, alpha = 0, lambda = grid)
plot(rig.fit2)
```




```
lambda2 <-  rig.fit2$lambda.min
pred_ridge2 <-  predict(ridge.mod2, s = lambda2, newx = x_test2)
MSE_ridge2 <- mean((y_test2 - pred_ridge2)^2)
print(MSE_ridge2)

## [1] 1.413394
```

The plot shows that for small values of lambda the only predictor that is not close to zero is density which then becomes close to zero for big values of lambda. We also see that the MSE computed with the ridge is close to the one of the linear model but it is slightly higher.

Now we want to take a look at the values of the coefficients

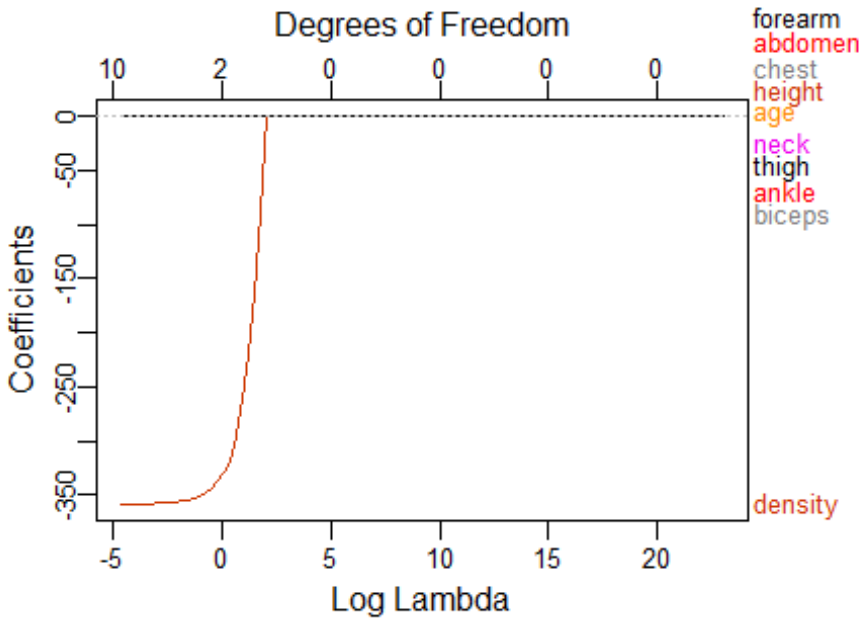
```
x2 <-  model.matrix(y ~., grasso2)[,-1]
y2 <-  grasso2$y
out_full_mod2 <-  glmnet(x2,y2,alpha = 0)
predict(out_full_mod2, type = 'coefficients', s = lambda2)

## 15 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  326.32439208
## density     -300.30885153
## age            0.02962150
```

```
## weight          0.00317344
## height         -0.04037713
## neck           -0.06671893
## chest           0.04502778
## abdomen         0.11053378
## hip             0.01460802
## thigh           0.03615197
## knee           -0.01423332
## ankle          -0.05471427
## biceps         -0.01889465
## forearm         0.08200456
## wrist          -0.28650324
```

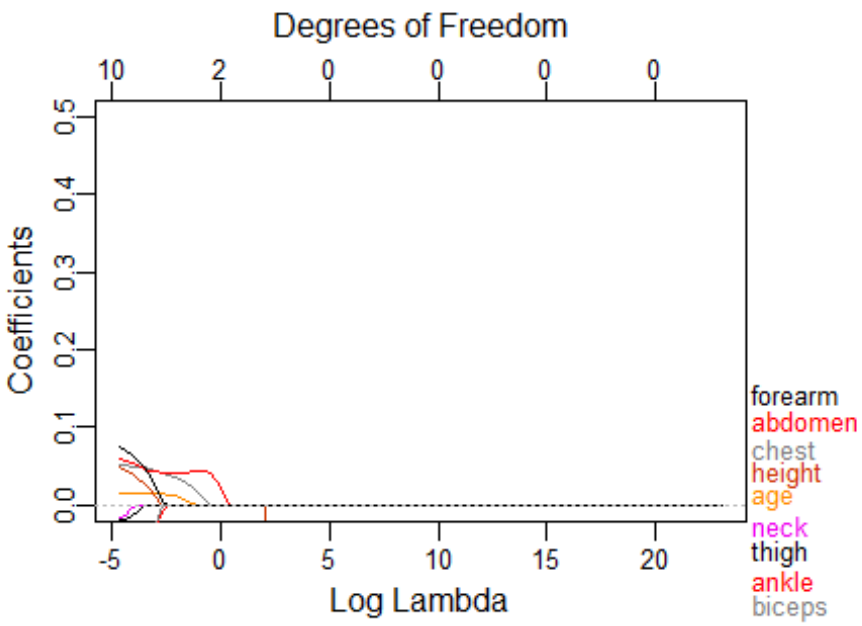
This results is another confirm of what we have said since now. Almost all the coefficients are close to zero a part from `density` which appears as the only important predictor for our analysis.

Now we want to see what happen when we apply the lasso regression.

```
lasso.mod2 <-  glmnet(x_train2, y_train2, alpha = 1 , lambda = grid)
plot_glmnet(lasso.mod2, xvar = 'lambda', label = T)
```

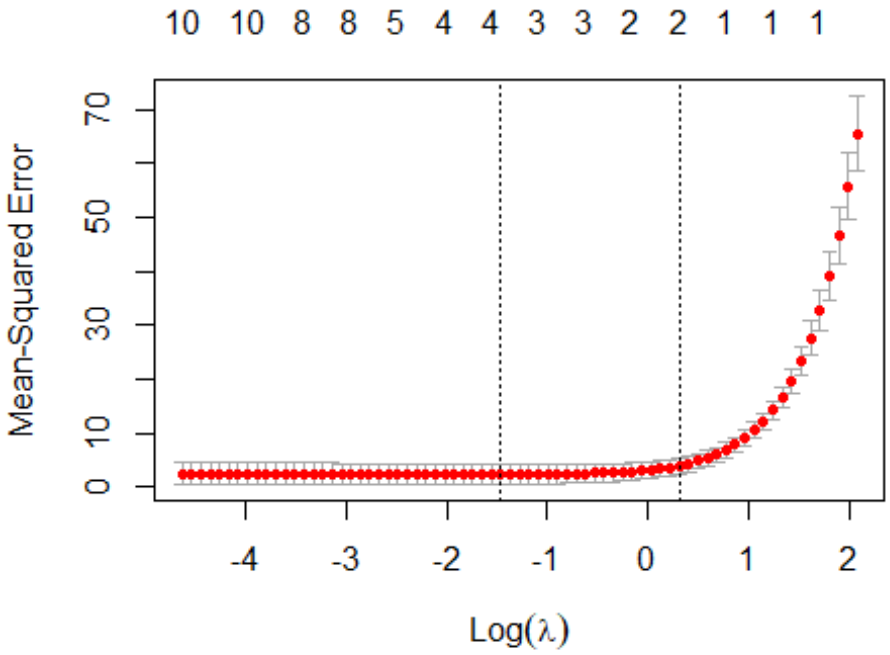


```
plot_glmnet(lasso.mod2, xvar = 'lambda', label = T, ylim = c(0,0.5))
```



Here we see that once again the unique predictor that is not close to zero for all the values of λ is `density`. From the zooming of the plot here we see that all the other variables are immediately close to zero and for slightly bigger values of λ they are equal to 0.

```
cv.out2  <-  cv.glmnet(x_train2, y_train2, alpha = 1)
plot(cv.out2)
```




```
bestlam2 <-  cv.out2$lambda.min
lasso.pred2 <- predict(lasso.mod2, s = bestlam2, newx = x_test2)
MSE_lasso2 <- mean((lasso.pred2 - y_test2)^2)
MSE_lasso2

## [1] 1.098913
```

Our intuition was correct. Indeed, we notice how the MSE of the lasso regression is lower than the one obtained with the ridge regression and also smaller than the one obtained with the linear model. Before we making a final comment we want to show the "importance" of the predictors for our analysis.

```
out2 <-  glmnet(x2,y2, alpha = 1, lambda = grid)
lasso.coef2 <-  predict(out2, type = 'coefficients', s = bestlam2)
lasso.coef2

## 15 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)  4.100673e+02
## density     -3.737525e+02
## age          .
## weight       .
## height       .
## neck         .
```

```
## chest        9.374596e-04
## abdomen      3.565425e-02
## hip          .
## thigh        .
## knee         .
## ankle        .
## biceps       .
## forearm      .
## wrist        .
```

From these we can see that almost all the predictors have been shrunken and are equal to 0 or almost 0 except from `density`.

FINAL COMMENT

We firstly sum up the results obtained creating a table.

```
MSEs_unbiased <- c(MSE.lm2, MSE_ridge2, MSE_lasso2)
Model <-  c('Linear regression', 'Ridge regression', 'Lasso regression')
results$MSEs_unbiased <-  MSEs_unbiased
knitr::kable(results)
```

Model	MSEs	MSEs_unbiased
Linear regression	0.0323437	1.293102
Ridge regression	0.0374436	1.413394
Lasso regression	0.0685454	1.098913

To conclude, the best model for analyzing the data set is the lasso made on the reduced model. To recap, we have to reduce the dataset because it does not have sense to predict a variable with a variable that is the practically the same (`y` and `siri` are the same thing). Indeed, the models created were almost perfect, we recall the first linear model has an **Adjusted R-squared** equal to 99. This also explains why the ridge and especially the lasso produced slightly worse results: there was no need to shrink the coefficients and this probably had a negative effects on the estimates. However, in all the methods with the full data set the MSE was really small. On the other hand, with the reduced data set the results are a bit different. The linear model is still a good way to predict `y`. This is because the variable `density`(which was also used to compute `siri`) is a really important predictor for estimating the percentage of body fat. Lastly, we have seen how the lasso outperforms both the ridge and the least squares because we are in the scenario where this model typically performs better hence, when there are just few (in this case only one) predictors that strongly affects the analysis.

Exercise 2

In this question, you will revisit the Hitters dataset. The goal is to predict the salary of baseball players, as a quantitative variable, from the other explanatory variables.

As usual, the first thing to do is importing the data set and exploring it.

```
library(ISLR)
baseball <-  ISLR::Hitters
dim(baseball)
```

```
## [1] 322  20
```

```
summary(baseball)
```

```
##      AtBat            Hits         HmRun            Runs       
##  Min.   : 16.0   Min.   :  1   Min.   : 0.00   Min.   :  0.00  
##  1st Qu.:255.2   1st Qu.: 64   1st Qu.: 4.00   1st Qu.: 30.25  
##  Median :379.5   Median : 96   Median : 8.00   Median : 48.00  
##  Mean   :380.9   Mean   :101   Mean   :10.77   Mean   : 50.91  
##  3rd Qu.:512.0   3rd Qu.:137   3rd Qu.:16.00   3rd Qu.: 69.00  
##  Max.   :687.0   Max.   :238   Max.   :40.00   Max.   :130.00  
##                                                                
##       RBI             Walks            Years            CAtBat       
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0  
##  1st Qu.: 28.00   1st Qu.: 22.00   1st Qu.: 4.000   1st Qu.:  816.8  
##  Median : 44.00   Median : 35.00   Median : 6.000   Median : 1928.0  
##  Mean   : 48.03   Mean   : 38.74   Mean   : 7.444   Mean   : 2648.7  
##  3rd Qu.: 64.75   3rd Qu.: 53.00   3rd Qu.:11.000   3rd Qu.: 3924.2  
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0  
##                                                                      
##      CHits            CHmRun           CRuns             CRBI        
##  Min.   :   4.0   Min.   :  0.00   Min.   :   1.0   Min.   :   0.00  
##  1st Qu.: 209.0   1st Qu.: 14.00   1st Qu.: 100.2   1st Qu.:  88.75  
##  Median : 508.0   Median : 37.50   Median : 247.0   Median : 220.50  
##  Mean   : 717.6   Mean   : 69.49   Mean   : 358.8   Mean   : 330.12  
##  3rd Qu.:1059.2   3rd Qu.: 90.00   3rd Qu.: 526.2   3rd Qu.: 426.25  
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.00  
##                                                                      
##      CWalks        League  Division    PutOuts          Assists      
##  Min.   :   0.00   A:175   E:157    Min.   :   0.0   Min.   :  0.0  
##  1st Qu.:  67.25   N:147   W:165    1st Qu.: 109.2   1st Qu.:  7.0  
##  Median : 170.50                    Median : 212.0   Median : 39.5  
##  Mean   : 260.24                    Mean   : 288.9   Mean   :106.9  
##  3rd Qu.: 339.25                    3rd Qu.: 325.0   3rd Qu.:166.0  
##  Max.   :1566.00                    Max.   :1378.0   Max.   :492.0  
##                                                                     
##      Errors          Salary       NewLeague
##  Min.   : 0.00   Min.   :  67.5   A:176    
##  1st Qu.: 3.00   1st Qu.: 190.0   N:146    
##  Median : 6.00   Median : 425.0            
```

```
##  Mean   : 8.04   Mean   : 535.9
##  3rd Qu.:11.00   3rd Qu.: 750.0
##  Max.   :32.00   Max.   :2460.0
##                  NA's   :59
```

As always, the summary of the data set gives us some useful information: the first thing that we notice is that there is some missing information for `salary`. Here we have two choices, that as always happen in life, involve a trade off: we can decide to remove the observation from the data set or to substitute the missing values with the mean or the average for that category. If we remove them we have less observation and this can negatively affect the analysis, also because we have many variables. On the other hand, if we substitute these values with the mean/median of the category we also risk to bias our data and consequently our analysis. To conclude, we decide to remove the missing observation because although we will have less observations, we know that there are some statistical techniques that can help us to create many random sub-samples in order to overcome the problems caused by small data set (e.g. bagging; boosting; random forests). Continuing our analysis, we look more in details our variables we notice that there are three categorical variables `NewLeague`, `League`, `Division`, all the others are continuous variables. Looking more in detail the variables, is not difficult to notice that they are on different scales. For example, the number of years in major league and the number of runs in the season are quite different: the number of years ranges between 1 and 24 while the numbers of runs ranges between 1 and 2165. This is an important factor that we should keep in mind for our analysis.

So, now we want to remove all the missing information and then we will consider the reduce data set for our analysis.

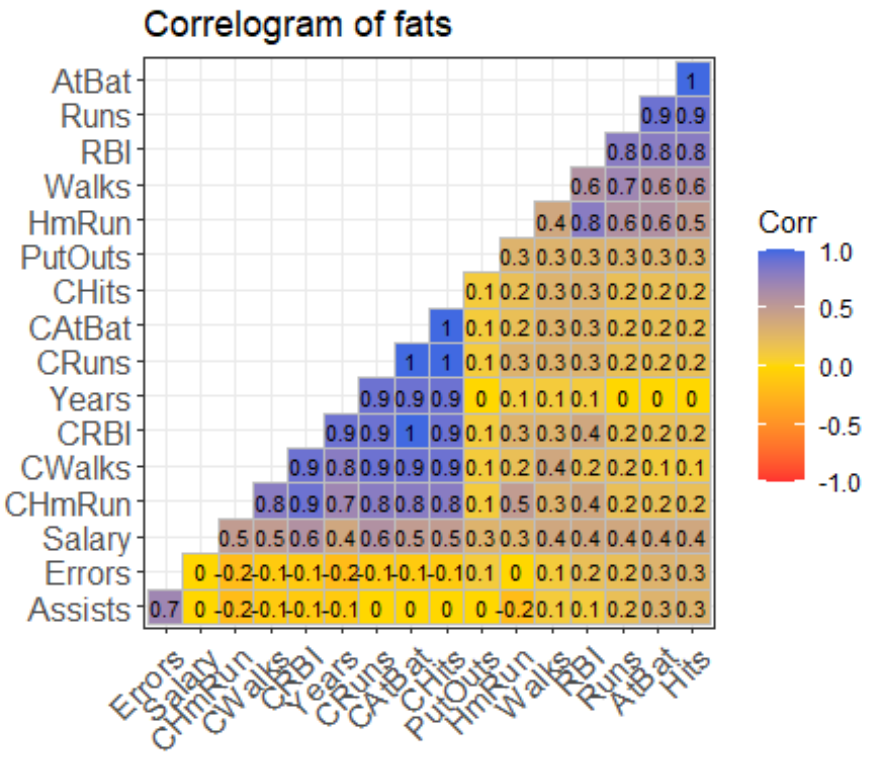
```
baseball <-  na.omit(baseball)
summary(baseball)
```

```
##      AtBat            Hits           HmRun            Runs
##  Min.   : 19.0   Min.   :  1.0   Min.   : 0.00   Min.   :  0.00
##  1st Qu.:282.5   1st Qu.: 71.5   1st Qu.: 5.00   1st Qu.: 33.50
##  Median :413.0   Median :103.0   Median : 9.00   Median : 52.00
##  Mean   :403.6   Mean   :107.8   Mean   :11.62   Mean   : 54.75
##  3rd Qu.:526.0   3rd Qu.:141.5   3rd Qu.:18.00   3rd Qu.: 73.00
##  Max.   :687.0   Max.   :238.0   Max.   :40.00   Max.   :130.00
##       RBI             Walks            Years            CAtBat
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
##  1st Qu.: 30.00   1st Qu.: 23.00   1st Qu.: 4.000   1st Qu.:  842.5
##  Median : 47.00   Median : 37.00   Median : 6.000   Median : 1931.0
##  Mean   : 51.49   Mean   : 41.11   Mean   : 7.312   Mean   : 2657.5
##  3rd Qu.: 71.00   3rd Qu.: 57.00   3rd Qu.:10.000   3rd Qu.: 3890.5
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
##      CHits            CHmRun           CRuns             CRBI
##  Min.   :   4.0   Min.   :  0.00   Min.   :   2.0   Min.   :   3.0
##  1st Qu.: 212.0   1st Qu.: 15.00   1st Qu.: 105.5   1st Qu.:  95.0
##  Median : 516.0   Median : 40.00   Median : 250.0   Median : 230.0
##  Mean   : 722.2   Mean   : 69.24   Mean   : 361.2   Mean   : 330.4
##  3rd Qu.:1054.0   3rd Qu.: 92.50   3rd Qu.: 497.5   3rd Qu.: 424.5
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.0
```

```
##      CWalks       League  Division    PutOuts          Assists
##  Min.   :   1.0   A:139   E:129    Min.   :   0.0   Min.   :  0.0
##  1st Qu.:  71.0   N:124   W:134    1st Qu.: 113.5   1st Qu.:  8.0
##  Median : 174.0                    Median : 224.0   Median : 45.0
##  Mean   : 260.3                    Mean   : 290.7   Mean   :118.8
##  3rd Qu.: 328.5                    3rd Qu.: 322.5   3rd Qu.:192.0
##  Max.   :1566.0                    Max.   :1377.0   Max.   :492.0
##      Errors           Salary       NewLeague
##  Min.   : 0.000   Min.   :  67.5   A:141
##  1st Qu.: 3.000   1st Qu.: 190.0   N:122
##  Median : 7.000   Median : 425.0
##  Mean   : 8.593   Mean   : 535.9
##  3rd Qu.:13.000   3rd Qu.: 750.0
##  Max.   :32.000   Max.   :2460.0
```

Now we want to investigate if there is an high degree of correlation among the variables of our data-set.

```
corr <- round(cor(baseball[,c(-14,-15,-20)]), 1)

# Plot
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
           method="square",
           colors = c("firebrick1", "gold", "royalblue"),
           title="Correlogram of fats",
           ggtheme=theme_bw)
```



This correlation plot gives us many useful pieces of information. The plot shows with different color the level of correlation among the continuous variables (the correlation plot does not accept categorical variables); in yellow are shown weak relations among the variable while in blue are represented the ones with an high degree of correlation . More specifically we spot a "little blue triangle" in the left part of our plot where there is an high degree of correlation among variables. The variables included in this area are the following: `CAtBat`, `CAtHits`, `CRuns`, `CHits`, `CRBI`, `CWalks`, `Years`, `CHRuns`. It is not difficult understand why these variables are correlated: they refer to the attack phase of a baseball match and the possible points that a player can do; they are correlated with the number of attempts that a player had. With this last consideration we end up the data exploration.

Split the data into training/test sets.

```
set.seed(99)
train_in <-  sample(nrow(baseball), nrow(baseball)*.5)
basetrain <- baseball[train_in,]
basetest <-  baseball[-train_in,]
```

As we did in the first exercise, we have split the data set in two halves, one will be used for the training the other will be instead used for the testing.

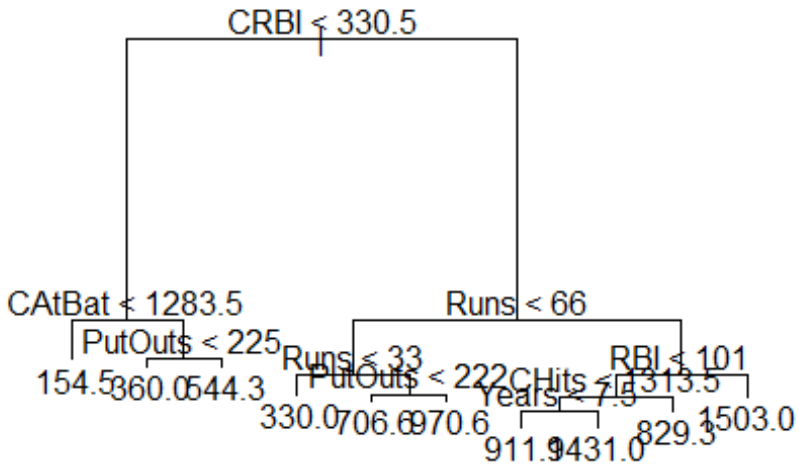
Fit a decision tree on the training data and plot the results. Choose the tree complexity by crossvalidation: plot the cross-validation deviance versus the number of terminal nodes and prune the tree if applicable. Finally, evaluate the optimal model by computing the test MSE.

Now we are asked to fit a regression tree to predict the salary of baseball players for the next year. The first thing that we do is to create a simple regression tree.

```
set.seed(99)
tree.baseball <-  tree(Salary ~., data = basetrain)
summary(tree.baseball)

##
## Regression tree:
## tree(formula = Salary ~ ., data = basetrain)
## Variables actually used in tree construction:
## [1] "CRBI"    "CAtBat"  "PutOuts" "Runs"    "RBI"     "CHits"   "Years"
## Number of terminal nodes:  10
## Residual mean deviance:  40470 = 4897000 / 121
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -603.400  -84.530   -6.615    0.000   56.180  909.100
```

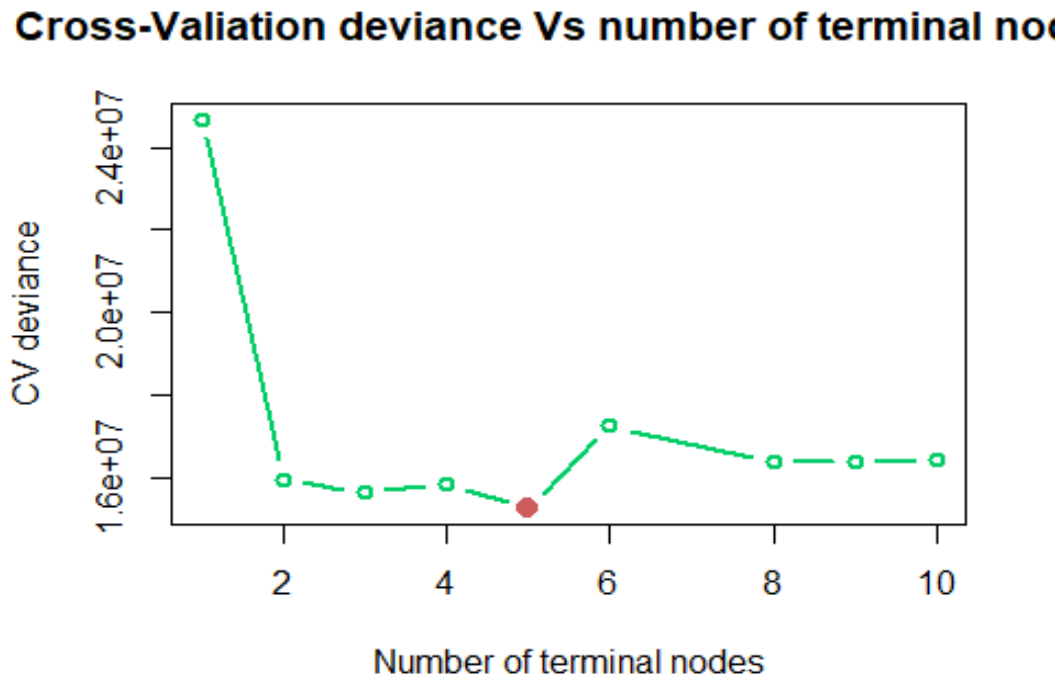
Let's now examine the output of the summary.It indicates that only seven variable have been used to construct the regression tree, the variables are the following: `CRBI`, `CAtBat`, `PutOuts`, `Runs`, `RBI`, `CHits` and `Years`. We also have information regarding the number of leaves: 10 and also about the the sum of squared error for the tree 40470. Now we decide to plot the regression tree; this allows us to take advantage of the characteristic that make the tree-based models competitive with other regression methods: they are easy to interpret and they provide nice graphical representation.

```
plot(tree.baseball)
text(tree.baseball, pretty = 0)
```



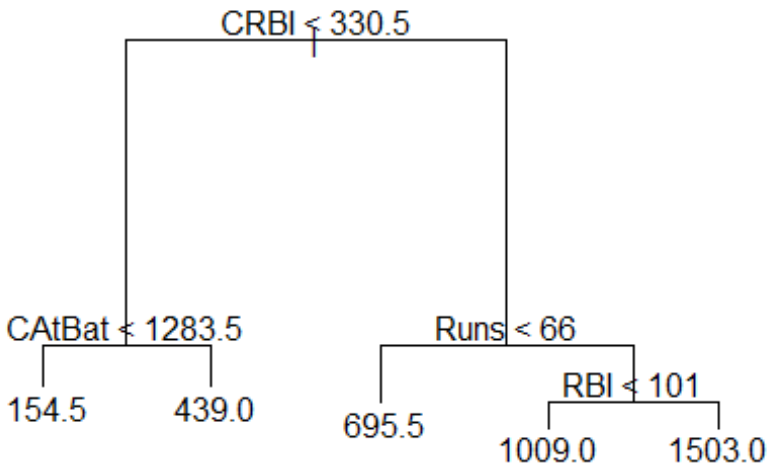
Now we have plotted the regression tree. This gives us a visual insight into the model. This representation is useful because we can visualize the terminal nodes, all the variable involved in the decision tree and the different splits. The first split involve the variable `CRBI`, that refers to the number of runs batted during the player's career, and so this could be considered as one of the most important predictor for our analysis. The variables used to create this regression tree have something in common: they refer to the scoring system used in baseball plus the variable `year`. Although this result is inaccurate, it shows that the next-season salary is related to the performance of the previous seasons. Now we are asked to choose the tree complexity by cross-validation and to plot the cross-validation deviance versus the number of terminal nodes.

```
set.seed(99)
cv.baseball <- cv.tree(tree.baseball)
plot(cv.baseball$size, cv.baseball$dev, type = 'b', col = 'springgreen3', main = '
Cross-Valiation deviance Vs number of terminal nodes', xlab = 'Number of terminal
nodes', ylab = 'CV deviance', lwd = 2)
bestpoint <- min(cv.baseball$dev)
cord <-  which.min(cv.baseball$dev)
points(cv.baseball$size[cord], bestpoint, col = 'indianred', cex = 2, pch=20)
```



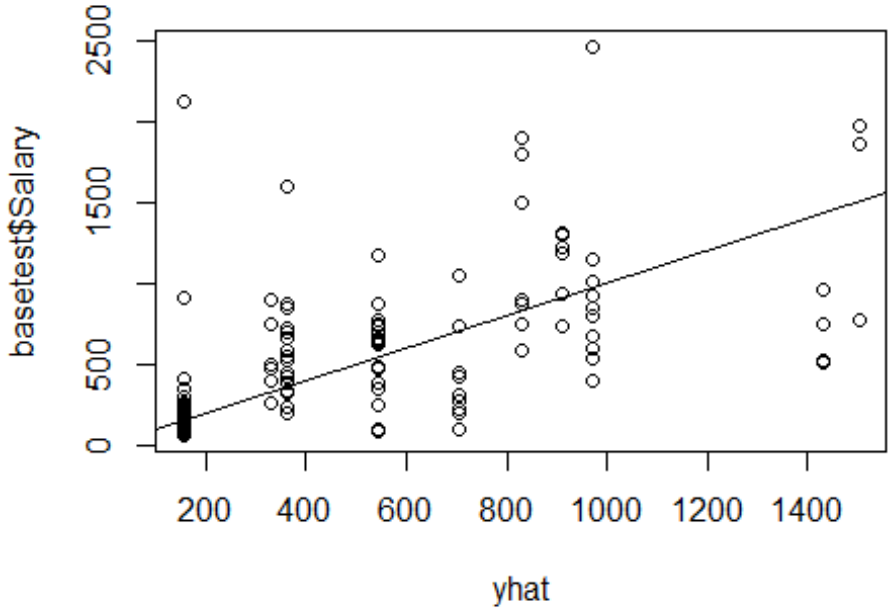
The above plot, *Cross-validation deviance vs number of terminal nodes*, shows that the tree with 5 terminal nodes, is selected by cross-validation. However, if we wish to prune the tree, we can also do as follows:

```r
set.seed(99)
prune.baseball <- prune.tree(tree.baseball, best = 5)
summary(prune.baseball)

##
## Regression tree:
## snip.tree(tree = tree.baseball, nodes = c(5L, 14L, 6L))
## Variables actually used in tree construction:
## [1] "CRBI"   "CAtBat" "Runs"   "RBI"
## Number of terminal nodes:  5
## Residual mean deviance:  59830 = 7538000 / 126
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -603.40  -92.26  -14.00    0.00   67.25  930.70

plot(prune.baseball)
text(prune.baseball, pretty = 0)
```



In the above plot we have shown the pruned tree with only 5 terminal nodes. This tree uses for its split just the following variables: `CRBI`, `CAtBat`, `Runs` and `RBI`. Now we want to find the regression tree that makes the best prediction. To compare the different trees we use the MSE. Hence, now we will compute the MSE for each regression tree.

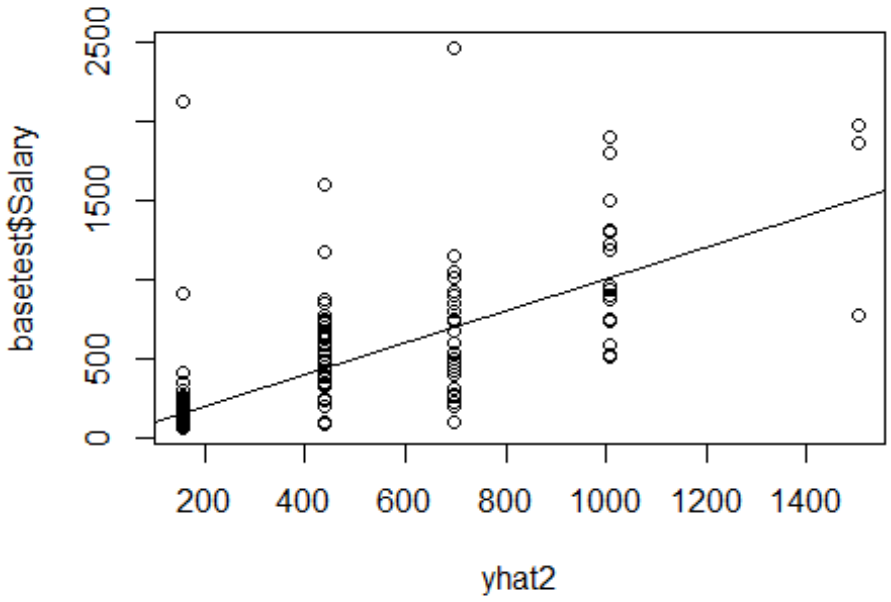
```r
yhat <- predict(tree.baseball, newdata = basetest)
plot(yhat, basetest$Salary)
abline(0,1)
```




```
MSE_Simple_tree <- mean((yhat- basetest$Salary)^2)
```

The test set MSE associated with the regression tree is 160051.3 Hence, the square root of the MSE is therefore around 400.0641 indicating that this model leads to test predictions that are within around $400.0641 of the true median salary for a baseball player of the major league.

```
yhat2 <- predict(prune.baseball, newdata = basetest)
plot(yhat2, basetest$Salary)
abline(0,1)
```




```
MSE_Pruned_tree <- mean((yhat2 - basetest$Salary)^2)
```

The MSE obtained from the pruned tree is equal to 137877.7 and it is lower than the one obtained from the simple regression tree. From this we can conclude that:

1) Pruning the tree has improved the performance and this helps to make more accurate and more interpretable results;

2) this second model leads to test prediction that are within around $371.3189 of the true median salary for a baseball player of the major league.

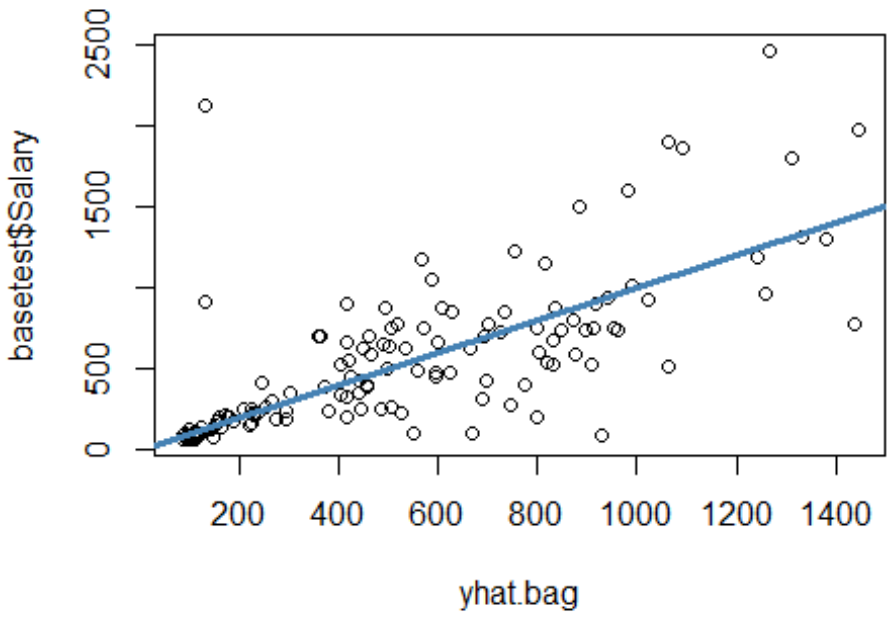
Apply bagging on the training portion of the data and evaluate the test MSE. Does bagging improve the performance?

Now we want to improve the performance of our model using the bagging which is a general purpose procedure for reducing the variance of a statistical learning methods.

```
set.seed(99)
bag.baseball <-  randomForest(Salary ~., data = basetrain, mtry=19, importance = T
) # mtry = 19 indicates that all predictors should be considered for each split of
the tree or, in other words, that bagging should be done.
bag.baseball

##
## Call:
```

```
##  randomForest(formula = Salary ~ ., data = basetrain, mtry = 19,      importanc
e = T)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##           Mean of squared residuals: 77176.72
##                     % Var explained: 57.86
```

Now that we have created this bagged model we want to see how well it performs on the test



data.

```
yhat.bag <-  predict(bag.baseball, newdata = basetest)
plot(yhat.bag, basetest$Salary)
abline(0,1, col = 'steelblue', lwd = 3)
```

```
MSE_bagged <- mean((yhat.bag-basetest$Salary)^2)
```

The test set MSE associated with the bagged regression tree is 112033.3, that is lower than the one obtained with the pruned tree. This means that our prediction of the salary will be corrected within a range of 334.71$. Hence, as we expected, applying the bagging improves the performance. This improvement is also visible from the plot where the observations are better spread.

When we grow a random forest, we have to choose the number m of variables to consider at each split. Remember that bagging is a particular case of random forest with m equal to the number of explanatory variables nvar. Set the range for m from 1 to nvar. Define a matrix with nvar rows and 2 columns and fill it with the test error (1st column) and OOB error on training data (2nd column) corresponding to each choice of m. Save the matrix as a dataframe and give it suitable column names. Compare OOB errors with test errors across the m values. Are the values different? Do they reach the minimum for the same value of m?

Now we are asked to check if the MSE is different from the OOB which is an alternative way to estimate the test error of a bagged model. To check these two values for all the possible values of `mtry` we use a for cycle. To store the data we create two empty vectors and we will append the corresponding values at each iteration.

```
list_of_MSE <-  c()
list_of_OOB <-  c()
list_of_mtry <-  c()
results <-  matrix(ncol = 3, nrow = 19)
giri <- ncol(baseball) - 1
set.seed(99)
for ( i in 1: giri){
  bag.baseball <-  randomForest(Salary ~., data = basetrain, mtry=i, importance =
T)
  yhat.bag <-  predict(bag.baseball, newdata = basetest)
  MSE.bag <- mean((yhat.bag-basetest$Salary)^2)
  list_of_MSE <- append(list_of_MSE, MSE.bag)
  OOB <- bag.baseball$mse[length(bag.baseball$mse)]
  list_of_OOB <- append(list_of_OOB,OOB )
  list_of_mtry <- append(list_of_mtry,i)
}
```

Now we aggreagte the results in the above constructed matrix.

```
results[,1] <-  list_of_MSE
results[,2] <-  list_of_OOB
results[,3] <-  list_of_mtry
results <-  data.frame(results)
colnames(results) <- c('MSE','OOB', 'mtry')
knitr::kable(results)
```

MSE	OOB	mtry
120828.7	77345.20	1
116112.6	72096.52	2
114641.7	74122.16	3
114075.0	73071.84	4
112767.8	73696.65	5
112641.0	74720.04	6

111772.3	76891.32	7
113394.2	75194.35	8
114816.0	78650.83	9
113430.4	76508.70	10
113442.1	76367.89	11
112667.8	75676.69	12
113381.2	78777.76	13
112811.5	80310.04	14
112803.4	76997.81	15
111683.0	78704.81	16
112102.5	78362.41	17
111745.6	76398.66	18
113283.3	77557.62	19

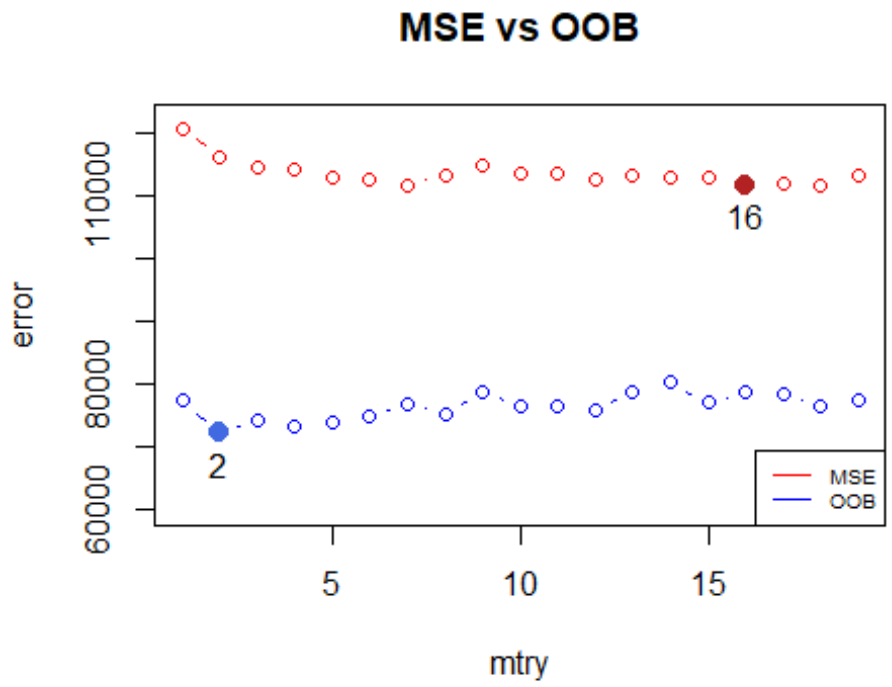
In the above table we show the values of the MSE error and the OOB error for each value of `mtry`. Now, we also plot the results in order to better visualize and compare the results

```
plot(seq(1,19), list_of_MSE, type='b', col = 'red', main = 'MSE vs OOB', xlab = 'm
try', ylab = 'error', ylim = c(60000,122000))
lines(seq(1,19),list_of_OOB, type='b', col = 'blue')
legend('bottomright', legend = c('MSE', 'OOB'), col=c('red','blue'), lty = 1, cex
= .64)
points(which.min(list_of_MSE), list_of_MSE[which.min(list_of_MSE)], col = 'firebri
ck', cex = 2, pch = 20)
text(which.min(list_of_MSE), list_of_MSE[which.min(list_of_MSE)], labels = which.m
in(list_of_MSE), pos = 1)
points(which.min(list_of_OOB), list_of_OOB[which.min(list_of_OOB)], col = 'royalbl
ue', cex = 2, pch = 20)
```

```
text(which.min(list_of_OOB), list_of_OOB[which.min(list_of_OOB)], labels = which.m
in(list_of_OOB), pos = 1)
```

Here we have plotted the values of the MSE and the OOB for each value of `mtry` and highlighted for which number of of `mtry` the two errors reach their minimum values. Now we have show again the best number of `mtry` for the OOB error and the MSE.

```
min_OBB <-  which.min(results$OOB)
min_MSE <-  which.min(results$MSE)
```




```
sprintf('The optimal number of mtry for the MSE is %d',min_MSE)

## [1] "The optimal number of mtry for the MSE is 16"

sprintf('The optimal number of mtry for the OBB is %x',min_OBB)

## [1] "The optimal number of mtry for the OBB is 2"
```

Firstly, it is easy to notice that the values for the MSE differ from the ones of the OBB. Moreover, we have also checked that they reached their minimum for different values of variables considered in the model.

Reach a conclusion about the optimal random forest model on the training data and evaluate the model performance on the test data. Identify the variables that are important for prediction.

Now we have to choose an optimal random forest model on the training data. Hence, we are asked to make a priori decision based on our training data without considering the validation set. Thus, the most logic thing to do is to choose our model based on the value of the OOB which is also a valid estimate of the test error since the response for each observation is predicted using only the trees that were not fit using that observation. So, we decide to choose the model with the minimal OBB which is the one that considers ($m = 2$) just 2 predictors for each split of the tree. This is good for two reasons:

1) This allows to avoid problems connected with strong predictors. This should lead to a substantial reduction of the variance;

2) Using a small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors as in this case.

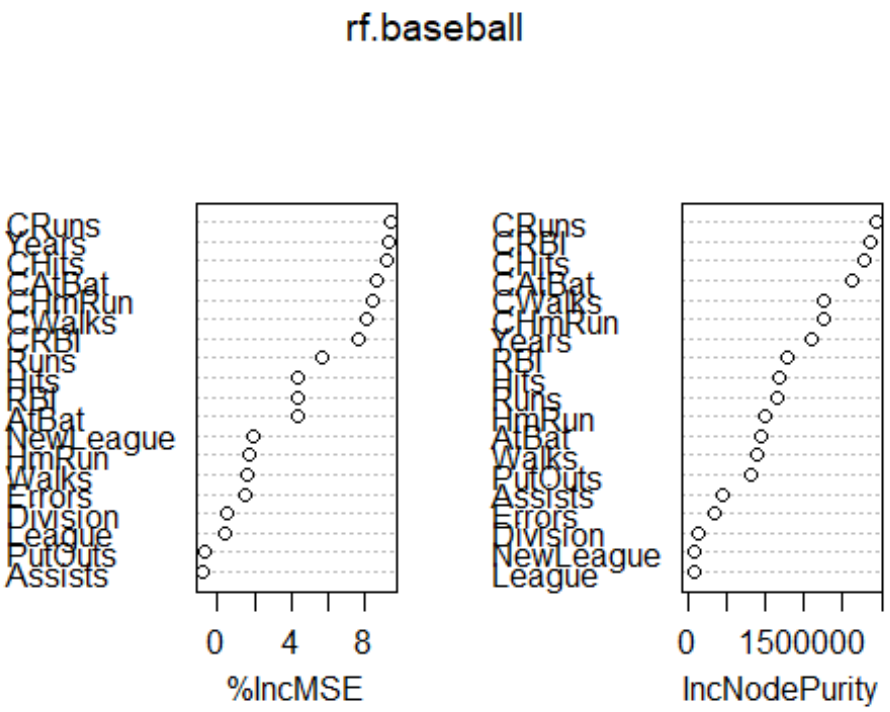
So, we decide to compute a random forest model based on the results obtained from the analysis of the OOB error.

```
set.seed(99)
rf.baseball <-  randomForest(Salary ~., data = basetrain, mtry=2, importance = T)
yhat.rf <-  predict(rf.baseball, newdata = basetest)
MSE.rf <- mean((yhat.rf-basetest$Salary)^2)


knitr::kable(importance(rf.baseball))
```

	%IncMSE	IncNodePurity
AtBat	4.3090218	943632.37
Hits	4.3892355	1170240.95
HmRun	1.6921886	980427.97
Runs	5.6912423	1149986.21
RBI	4.3522249	1290104.57
Walks	1.6263727	875513.17
Years	9.2525050	1589859.91
CAtBat	8.5840894	2110067.55
CHits	9.1834240	2268255.72
CHmRun	8.4066051	1742022.29
CRuns	9.3432940	2423485.83
CRBI	7.6582133	2349808.55
CWalks	8.0737597	1758369.82

League	0.3710813	77094.94
Division	0.4857131	121121.35
PutOuts	-0.6572321	802898.58
Assists	-0.7514112	431420.90
Errors	1.4669922	333373.78
NewLeague	1.9130612	80208.22

```
varImpPlot(rf.baseball)
```



Now we want to inspect the results obtained from the just compute random forest model. As we can see from the above plot and the previous table there are many important predictors. `CRuns` is probably the most important one, the other relevant predictors are the following `CRBI` and `CHits`. However, we wonder if the performance of the random forest can be improved. Actually, we are quite suspicious about our results because we expected an improvement from the bagging to the random forest but it did not occur; the MSE of the random forest is higher than the one obtained

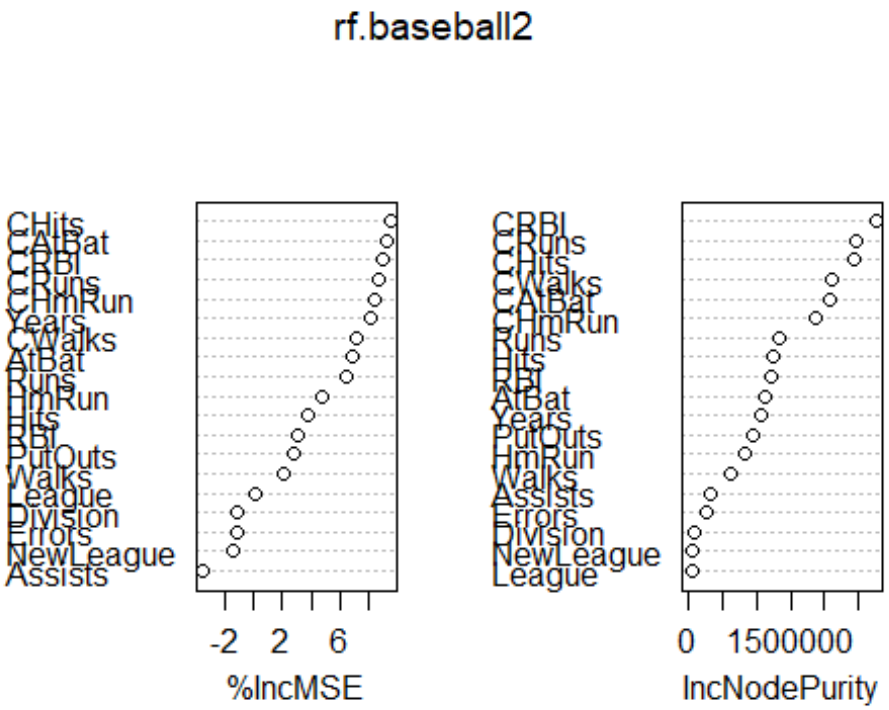
with the bagging. We have made our decision on the basis of our training data but we know that considering $m \approx \sqrt{p}$ typically leads to better results. Thus, we want to verify it.

```
set.seed(99)
m <-  round(sqrt(ncol(baseball)-1),1)
rf.baseball2 <-  randomForest(Salary ~., data = basetrain, mtry= m, importance = T
)
yhat.rf2 <-  predict(rf.baseball2, newdata = basetest)
MSE.rf2 <- mean((yhat.rf2-basetest$Salary)^2)
print(MSE.rf2)

## [1] 112913.5

knitr::kable(importance(rf.baseball2))
```

	%IncMSE	IncNodePurity
AtBat	6.8389001	1127056.54
Hits	3.7428797	1254359.43
HmRun	4.7291682	835021.72
Runs	6.4780626	1318048.50
RBI	3.0294201	1209111.78
Walks	2.0427366	605911.33
Years	8.0436245	1072939.23
CAtBat	9.2363618	2066433.77
CHits	9.4290757	2424913.42
CHmRun	8.3627685	1861750.91
CRuns	8.6904025	2464509.47
CRBI	8.8984851	2750663.65
CWalks	7.0851763	2107975.27
League	0.1878733	57952.73
Division	-1.0713303	65655.10
PutOuts	2.7420086	943819.96
Assists	-3.4378088	306924.16
Errors	-1.1720844	246372.06
NewLeague	-1.4277839	62665.53

We have verified that using a value of $m \approx \sqrt{p}$ has improved the performance of the random forest although the MSE is still slightly higher than the one obtained with the bagging. Lastly, we now want to make a focus on the importance of the variables of the new random forest.

```
varImpPlot(rf.baseball2)
```

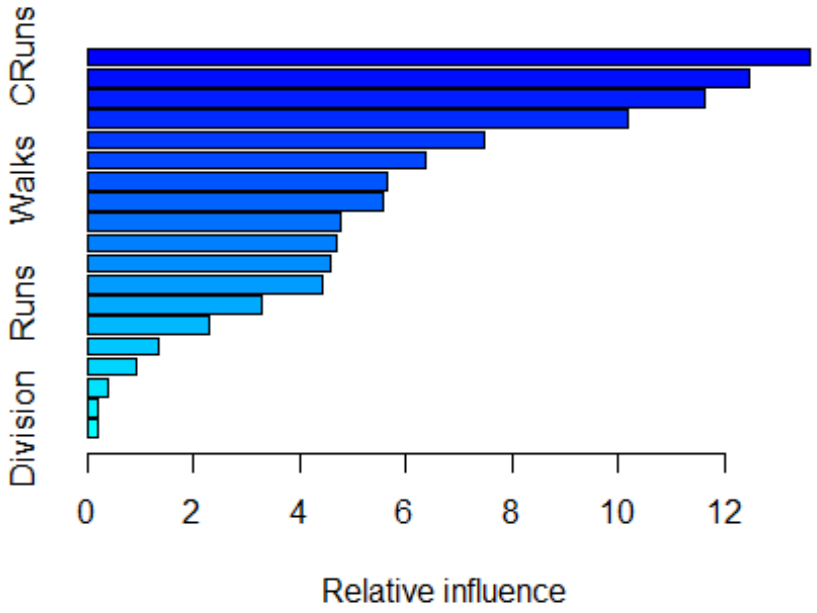


From this second importance plot we get that the most important predictors are still the same: `CRuns`,`CRBI` and `CHits` although, some little difference in the values.

Fit a regression tree on the training data using boosting. Find the optimal number of boosting iterations, both by evaluating the OOB error and the cross-validation error. Produce plots with OOB error and CV error against the number of iterations: are the two methods leading to the same choice of the optimal number of iterations? Reach a conclusion about the optimal model, evaluate the test MSE of this model and produce a partial dependence plot of the resulting top N variables (N of your choice).

We now discuss boosting, another approach for improving the predictions resulting from a decision tree. Boosting works similarly to bagging but with this new method, trees are grown *sequentially*, each tree is grown using information from previously grown trees and each tree is fit on a modified version of the original data set. Boosting approach learns slowly and typically these kind of methods are the ones that perform better. So, we expected an improvement in our results after having applied boosting.

```
set.seed(99)
boost.baseball <-  gbm(Salary~., data = basetrain, distribution = 'gaussian', n.tr
ees = 5000, interaction.depth = 4)
knitr::kable(summary(boost.baseball))
```



	var	rel.inf
CRuns	CRuns	13.6242310
CWalks	CWalks	12.4667275
CRBI	CRBI	11.6202464
CHmRun	CHmRun	10.1976898
CAtBat	CAtBat	7.4630522
CHits	CHits	6.3758525
Walks	Walks	5.6614232
PutOuts	PutOuts	5.5739918
Years	Years	4.7751211
RBI	RBI	4.6802231
AtBat	AtBat	4.5787440
Hits	Hits	4.4122468
Runs	Runs	3.2686934
HmRun	HmRun	2.2790007
Errors	Errors	1.3274653
Assists	Assists	0.9156433
NewLeague	NewLeague	0.3834513
League	League	0.2129158

Division	Division	0.1832809

```
yhat._first_boost <-  predict(boost.baseball, newdata = basetest, n.trees = 5000)
MSE_first_Boost <- mean((yhat._first_boost - basetest$Salary)^2)
```

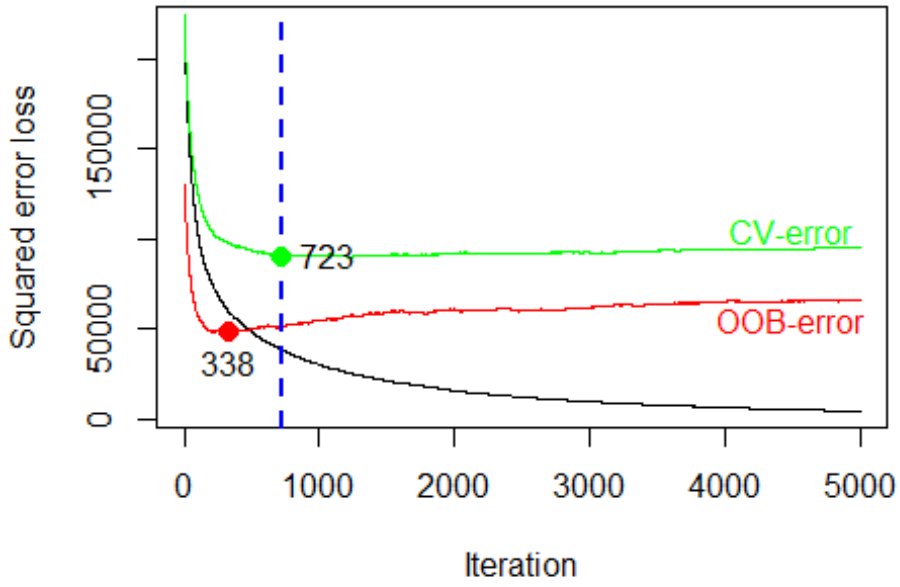
Here we have computed the boosting using 5000 trees and the MSE associated with this new model. At a first look boosting does not lead to a significant improvement of the performances. However, we now that we can tune three parameters in order to improve boosting. We have also produced the relative influence plot and also the relative influence statistics. These two gives us very useful information about the most important predictors although, this is not the definitive model. The first thing we notice is that the most important predictors are the ones that show the whole career statistics. The second thing is that these predictors also show statistics about the performances of the players. Hence, we may seriously start thinking that the salary of a player depends on the overall performances in his career.

Now we are asked to find the optimal number of boosting iterations by evaluating both the OOB error and the cross-validation error.

```
set.seed(99)

boost_hit <- gbm(Salary~., data=basetrain, distribution = 'gaussian',
          interaction.depth=4,
          bag.fraction = 0.5, # info che passo da un albero all'altro
          train.fraction = 0.6, # Uso il 60% del train il resto per la validazione
          cv.folds=10,
          n.trees = 5000,
          shrinkage = 0.01, verbose = F)
```

Now we decide to plot the results

```
cv_boost <- gbm.perf(boost_hit, method = 'cv')
#oob_boost <- gbm.perf(boost_hit, method = 'OOB')
#boost_hit$valid.error # OOB
#boost_hit$cv.error # cv
n_iteration_cv <- which.min(boost_hit$cv.error)
n_iteration_oob <-  which.min(boost_hit$valid.error)
points(n_iteration_oob, min(boost_hit$valid.error), col = 'red', cex = 2, pch = 20
)
points(n_iteration_cv, min(boost_hit$cv.error), col = 'green', cex = 2, pch = 20)
text(n_iteration_oob, min(boost_hit$valid.error), labels = n_iteration_oob, pos =
1)
text(n_iteration_cv, min(boost_hit$cv.error), labels = n_iteration_cv, pos = 4)
#legend(3700, 200000,legend = c('OOB-error', 'CV-error'), col= c('red','green'), b
ox.lty=0)
text(4500, 105000, labels = 'CV-error', col = 'green')
text(4500, 55000, labels = 'OOB-error', col = 'red')
```



In the above plot we show how the cross-validation error and the OOB error change according to the number of iterations. Here the main point is to choose the optimal number of iteration for boosting. The OOB error and the CV error show different results. For the OOB the best number of iteration is 338 while for CV-error is 723. However, we tend to prefer the output provided by the CV-error. Indeed, if run `oob_boost <- gbm.perf(boost_hit, method = 'OOB')` R gives us a warning *OOB generally underestimates the optimal number of iterations although predictive performance is reasonably competitive.* This underestimation may be due to the fact that the GBM method is partly estimated on in-bag samples, as the OOB samples for the Nth iteration are likely to have been in-bag in previous iterations. This approach is known to underestimate the number of required iterations, which means that it's not very useful in practice. Hence, the main finding is that CV remains the most reliable approach. In conclusion, stick to cross-validation for the best results which is 723 iterations. Now that we have chosen a criteria to define the optimal number of iteration for the boosting we recompute the bagging and we estimate the MSE

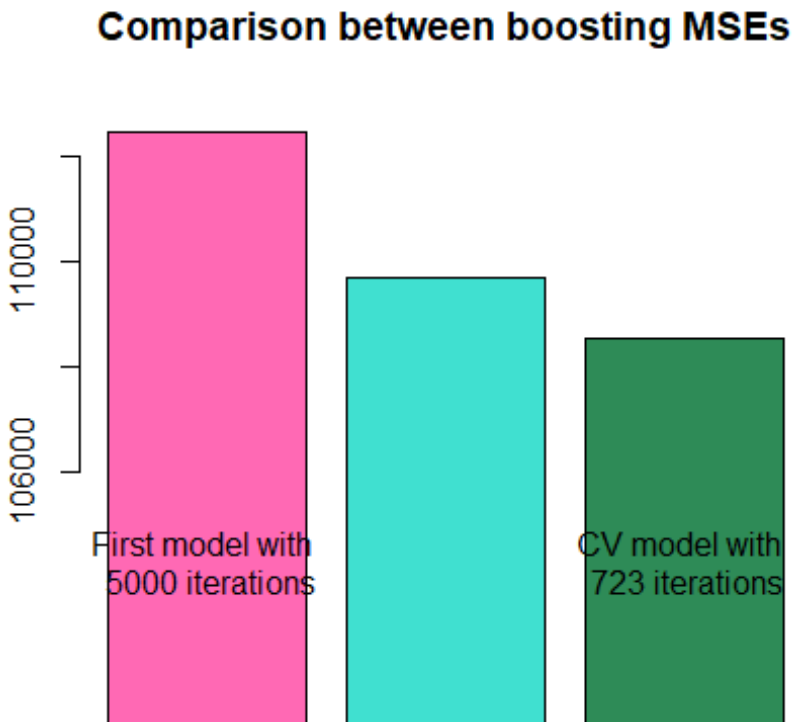
```
set.seed(1)
best.boost  <-  gbm(Salary~., data = basetrain, distribution = 'gaussian', n.trees
= n_iteration_cv, interaction.depth = 4)
oob.boost <- gbm(Salary~., data = basetrain, distribution = 'gaussian', n.trees =
n_iteration_oob, interaction.depth = 4)

yhat.boost <-  predict(best.boost, newdata = basetest, n.trees = n_iteration_cv)
MSE_BBoost <- mean((yhat.boost - basetest$Salary)^2)
yhat.boost_oob <- predict(best.boost, newdata = basetest, n.trees = n_iteration_oo
b)
MSE_OOBboost <- mean((yhat.boost_oob - basetest$Salary)^2)
```

Here we have computed the MSE of our relatively 'optimal' boosting model with the number of interactions choose by the CV-error and also the MSE of the model with the optimal number of iterations defined by the OOB error. (The purpose of this was for double checking that the choice based on the CV-error). We see that the MSE of the model computed with a number of iteration defined by the CV is slightly lower than the one computed with the iteration term defined according to the OOB-error. However, both the new models have shown a lower MSE compared to the first boosting model computed with a default number of iterations (5000).

```
B_results <-  c(MSE_first_Boost, MSE_OOBboost, MSE_BBoost)
etichette <-  c('First model with \n 5000 iterations', 'OOB model with\n 338 itera
tions', 'CV model with \n 723 iterations')
boosting_results <-  data.frame(B_results, etichette)
knitr::kable(boosting_results)
```

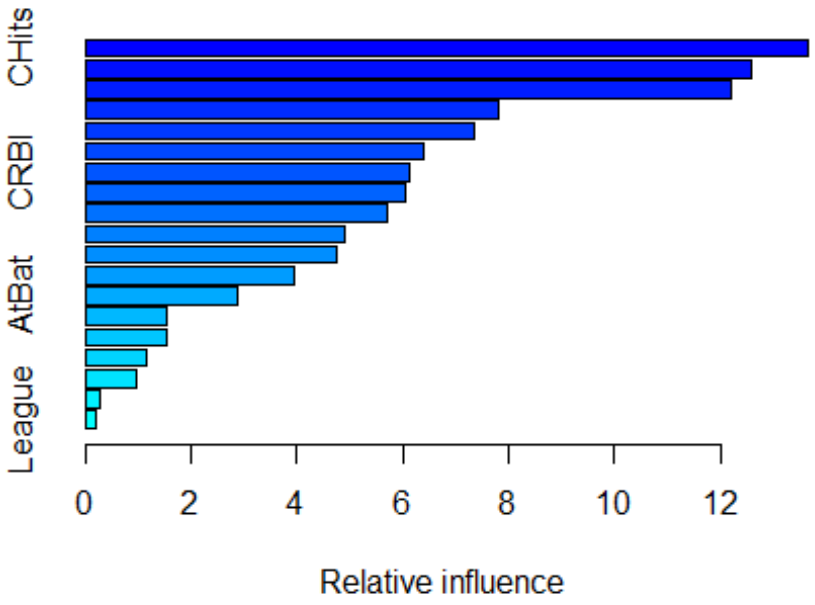
B_results	etichette
112485.9	First model with 5000 iterations
109687.7	OOB model with 338 iterations
108517.5	CV model with 723 iterations

```
barplot(B_results, names.arg = etichette, col = c('hotpink', 'turquoise', 'seagree
n'), ylim = c(105000,113000), main = 'Comparison between boosting MSEs', width = 5
)
```



Here we are representing the results of the MSEs of the boosting with different number of iterations. We have decided to report a table and a graph because it is important to take a look to the numerical values before watching the bar plot. We say so because we are zooming in to the bar plot in order to make visible the difference among the MSEs. However, this is not a common practice in dataviz because this kind of representation could be misleading for the reader. However, statistically speaking, we have checked that the MSE computed with the number of iteration defined by the CV-error is better than the other one.

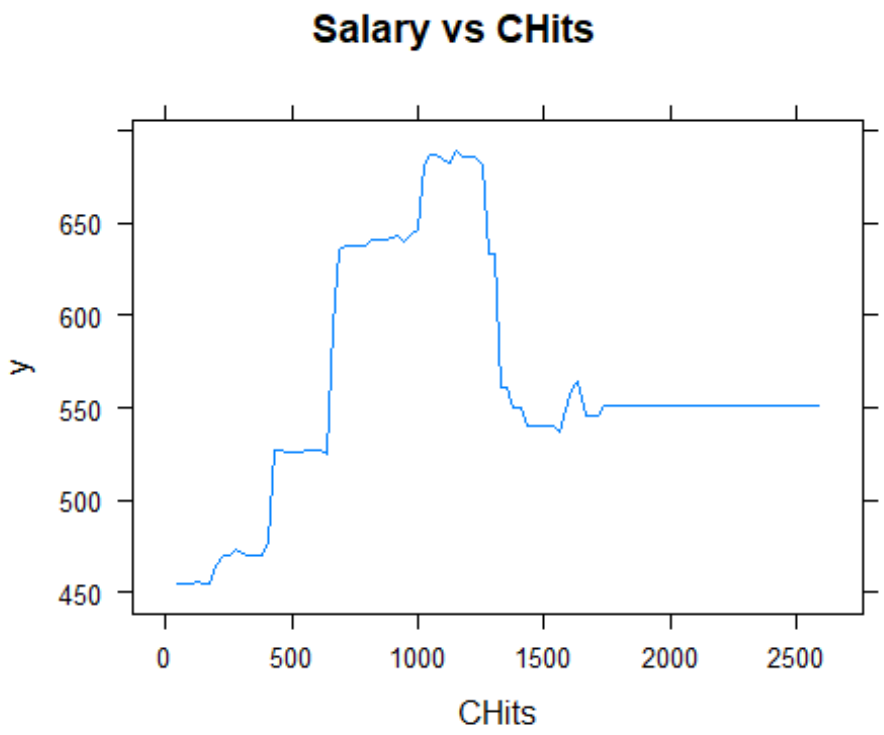
```
knitr::kable(summary(best.boost))
```



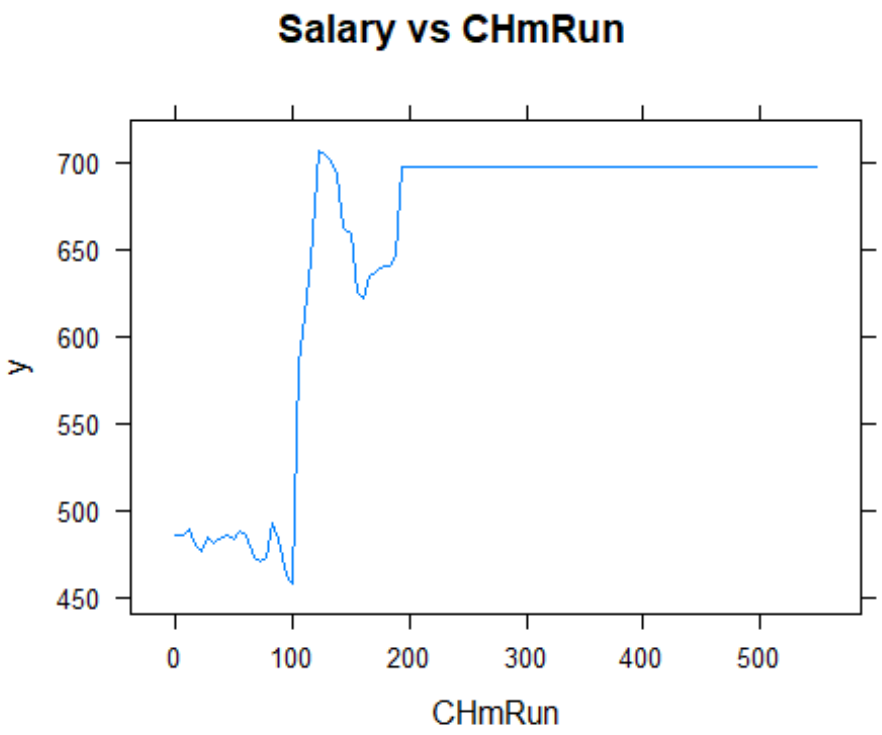
	var	rel.inf
CHits	CHits	13.6654242
CHmRun	CHmRun	12.6065833
CWalks	CWalks	12.1885860
CRuns	CRuns	7.8206337
Hits	Hits	7.3480709
CAtBat	CAtBat	6.3813488
CRBI	CRBI	6.1362801
PutOuts	PutOuts	6.0406042
Walks	Walks	5.7210478

Runs	Runs	4.9139788
Years	Years	4.7342110
RBI	RBI	3.9515418
AtBat	AtBat	2.8758034
HmRun	HmRun	1.5415683
Errors	Errors	1.5198583
Assists	Assists	1.1378682
Division	Division	0.9645647
NewLeague	NewLeague	0.2631260
League	League	0.1889007

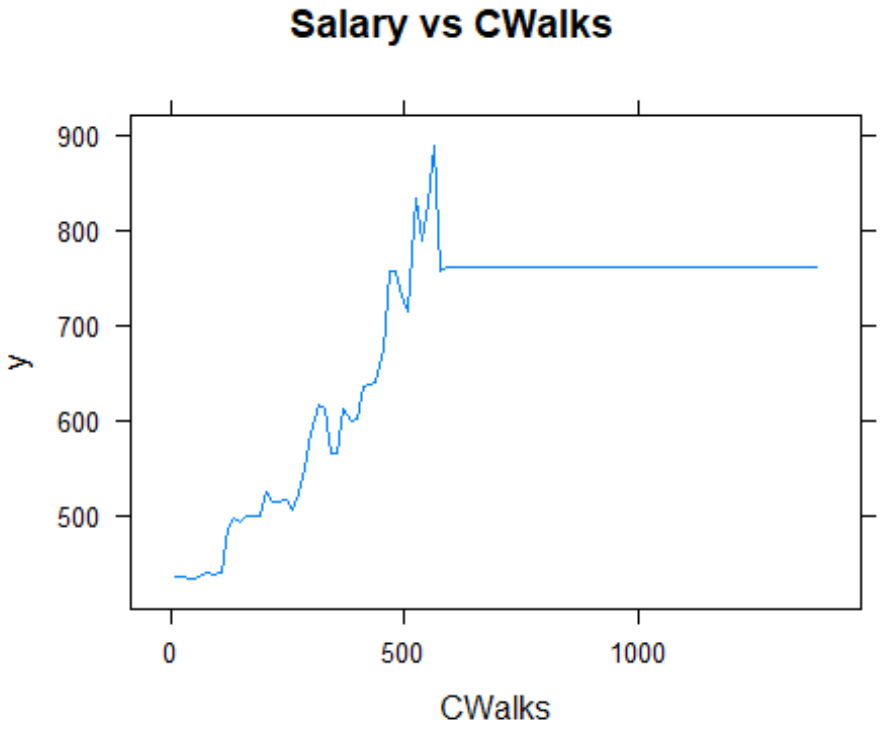
```
plot(best.boost, i = 'CHits', main='Salary vs CHits')
```




```
plot(best.boost, i = 'CHmRun', main='Salary vs CHmRun')
```




```
plot(best.boost, i = 'CWalks',  main='Salary vs CWalks')
```



Now to conclude this task we are going to plot the dependence plot of the 3 most important predictors and to comment the results.

According to `best.boost` the three most important variables are `Chits`, `CHmRun`, `CWalks`. The first thing that we notice is that all these variables refer to statistics that consider the whole career of a player, this can help us to formulate some hypothesis: for example, we may think that the `Salary` of player depends more on his whole career rather than a single season. Now we look more in details the individual dependence plot:

1) *Salary vs CHits* we see that the `Salary` increase when the number of total hits in career grows until it is reached the peak. After that it start to decrease and then it becomes flat and fixed to a certain level. This results can be interpreted as follows: the salary of player grows with the number of hits which is also probably a good estimate of games played. So, a player earns more when he becomes more experienced but until a certain level, because when players become too old their performance probably decrease and with it the money that a team is eager to pay.

2) *Salary Vs CHmRun* we see an almost stable trend for the salary in relation with the number of home runs in career. However, reached 100 home runs the `Salary` has an enormous growth then it slightly decreases and then it grows again reaching almost the peak value to remain stable for a high number of total home runs. To better interpret the results we may have to consider that the home run is the highest point that a player can do when he is trying to hit the ball. Hence, we may think that a player with more than 100 home runs is a good player and deserves an high salary. We may probably assume that a player reaches this number of home runs after some years played in the major league and after many hits. So this is coherent with the previous commented plot. Lastly, we notice that for high numbers of home runs (starting from 200) there is a fixed and high level of income for the players. This can be explained by saying that players who arrive to these numbers of home run are exceptional players.

3) *Salary vs CWalks* As for the other variables we see that the Salary grows with the number of walks in career until then after a value of 500 walks the `Salary` decrease a bit and then it becomes fixed for higher values of walks. The walks in baseballs refers to a point that can be done by a pitcher. The trend of walks and salary can be explained as follows: the number of walks in career increase with the number of games played and so as for the number of home runs a player is awarded for his good performances with an increase in the salary until it becomes too old and the salary slightly decrease and then it becomes high and constant for player that have achieved very good results.

Draw some general conclusions about the analysis and the different methods that you considered

To conclude our analysis, we want to firstly compare the model by visualizing the different MSEs provided by the model and then on the basis of the optimal models try to explain the values that we have computed.

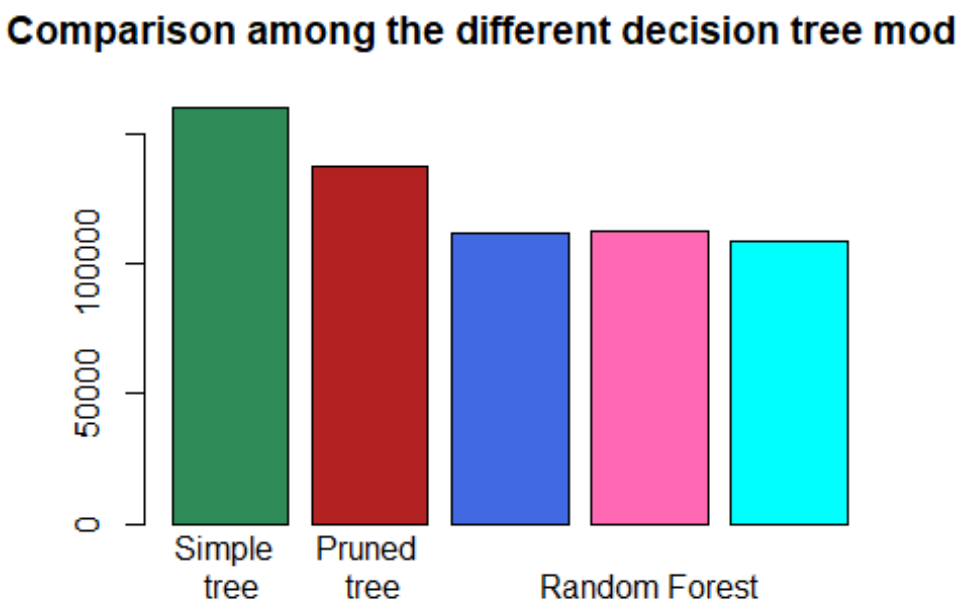
```
MSEs_decision_trees <-  c(MSE_Simple_tree, MSE_Pruned_tree, MSE_bagged, MSE.rf2,MS
E_BBoost)
decision_tree <-  c('Simple tree', 'Pruned tree', 'Bagging', 'Random Forest', 'Boo
sting')
final_results <-  data.frame(MSEs_decision_trees, decision_tree)
```

```r
colnames(final_results) <-  c('MSE','Model')
knitr::kable(final_results)
```

MSE	Model
160051.3	Simple tree
137877.7	Pruned tree
112033.3	Bagging
112913.5	Random Forest
108517.5	Boosting

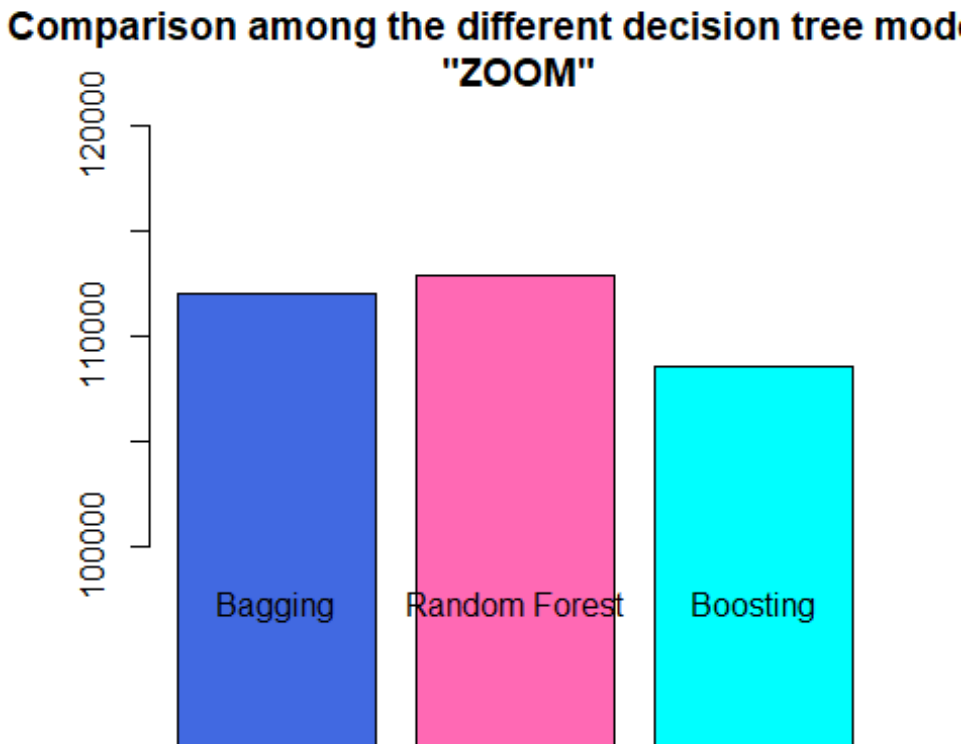
Here we have created a small data frame wehre we summarize the results obtained. Now we decide to mske a barplot to have a visual insight into the results.

```r
barplot(MSEs_decision_trees, names.arg =  c('Simple \n tree', 'Pruned \n tree', 'Bagging', 'Random Forest', 'Boosting'),main='Comparison among the different decision tree models', col = c('seagreen', 'firebrick', 'royalblue', 'hotpink', 'cyan1'))
```



Now we zoom into the three best models.

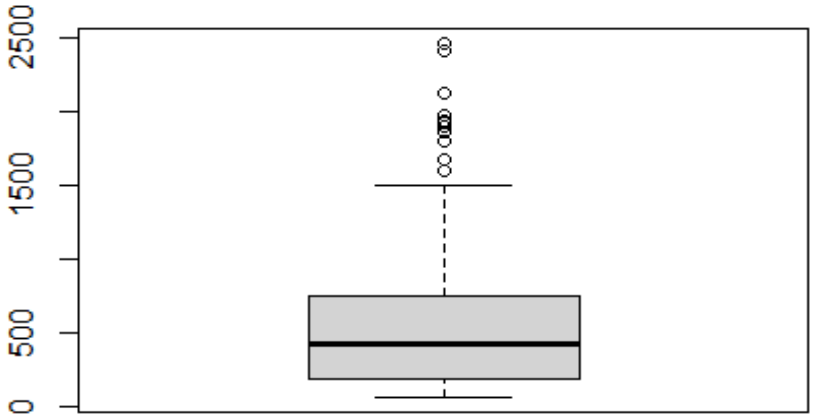
```r
barplot(MSEs_decision_trees[3:5], names.arg =  c('Bagging', 'Random Forest', 'Boosting'),main='Comparison among the different decision tree models \n "ZOOM"', col = c('royalblue', 'hotpink', 'cyan1'), ylim= c(100000, 120000) )
```



Here we have shown two graphs: the first is a bar plot which represent the MSE associated with each model, the second graph is just a zoom of the previous plot and it was made to make more visible the difference among the three best models (Bagging, Random Forest and Boosting). The first step of the analysis was creating a simple regression tree however, as we know this method can be easily improved in many different ways, the first improvement has been made by pruning the tree. Indeed, pruning should reduce problems connected with overfitting that are typically connected with too complex decision trees. The second attempt to improved the accuracy of our predictions involved the *bootstrap aggregation*. This procedure typically reduces variance and has no effect on the bias. Actually, bagging has dramatically reduced the MSE of the original regression tree. The next natural move has been trying to apply the random forest procedure. This method usually improves the performances of bagging because this procedure tries to solve the problems caused by highly correlated trees and the presence of strong predictors. Nonetheless, here applying the random forest procedure not only did not lead to an improvement of the performance but it has also increased the MSEs. It is difficult to explain this result also because we have some highly correlated variables that typically negatively affect the bagging but not the random forest. One possible explanation, although quite bizarre, is that this result could have been influenced by the seed. Lastly, we have tried to apply boosting which is another approach for improving the predictions resulting from a decision tree. Here we have experienced an improvement of the performances that has lead to the best tree based method. Having clear all the statistical procedure we can try to give a more concrete interpretation of our output. We had to analyze a data set containing the statistics regarding all the baseball players in the major league from 1986 to 1987. The data set is made by 20 variables, many of them are highly correlated because they refer to the same stage of the match. For example, it is clear that the number of hits is related with the number of times at bat. We also notice that we can define two

macro classes of variables: the first refers to the statistics in the current season while the second takes into account the overall performance of a player in his career. The results along all the analysis have highlighted that the most relevant variables for predicting the next season salary are the ones that refers to the whole career statistics. More specifically, what determines the salary of a player is his performance in terms of points. Indeed, for the boosting method, the most important predictors are `CHits`, `CHmRun`, `CWalks` and `CRuns`. Furthermore, we have also seen from the dependence plot that there are some particular trends: until a certain point the `Salary` and the above mentioned predictors are almost proportional (when one grows also the other grows) but after the peak there was a decrease and then another increase of the salary that becomes constant. One possible explanation, for this particular trend is the following: young players are less experienced and have 'worst' statistics and they are probably the less paid. But when they start to gain experience their performance improve and with them also their salary. It is also reasonable to think that in this phase of the career will arise the differences between the average players and the top players. Indeed, we expect that after a certain value for the cumulative statistics (the ones with C) the salary of the players decrease because they have became old, but this is not the case of the above mentioned top players who probably reaches these high values before becoming too old and for their good performances they are still more paid when they are "old". We think that this could be a quite reasonable explanation for our results also because this is what typically happens in many other sports such as football or basketball.

```
boxplot(baseball$Salary)
```



To conclude we have also decided to plot this boxplot about the salary of the players which shows that the box is quite flattened and shows a quite big upper whisker and many outliers in the upper region. This also confirms that there are few players which earn more than the average.