TMA4268-Project2

Magnus Grytten, Jokhongir Khayrullaev & Rashad Naghiyev

```
# Libraries used throughout the exercise
library("ggplot2")
library("tidyverse")
library("palmerpenguins")
library("GGally")
library("MASS")
library("caret")
library("leaps")
library("glmnet")
library("pls")
library("gam")
library("e1071")
library("tree")
library("randomForest")
library("ggfortify")
```

Problem 1

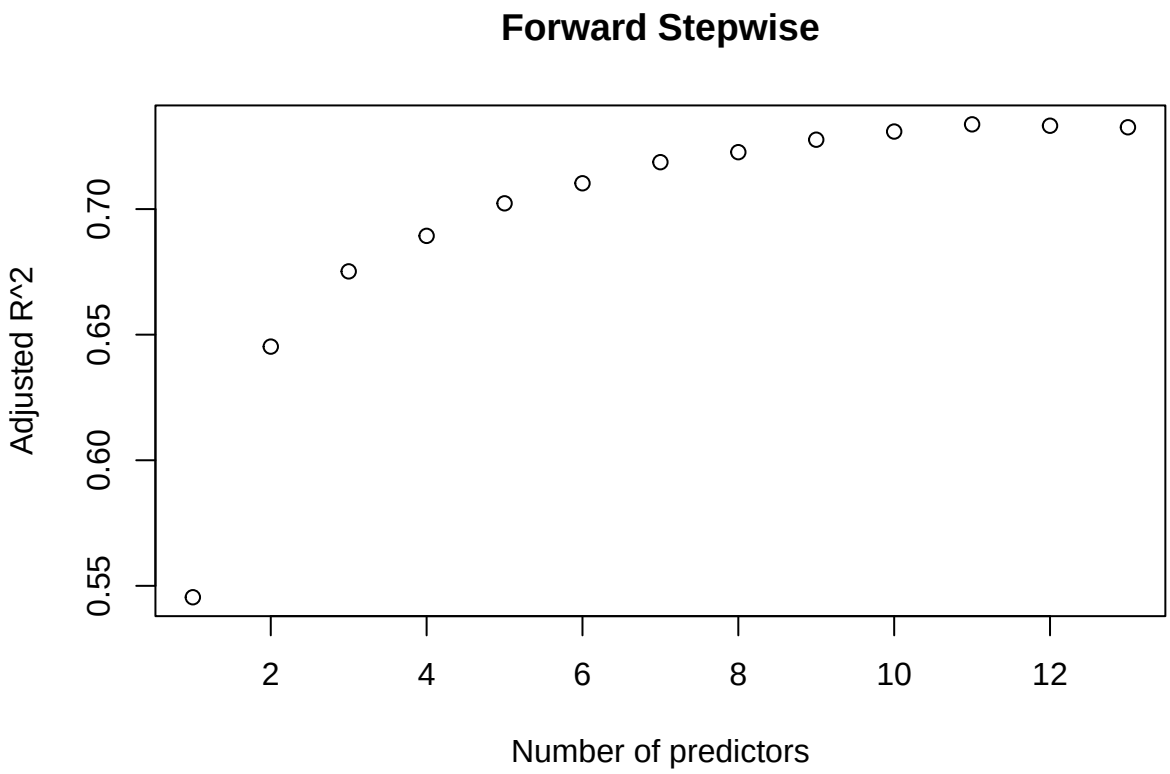
```
set.seed(1)
# pre-processing by scaling NB! Strictly speaking, pre-processing should be done
# on a training set only and it should be done on a test set with statistics of
# the pre-processing from the training set. But, we're preprocessing the entire
# dataset here for convenience.
boston <- scale(Boston, center = T, scale = T)
# split into training and test sets
train.ind = sample(1:nrow(boston), 0.8 * nrow(boston))
boston.train = data.frame(boston[train.ind, ])
```

a)

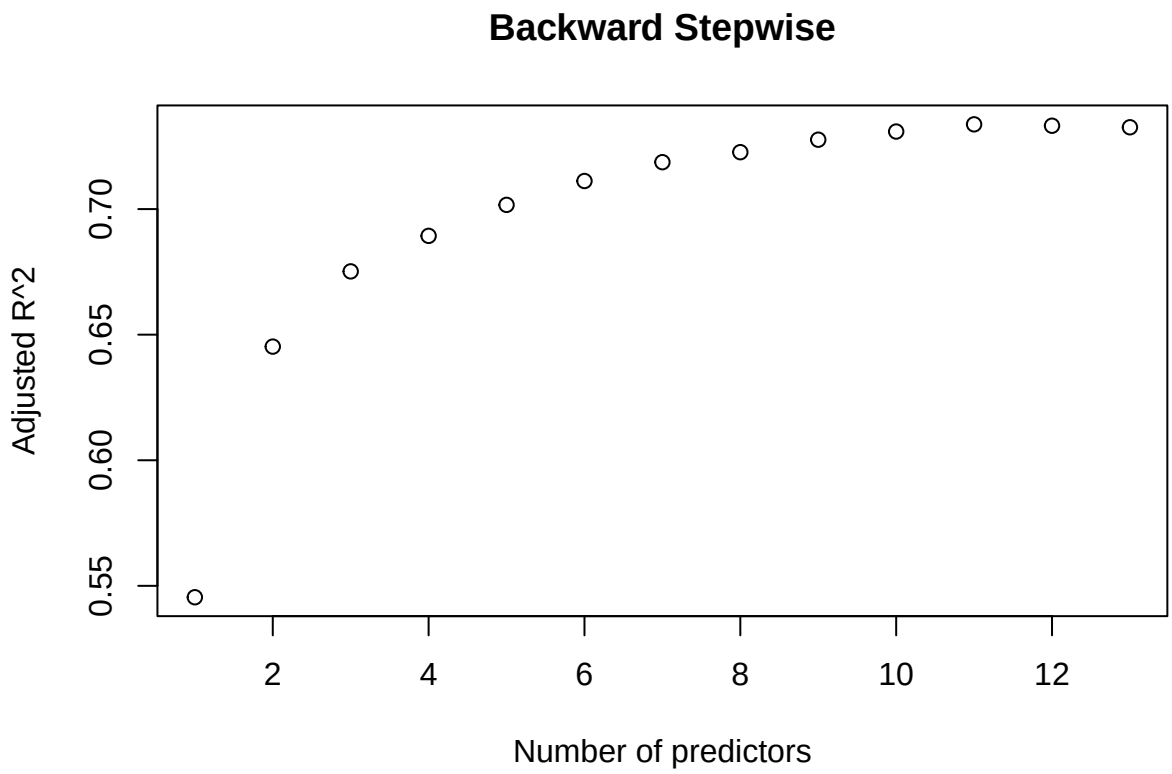
```
res_forward <- regsubsets(medv~. ,data = boston.train, nvmax = 13, method = "forward")
res_forward_sum <- summary(res_forward)

res_backward <- regsubsets(medv~. ,data = boston.train, nvmax = 13, method = "backward")
res_backward_sum <- summary(res_backward)

plot(res_forward_sum$adjr2, xlab = "Number of predictors",
     ylab = "Adjusted R^2", main = "Forward Stepwise")
```




```
plot(res_backward_sum$adjr2, xlab = "Number of predictors",
     ylab = "Adjusted R^2", main = "Backward Stepwise")
```



b)

```
coefs <- res_forward_sum$which[4,-1]
print(names(coefs[coefs == TRUE]))
```

```
## [1] "rm"      "dis"     "ptratio" "lstat"
```

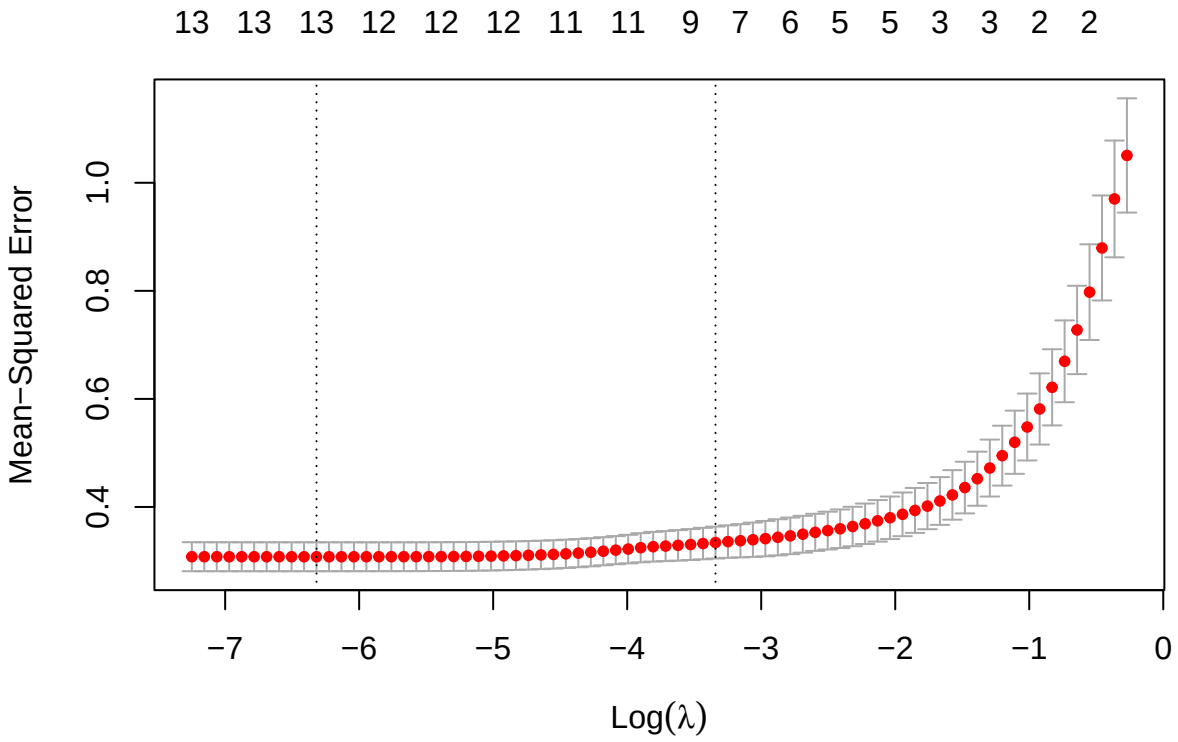
From our previous results we find that the four best predictors from forward stepwise are, "rm", "dis", "ptratio" and "lstat". These are not however necessarily the best overall four predictors we would find from best subset selection.

c)

(i)

```
x <- model.matrix(medv~.,-1,data = boston.train)
y <- boston.train$medv

cv_lasso <- cv.glmnet(x,y,nfolds = 5)
plot(cv_lasso)
```



(ii)

```
best_lambda <-cv_lasso$lambda.min
print(best_lambda)
```

```
## [1] 0.001803259
```

(iii)

```
fit_lasso <- glmnet(x,y)
coef(fit_lasso,s = best_lambda)
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.023606403
## (Intercept)  .
## crim        -0.082544050
## zn           0.095554758
## indus        0.006767917
## chas         0.087275698
## nox         -0.176974474
## rm           0.312664541
## age         -0.011672213
## dis         -0.317825160
```

```
## rad          0.274929406
## tax         -0.212678382
## ptratio     -0.204780585
## black        0.103330382
## lstat       -0.428624759
```

d)

(i) True
(ii) False
(iii) False
(iv) True

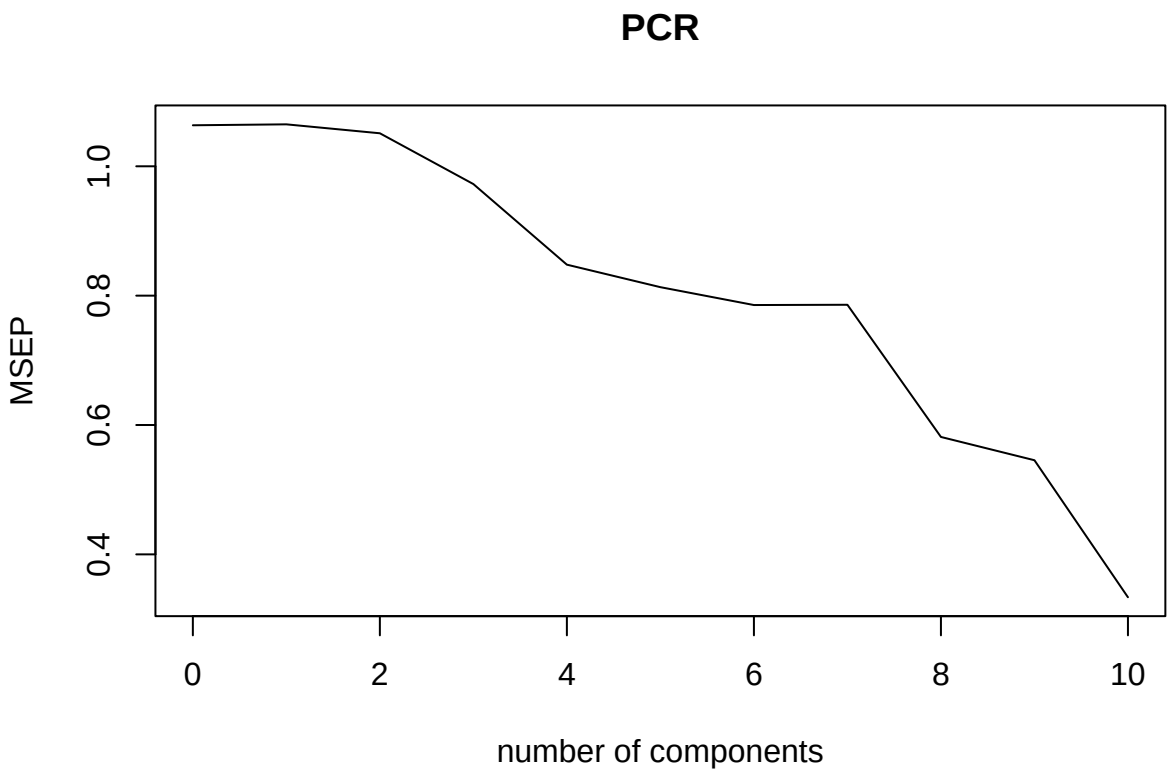
Problem 2

```
# load a synthetic dataset
id <- "1CWZYfrLOrFdrIZ6Hv73e3xxt0SFgU4Ph" # google file ID
synthetic <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
id))
# split into training and test sets
train.ind = sample(1:nrow(synthetic), 0.8 * nrow(synthetic))
synthetic.train = data.frame(synthetic[train.ind, ])
synthetic.test = data.frame(synthetic[-train.ind, ])
```

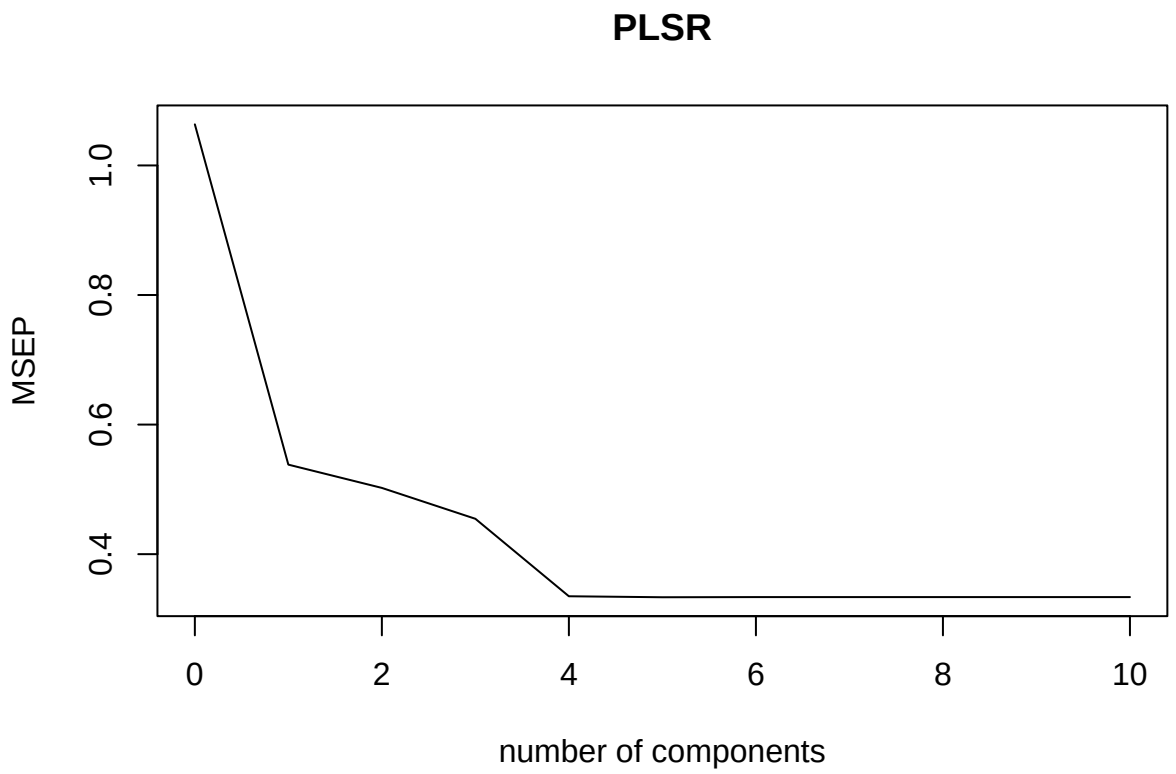
a)

```
pcr_fit = pcr(Y~., data = synthetic.train, scale = TRUE, validation = "none")
pls_fit = plsr(Y~., data = synthetic.train, scale = TRUE, validation = "none")

validationplot(pcr_fit, val.type = "MSEP", newdata = synthetic.test, main = "PCR")
```



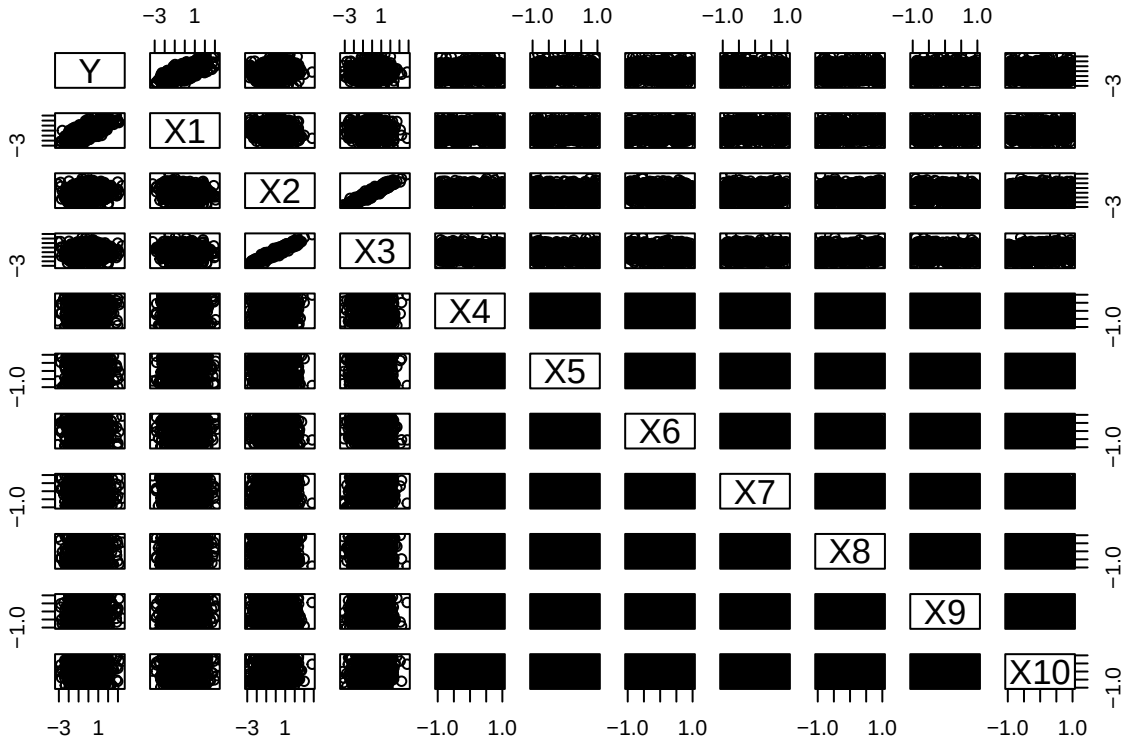

```
validationplot(pls_fit, val.type = "MSEP", newdata = synthetic.test, main = "PLSR")
```



b)

Seeing as PCR and PLSR tend to behave similarly and have similar performance it is somewhat surprising that PLSR outperforms PCR in this case. However by looking at the dataset,

```
plot(synthetic)
```



we find that X4 to X10 seems to all be uniformly distributed between -1 and 1 and uncorrelated to the other predictors. We also see that Y and X1, and X3, X4 are correlated, while the rest of the predictors are weakly- or uncorrelated. Since there are so many of the predictors that are just noise it now makes more sense that PLSR outperforms PCR since it is a supervised method, that attempts to find directions that explain the response. As opposed to PCR where in this case many of the principal components end up being weakly correlated to the repose due to many of the predictors. being noise.

Problem 3

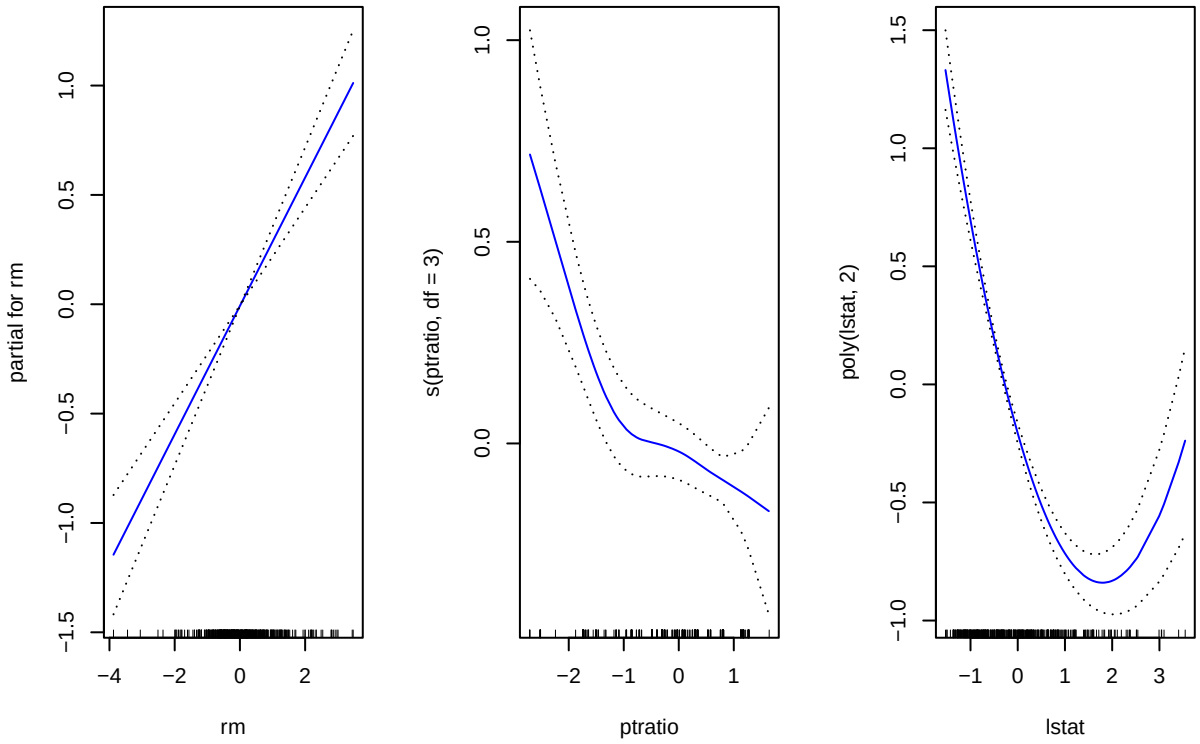
a)

i) True
ii) False
iii) False
iv) True

b)

```
fit = gam(medv ~ rm + s(ptratio, df=3) + poly(lstat, 2), data=boston.train)


par(mfrow = c(1,3))
plot(fit, se=T, col='blue')
```



Problem 4

a)

i) False
ii) True
iii) True
iv) True

b)

```
#knitr::include_graphics("tree.png")
```

c)

```
par(mfrow=c(1,1))
data(penguins)
names(penguins) <- c("species", "island", "billL", "billD", "flipperL", "mass", "sex",
                     "year")
Penguins_reduced <- penguins %>% dplyr::mutate(mass = as.numeric(mass), flipperL = as.numeric(flipperL)
                                               year = as.numeric(year)) %>% drop_na()
```

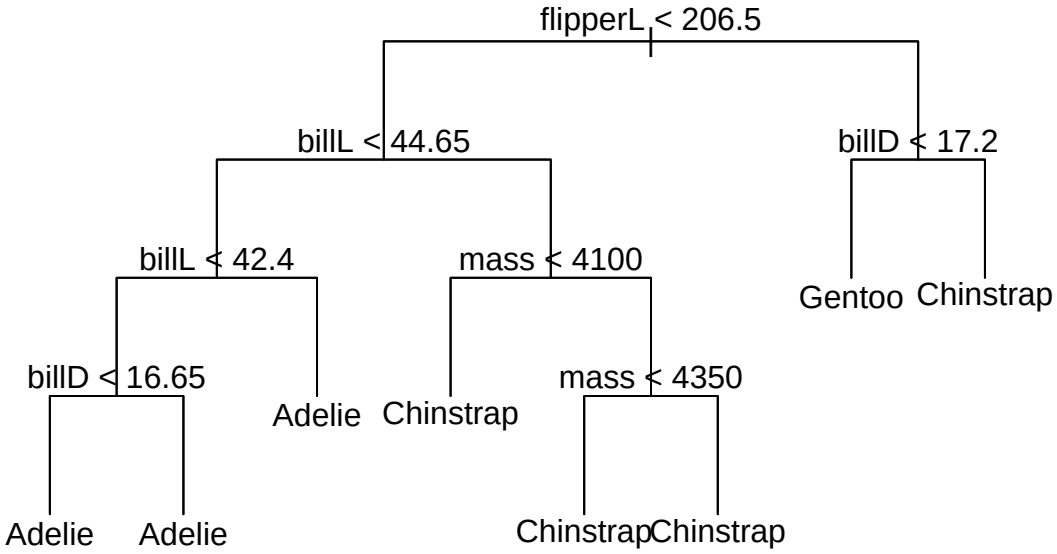
```r
# We do not want 'year' in the data (this will not help for future predictions)
Penguins_reduced <- Penguins_reduced[, -c(8)]
set.seed(4268)
# 70% of the sample size for training set
training_set_size <- floor(0.7 * nrow(Penguins_reduced))
train_ind <- sample(seq_len(nrow(Penguins_reduced)), size = training_set_size)
train <- Penguins_reduced[train_ind, ]
test <- Penguins_reduced[-train_ind, ]
```

i)

```r
species_treeG = tree(species ~ ., train, split='gini')
summary(species_treeG)
```

```
##
## Classification tree:
## tree(formula = species ~ ., data = train, split = "gini")
## Variables actually used in tree construction:
## [1] "flipperL" "billL"    "billD"    "mass"
## Number of terminal nodes:  8
## Residual mean deviance:  0.1869 = 42.06 / 225
## Misclassification error rate: 0.04292 = 10 / 233
```

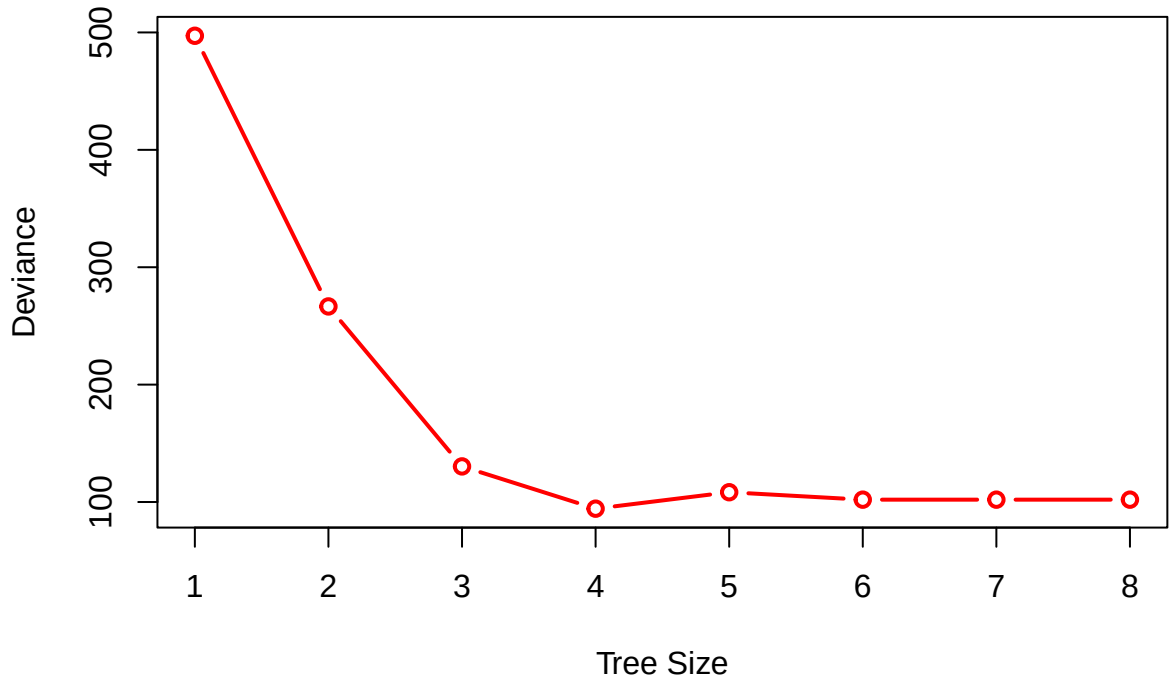
```r
plot(species_treeG, type="uniform")
text(species_treeG, pretty=0)
```



ii)

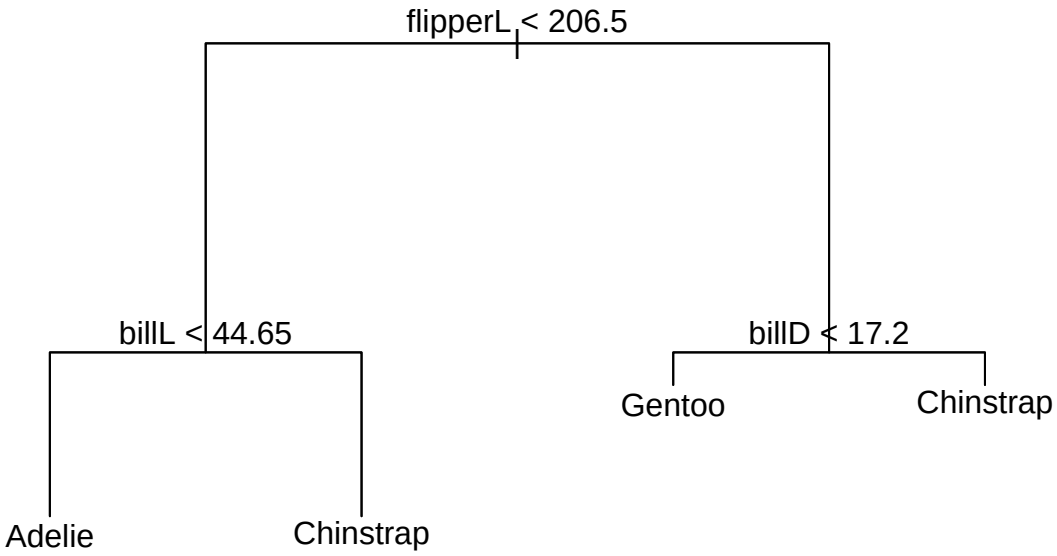
```
set.seed(123)

cv.species <- cv.tree(species_treeG, K = 10)
plot(cv.species$dev ~ cv.species$size, type = "b", lwd = 2, col = "red",
     xlab = "Tree Size", ylab = "Deviance")
```



From CV plot, we can see that the best tree size is 4. So, we use this size to prune our tree.

```
prune.species <- prune.tree(species_treeG, best = 4)
plot(prune.species)
text(prune.species, pretty = 0)
```



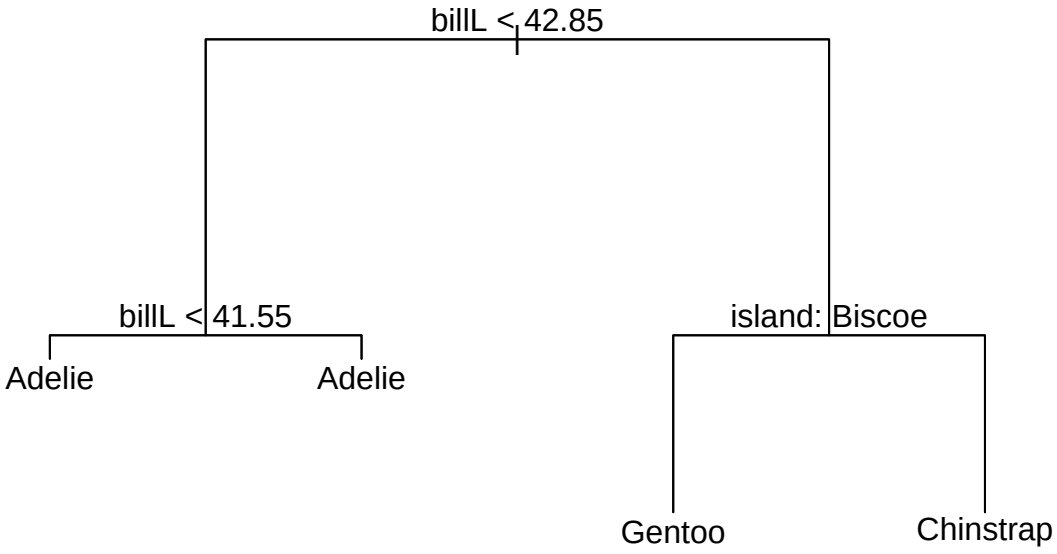
This is pruned tree using training dataset.

iii) Now we can use our test dataset.

```
species_treeG_test = prune.tree(tree(species ~ ., test,  split='gini'), best=4)
summary(species_treeG_test)
```

```
##
## Classification tree:
## tree(formula = species ~ ., data = test, split = "gini")
## Variables actually used in tree construction:
## [1] "billL"  "island"
## Number of terminal nodes:  4
## Residual mean deviance:  0.09899 = 9.503 / 96
## Misclassification error rate: 0.02 = 2 / 100
```

```
plot(species_treeG_test)
text(species_treeG_test, pretty = 0)
```



And the misclassification:

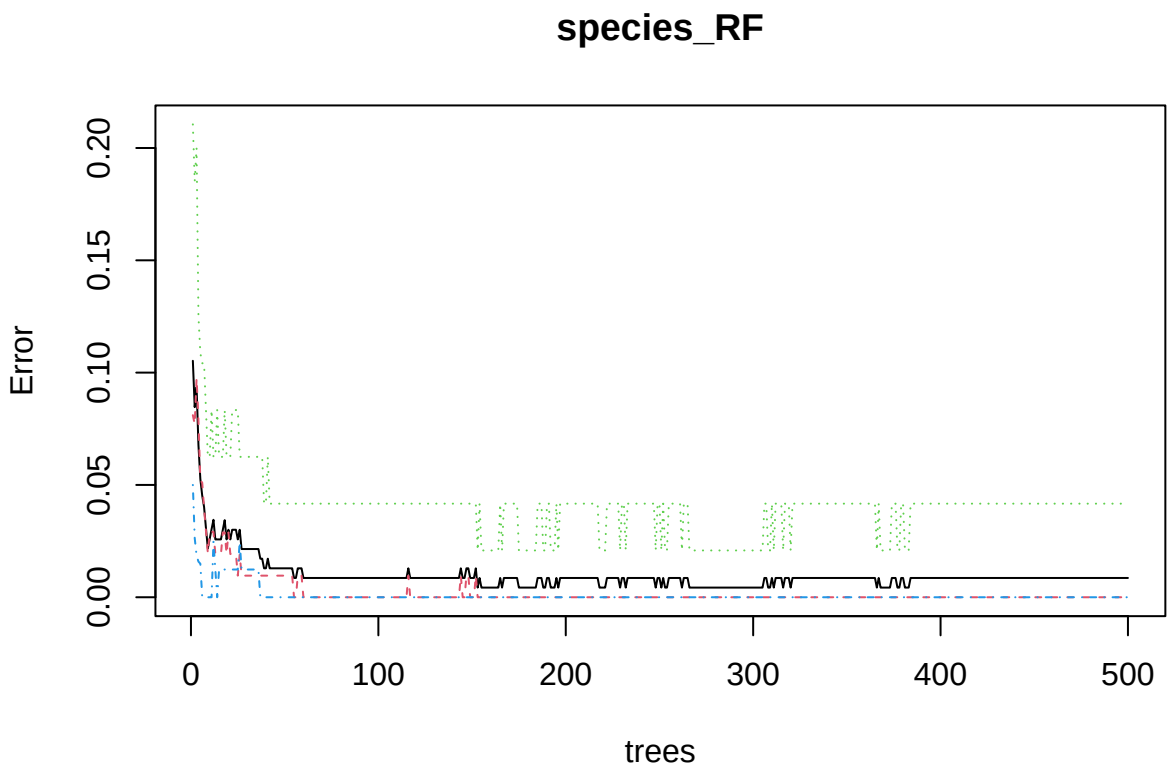
```
print(paste( "Misclassification error -", 0.02))
```

```
## [1] "Misclassification error - 0.02"
```

d)

Now, we try the same idea with more complex method - random forests.

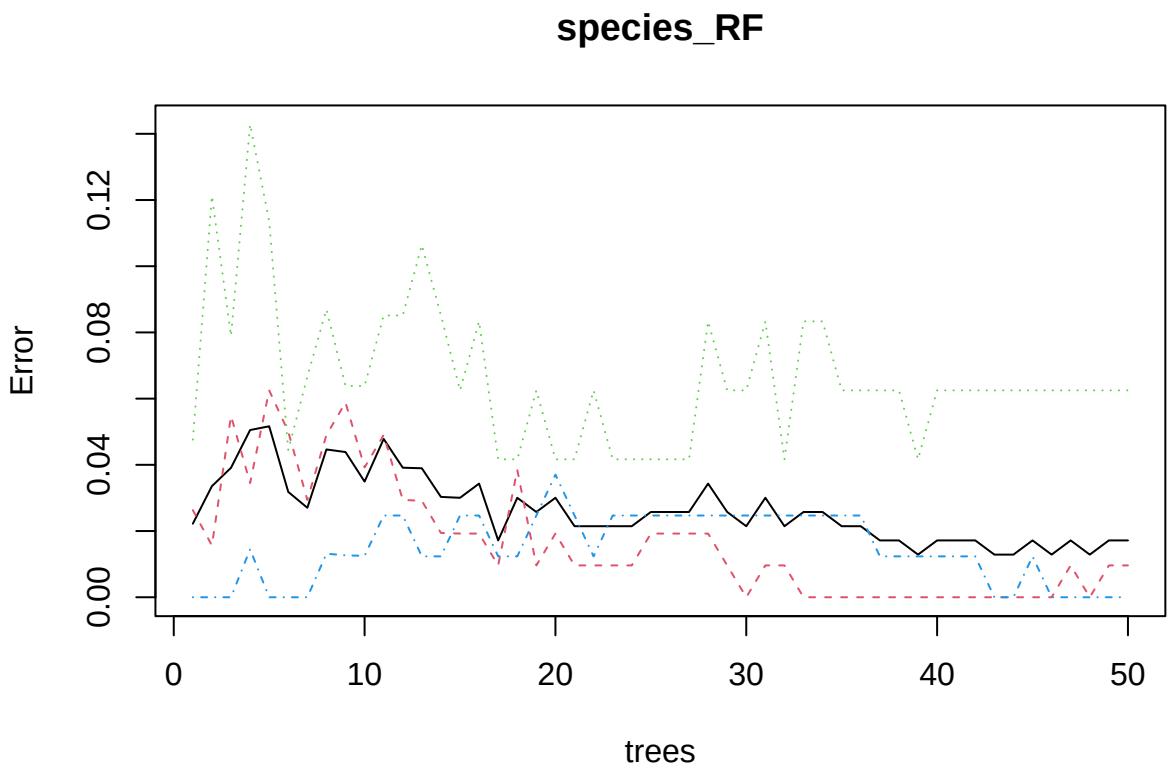
```
species_RF = randomForest(species ~ ., data = train,
                          mtry = 2, importance = TRUE)
plot(species_RF)
```



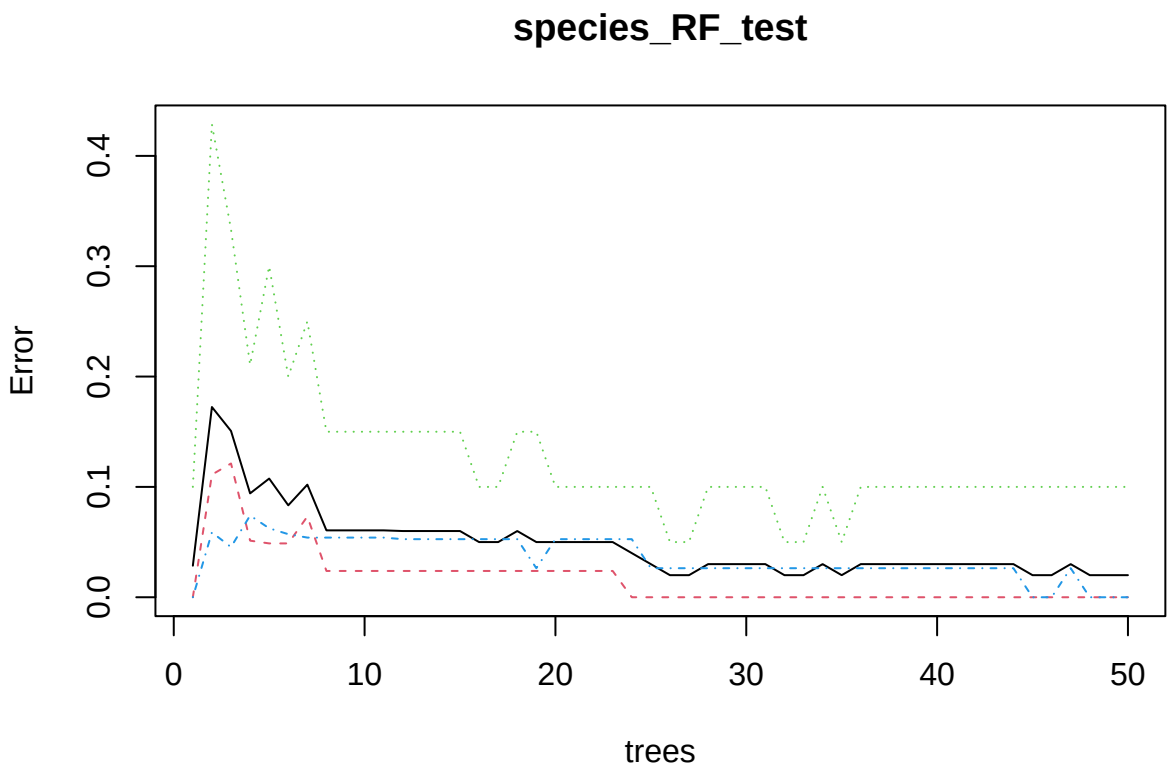
mtry - split number is found by taking square root of number of predictors ($\sqrt{6} \approx 2$)

From the graph, we can say that the best tree size is approximately 50.

```
species_RF = randomForest(species ~ ., data = train,
                          mtry = 2, ntree =50, importance = TRUE)
plot(species_RF)
```




```
species_RF_test = randomForest(species ~ ., data = test,
                           mtry = 2, ntree =50, importance = TRUE)
plot(species_RF_test)
```




```
yhat.rf = predict(species_RF_test, newdata = test)
misclass.rf = table(yhat.rf, test$species)
print(misclass.rf)
```

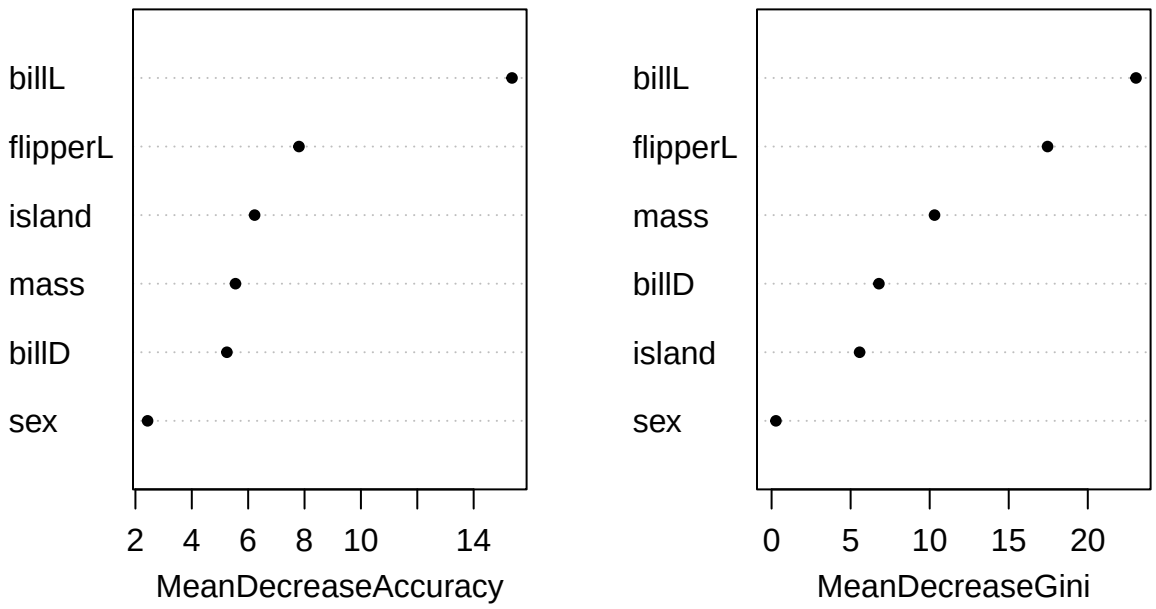
```
##
## yhat.rf     Adelie Chinstrap Gentoo
##   Adelie        42         0      0
##   Chinstrap      0        20      0
##   Gentoo         0         0     38
```

```
1-sum(diag(misclass.rf))/(sum(misclass.rf))
```

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

Importance

```
varImpPlot(species_RF_test, pch = 20, main = "")
```



From the importance graph, it is clearly seen that variables such as bill and flipperL are most influential.