Compulsory exercise 2: Group 38

TMA4268 Statistical Learning V2022

Lucas Michael Cammann, Ruben Mustad and Michinori Asaka Sciences, NTNU

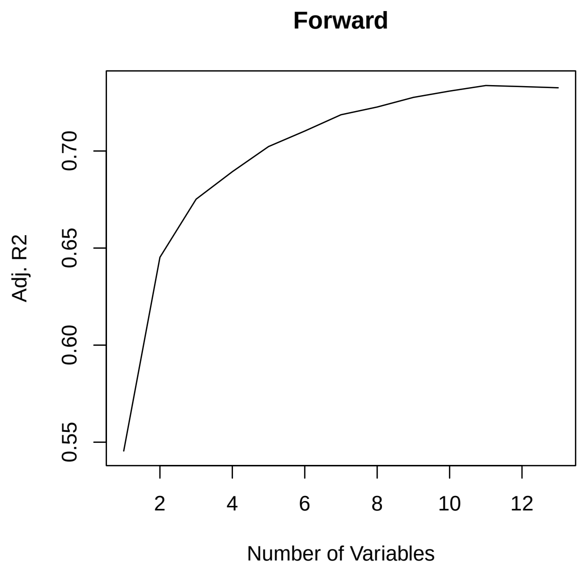
10 august, 2022

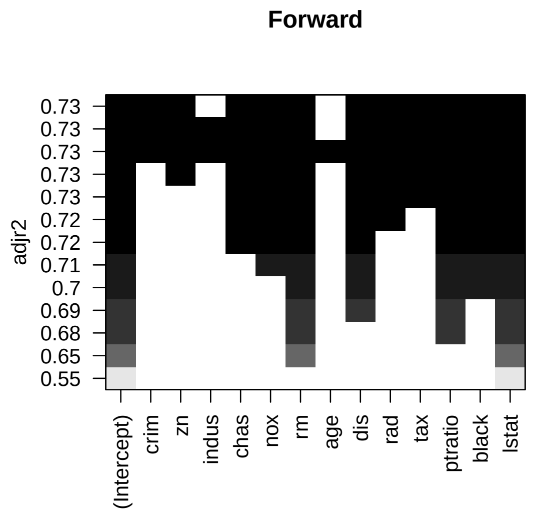
Problem 1

a)

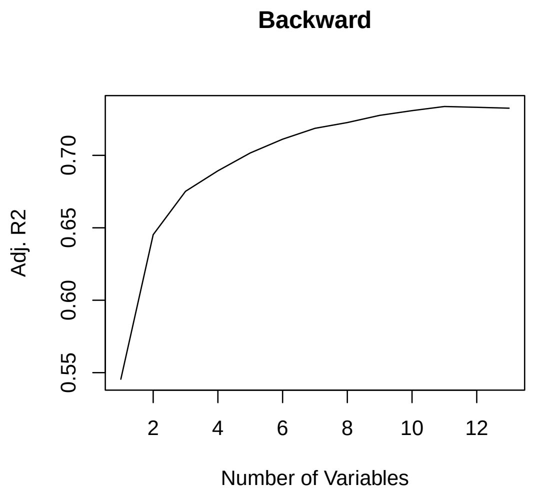
The Boston house dataset consists of 13 predictors and one response variable. We now want to use forward and backward stepwise selection to see how the predictors are going to affect our model.

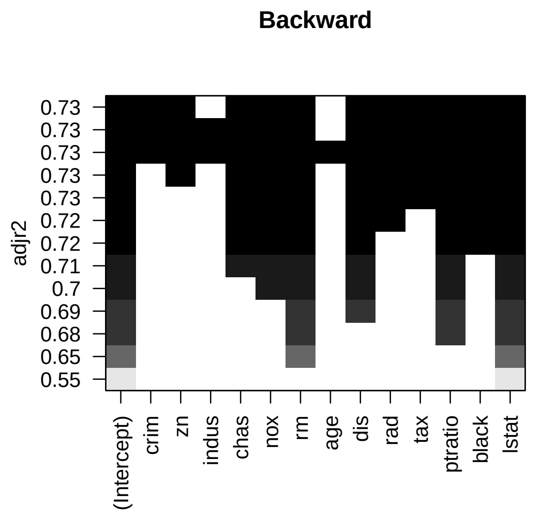
```r
par(mfrow = c(1, 2))
set.seed(1)
boston <- scale(Boston, center = T, scale = T)
train.ind = sample(1:nrow(boston), 0.8 * nrow(boston))
boston.train = data.frame(boston[train.ind, ])
boston.test = data.frame(boston[-train.ind, ])

# Forward
forward <- regsubsets(medv ~ ., data = boston.train, nbest = 1, nvmax = ncol(boston) -
    1, method = c("forward"))
forward_results = summary(forward)

plot(forward_results$adjr2, xlab = "Number of Variables", ylab = "Adj. R2", type = "l",
    main = "Forward")
plot(forward, scale = "adjr2", main = "Forward")
```





```
# Backward
backward <- regsubsets(medv ~ ., data = boston.train, nbest = 1, nvmax = ncol(boston) -
    1, method = c("backward"))
backward_results = summary(backward)

plot(backward_results$adjr2, xlab = "Number of Variables", ylab = "Adj. R2", type = "l",
    main = "Backward")
plot(backward, scale = "adjr2", main = "Backward")
```





From the figure above, we observe that both the forward and backward stepwise selections are quite similar. The best model (according to adjusted R^2) will be the model with every predictor in the base model except indus and age. However, we see that every predictor is included in the 3rd best model, which has only a slightly worse adjusted R^2 value.

b)

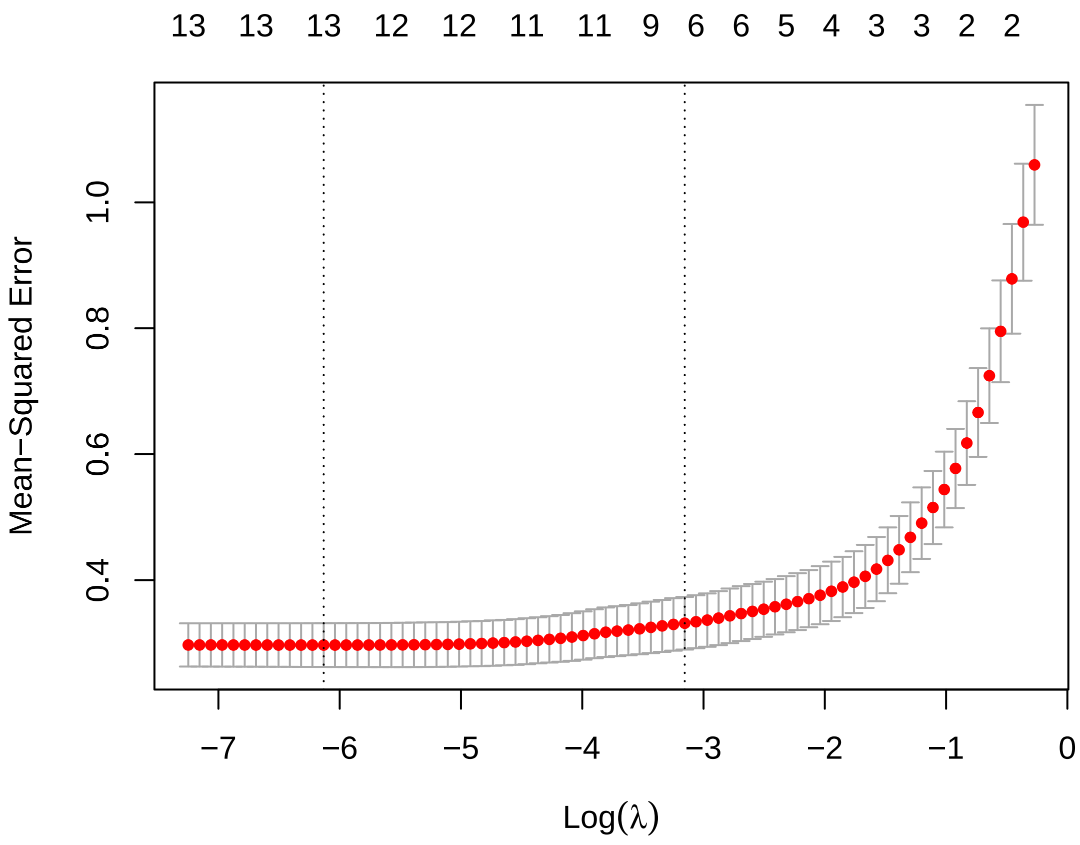
From the forward step-wise selection, we find that the four best predictors are

- rm: average number of rooms per dwelling
- dis: Weighted distances to five Boston employment centers
- ptratio: Pupil-teacher ratio by town
- lstat: Percentage of lower status of the population

which gives us an adjusted R^2 score of 0.69.

c)

```r
set.seed(1)

# Divide the training and test set into X (predictors) and y (response) for
# train and test
X_train = as.matrix(boston.train[, 1:13])
y_train = as.matrix(boston.train[, 14])
X_test = as.matrix(boston.test[, 1:13])
y_test = as.matrix(boston.test[, 14])


# Ridge: alpha = 0, Lasso: alpha = 1
lasso_cv = cv.glmnet(X_train, y_train, alpha = 1, nfold = 5)
plot(lasso_cv)
```



```
# Find the lambda that gives us the smallest MSE
lam = lasso_cv$lambda
best_lambda = lasso_cv$lambda.min
sprintf("The best lambda value is %#.5f", best_lambda)
```

```
## [1] "The best lambda value is 0.00217"
```

```
# Train Lasso on the training set with the smallest lambda
lasso_4c = glmnet(X_train, y_train, alpha = 1, lambda = best_lambda)

# Make predictions on the training set and test set
lasso_train = predict(lasso_4c, X_train)
sprintf("The MSE on the training set is %#.4f", mean((lasso_train - y_train)^2))
```

```
## [1] "The MSE on the training set is 0.2757"
```

```
lasso_test = predict(lasso_4c, X_test)
sprintf("The MSE on the test set is %#.4f", mean((lasso_test - y_test)^2))
```

```
## [1] "The MSE on the test set is 0.2045"
```

The best λ value (i.e., the one that gave us the smallest MSE) found using a 5-fold CV is 0.00217. From the plot, we observe that the MSE decreases as the λ (or $\log(\lambda)$) decreases until around -5, after this, the change in MSE is quite small.

The MSE on the test set is 0.2045, while the MSE on the training set is 0.2757. It is a bit unusual to get better performance on the test set than on the training set, as we trained on the training set, but we will get back to this in Problem 3.

Using a seed of 1, we see that Lasso gives us no zero coefficients. If we did not use a seed, we observed that the predictor indus had zero as a coefficient sometimes. Indus is also the coefficient with the lowest value when using a seed of 1.

```
coef(lasso_4c)  # get the coefficients
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.023622904
## crim        -0.081992849
## zn           0.094717791
## indus        0.002619428
## chas         0.087341100
## nox         -0.175365927
## rm           0.312648954
## age         -0.011212120
## dis         -0.317143728
## rad          0.270168177
## tax         -0.207314714
## ptratio     -0.204052488
## black        0.102877803
## lstat       -0.428298373
```

d)

1. When comparing computational speed between step-wise feature selection methods and Lasso for features selection, Lasso is much faster: True
2. It is easier for ridge regression than Lasso to result in coefficients equal zero, namely due to the quadratic penalization term in ridge: False
3. For the purpose of feature selection, both Ridge and Lasso are equally appropriate: False
4. Elastic Net is a combination of Lasso and Ridge: True

Problem 2 (6P)

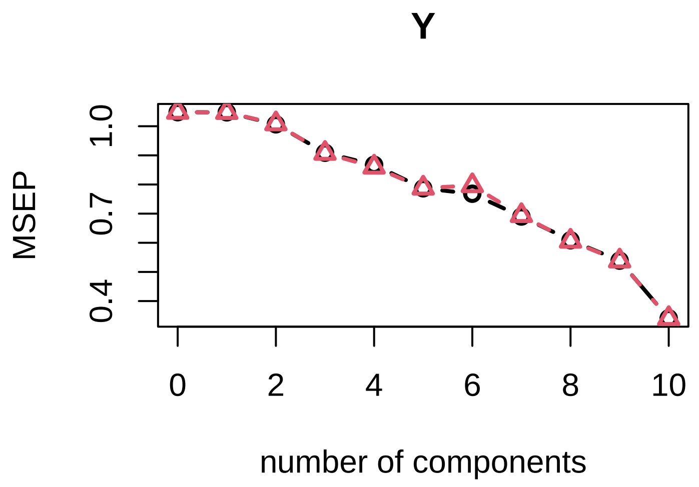
a) (2P)

Fit PCR:

```
pcr_model <- pcr(Y ~ ., data = synthetic.train, scale = TRUE, validation = "CV")
```

Here is a graph of MSEP for PCR model:

```r
validationplot(pcr_model, val.type = "MSEP", type = "b", lwd = 2)
```

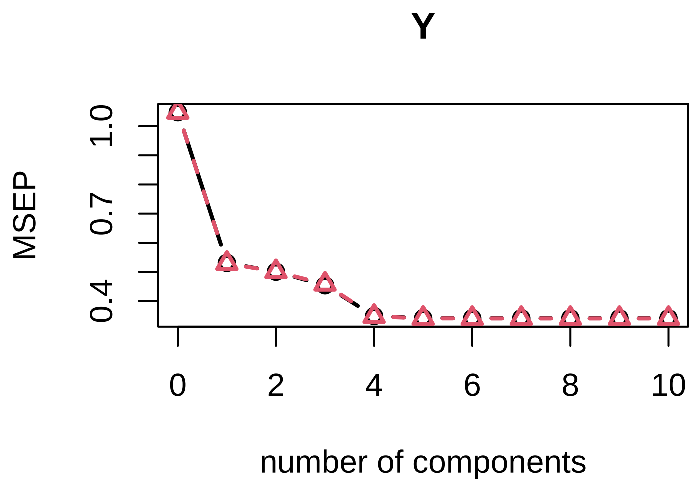


Fit PLSR:

```r
pls_model <- plsr(Y ~ ., data = synthetic.train, scale = TRUE, validation = "CV")
```

Here is a graph of MSEP for PLSR model:

```r
validationplot(pls_model, val.type = "MSEP", type = "b", lwd = 2)
```

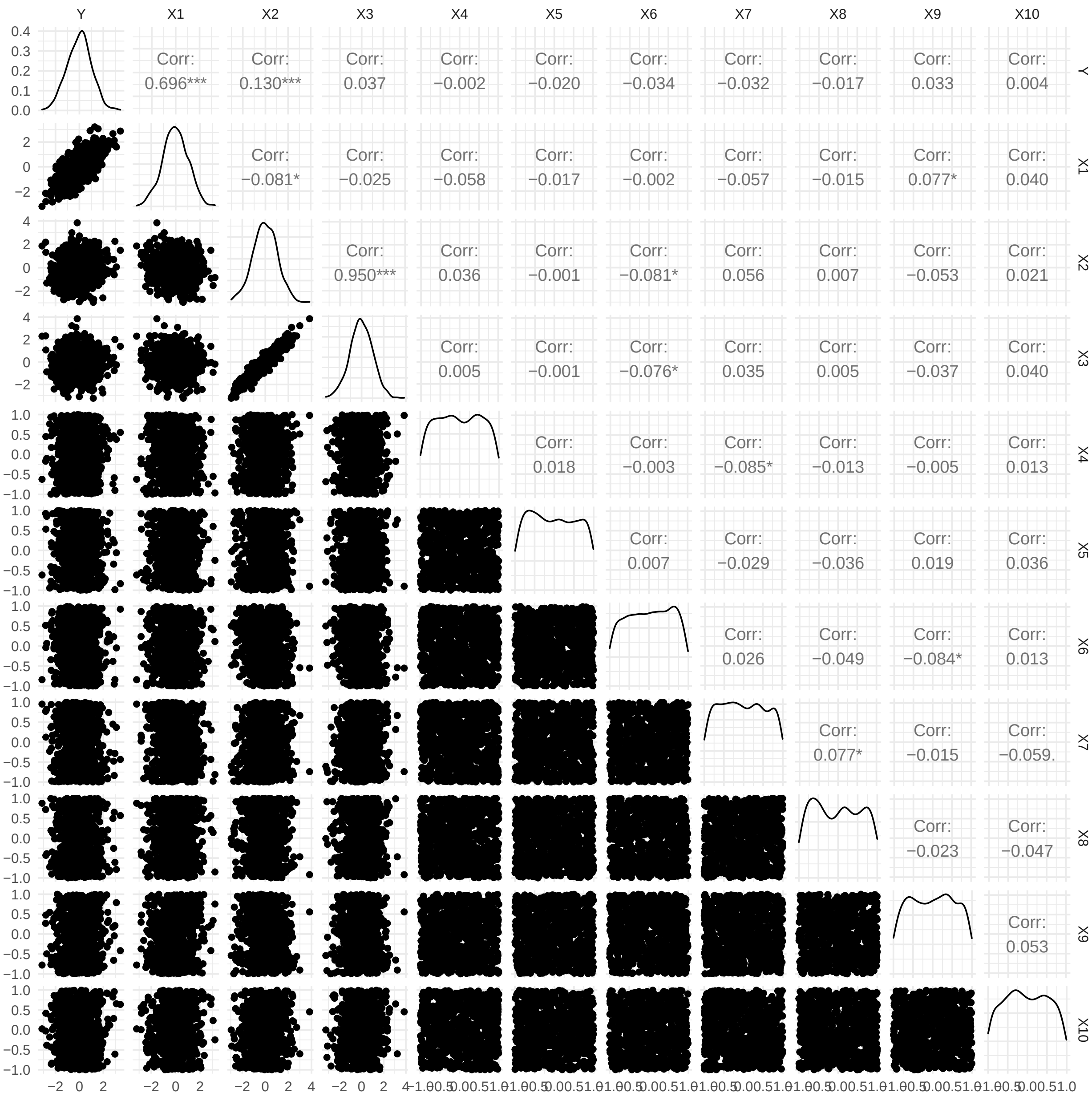


b) (4P)

MSEP of PCR shows gradual decrease as a function of number of components, while that of PLS shows sharp decrease at a number of components of 1 and is quickly stabilized at a number of component of 4.

This is associated with characteristics of PLR and PCR. In PCR, the dimensionality reduction is done via an unsupervised method (PDA); It is therefore not garanteed that the directions that best explain the predictors will also be the best directions to use for predicting the response. On the other hand, PLS puts highest weight on the variables that are most strongly related to the response when obtaining the first PLS direction.

Large MSEP reduction at a number of component of 1 is therefore expected if one of the predictor variables has strong correlation with the response. This is the case here, since a predictor variable X1 has strong correlation with the response:



Problem 3

a)

1. For the polynomial regression (where polynomial functions of features are used as predictors), variance increases when including predictor with a high order of the power: True

2. If the polynomial functions from (1) are replaced with step functions, then the regression model is too simple to be overfitted on a dataset even with multiple cutpoints: False
3. The smoothing spline ensures smoothness of its function, g, by having a penalty term $\int g'(t)^2 \, dt$ in its loss: True
4. The K-nearest neighbors regression (local regression) has a high bias when its parameter, k, is high: True

b)

We will now fit an additive model on the Boston training set with the predictors rm, ptratio and lstat. We will let ptratio be a smoothing spline with 3 degrees of freedom and lstat be a 2nd-degree polynomial.

```
model_task3 = gam(medv ~ rm + s(ptratio, df = 3) + poly(lstat, 2), data = boston.train)
# summary(model_task3)


gam_train = predict(model_task3, boston.train)
sprintf("The MSE on the training set is %#.4f", mean((gam_train - y_train)^2))
```

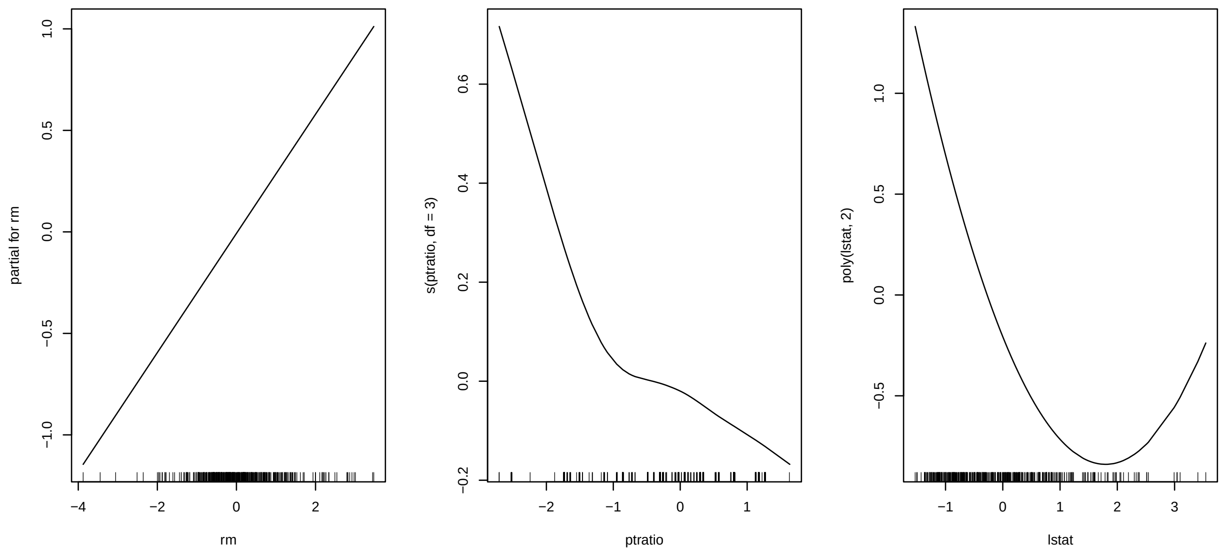
```
## [1] "The MSE on the training set is 0.2772"
```

```
gam_test = predict(model_task3, boston.test)
sprintf("The MSE on the test set is %#.4f", mean((gam_test - y_test)^2))
```

```
## [1] "The MSE on the test set is 0.2335"
```

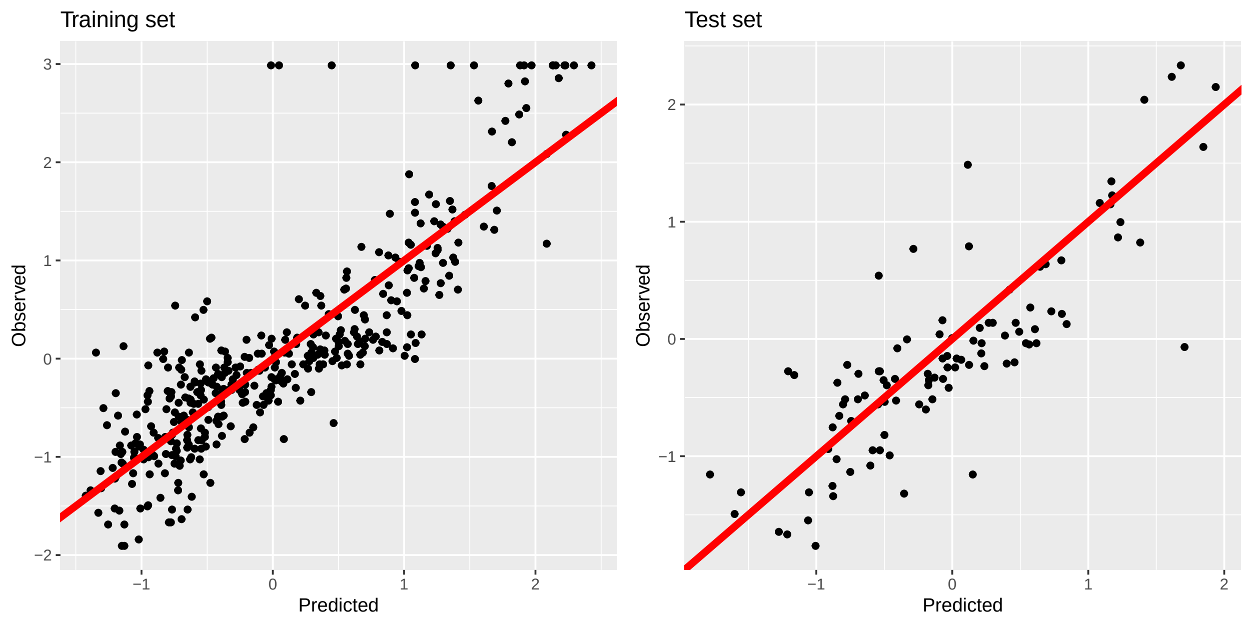
Below we plot the polynomials and the spline.

```
par(mfrow = c(1, 3))
plot(model_task3)
```



We again observe that we get better MSE on the test set than on the training set again. To understand why, it can be smart to do some plotting, so we decided to plot the observed values vs predicted values. We see that in the training set, there are quite a few observed values that have a value of 3, that our model does not manage to predict well. In the test set, we don't see those, causing us to get a better MSE on the test set.

```r
data_mod <- data.frame(Predicted = predict(model_task3, boston.train), Observed = boston.train$medv)

data_mod_test <- data.frame(Predicted = predict(model_task3, boston.test), Observed = boston.test$medv)

plot1 <- ggplot(data_mod, aes(x = Predicted, y = Observed)) + ggtitle("Training set") +
    geom_point() + geom_abline(intercept = 0, slope = 1, color = "red", size = 2)

plot2 <- ggplot(data_mod_test, aes(x = Predicted, y = Observed)) + ggtitle("Test set") +
    geom_point() + geom_abline(intercept = 0, slope = 1, color = "red", size = 2)

ggarrange(plot1, plot2, ncol = 2, nrow = 1)
```



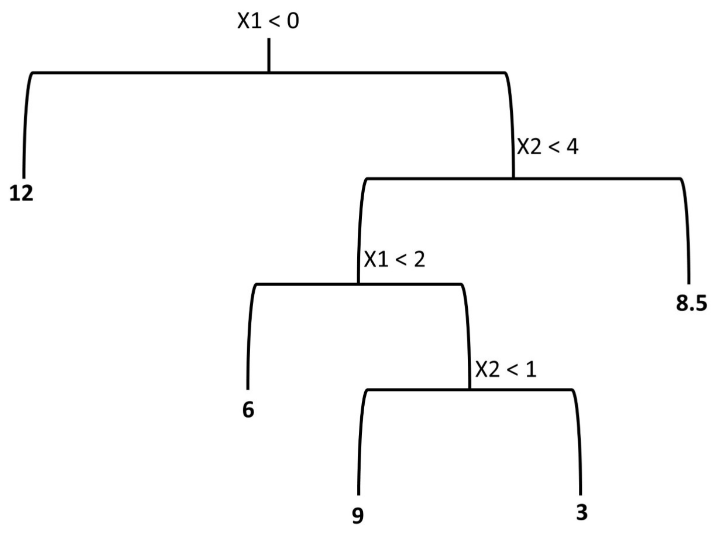
Problem 4 (11P)

a) (2P) - Multiple choice

1. A downside of simple regression trees is that they cannot handle interaction terms: False
2. In boosting, the parameter d controls the number of splits allowed in each try. When $d = 2$, we allow for models with 2-way interactions: True
3. The random forest approach improves bagging, because it reduces the variance of the predictor function by decorrelating the trees: True
4. The number of trees B in boosting is a tuning parameter: True

1: False 2: True 3: True 4: True

b) (2P)



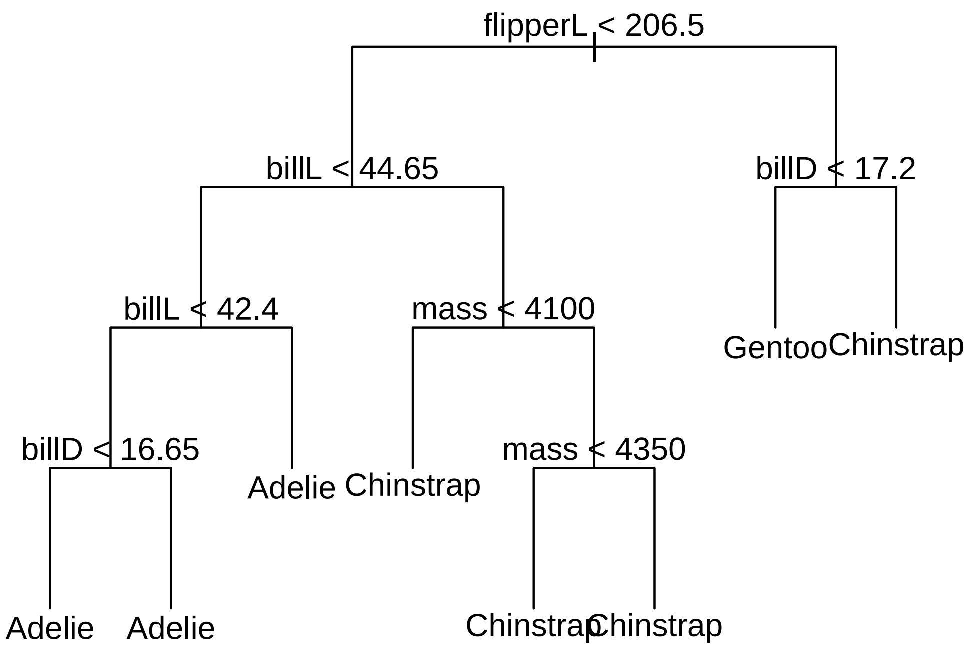
c) (4P)

1)
Generate a simple classification tree using the Gini index:

```
tree.Penguins = tree(species ~ ., data = train, split = "gini")
```

Plot of the resulting tree:

```
plot(tree.Penguins, type = "uniform")
text(tree.Penguins, pretty = 1)
```



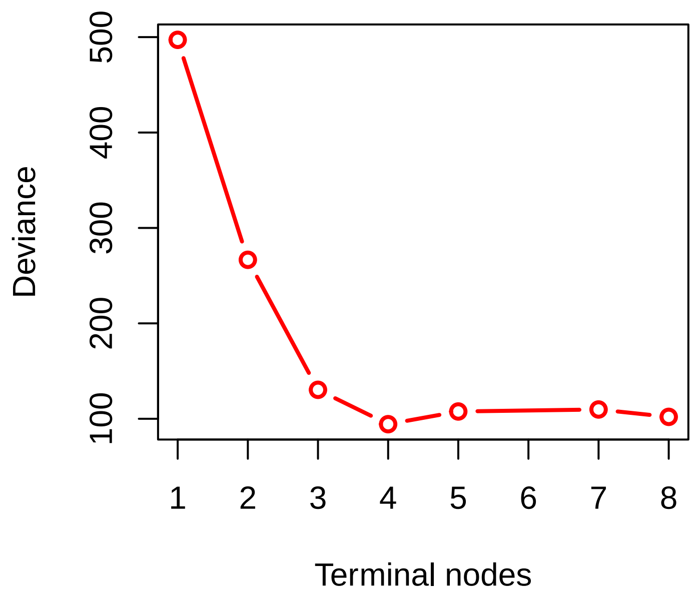
2)

Apply cost-complexity pruning using 10-fold CV:

```r
set.seed(123)
tree.Penguins <- tree(species ~ ., data = train)
cv.Penguins <- cv.tree(tree.Penguins, K = 10)
```

Plot the deviance as a function of tree size:

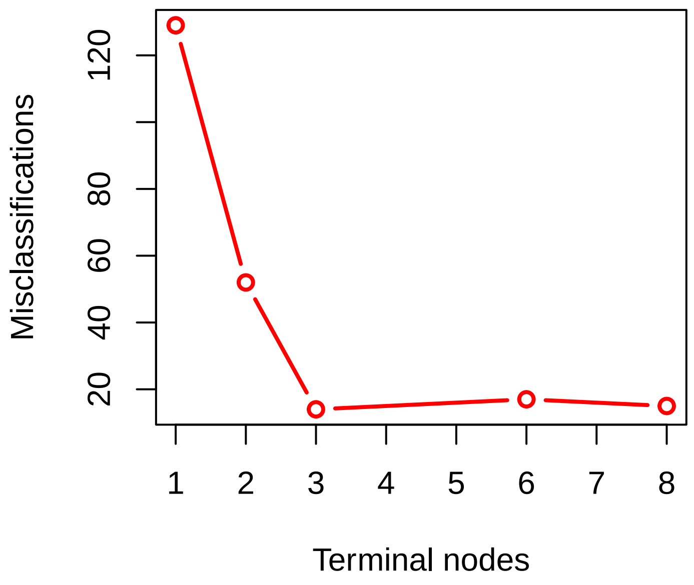
```r
plot(cv.Penguins$dev ~ cv.Penguins$size, type = "b", lwd = 2, col = "red", xlab = "Terminal nodes",
    ylab = "Deviance")
```



This plot suggests that 4 leaves would work well.

3)
Run cross-validation and plot the number of misclassifications as a function of tree size:

```
set.seed(123)
tree.Penguins <- tree(species ~ ., data = train)
cv.Penguins = cv.tree(tree.Penguins, FUN = prune.misclass)
plot(cv.Penguins$dev ~ cv.Penguins$size, type = "b", lwd = 2, col = "red", xlab = "Terminal nodes",
    ylab = "Misclassifications")
```



This plot suggests that 3-8 nodes gives similarly low misclassifications.
So we choose 3 terminal nodes, which is the smallest model (lowest variance), as an optimal model parameter.

Perform prediction on the test set with 3 nodes and calculate the misclassification rate:

```
bestmod.Penguins <- prune.tree(tree.Penguins, best = 3)
tree.pred = predict(bestmod.Penguins, newdata = test, type = "class")
misclass.tree = table(tree.pred, test$species)
MER <- (nrow(test) - sum(diag(misclass.tree)))/nrow(test)
```

The misclassification rate is:

```
## [1] 0.06
```

d) (3P)

Construct a classification tree based on random forests.
We set mtry = 2, since it is approximately equal to sqrt(number of variables):

```
rf.Penguins = randomForest(species ~ ., data = train, mtry = 2, importance = TRUE)
```

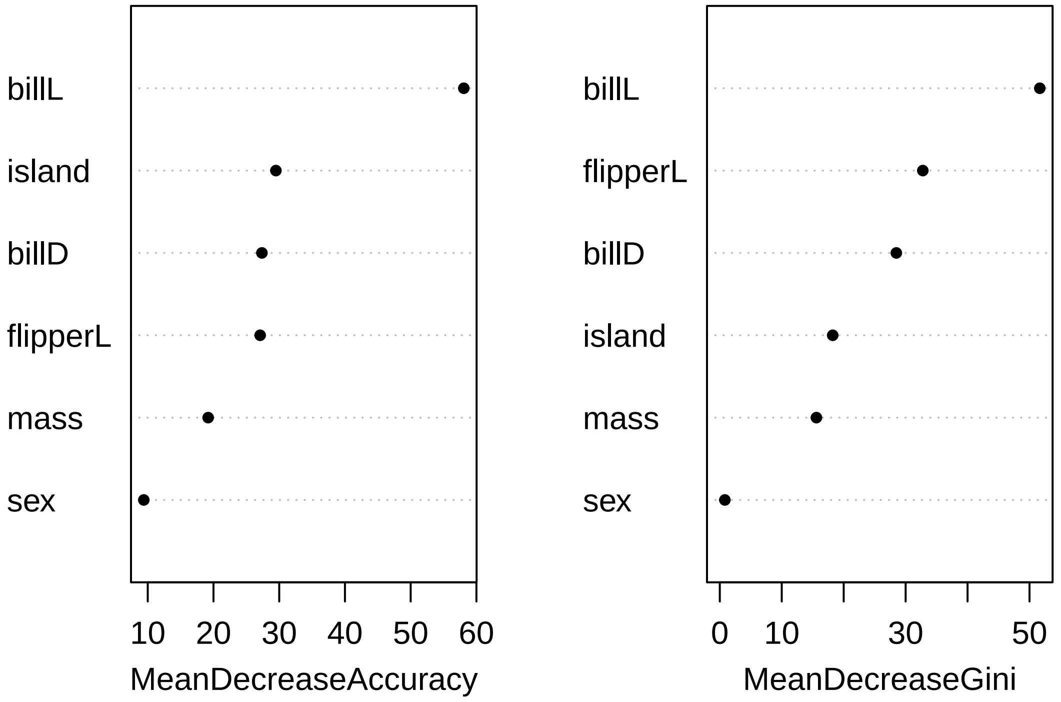
Perform prediction on the test set and calculate the misclassification rate:

```
rf.pred = predict(rf.Penguins, newdata = test)
misclass.rf = table(rf.pred, test$species)
MER <- (nrow(test) - sum(diag(misclass.rf)))/nrow(test)
```

The misclassification rate is:

```
## [1] 0.02
```

The variable importance plots suggest that bill length and flipper length are the most influential parameters in the prediction of the species:



Problem 5 (6P)

a) (2P) - Multiple choice

1. Logistic regression is the preferred method for this data set, because it gives nice interpretable parameter estimates: False
2. In this dataset we are guaranteed to find a separating hyperplane, unless there are exact feature ties (two patients with the exact same gene data, but different outcome): True
3. When fitting a support vector classifier, we usually have to standardize the variables first: True
4. By choosing a smaller budget parameter C we are making the model less biased, but introduce more variance: True

b) (4P)

(i) (2P)

```r
# First for the linear boundary
linres <- tune.svm(species ~ ., data = train, kernel = "linear", cost = c(0.1, 1,
    10))
linerror <- linres$best.performance
lincost <- linres$best.model$cost
# Then for the radial boundary
```

```r
radres <- tune.svm(species ~ ., data = train, kernel = "radial", cost = c(0.1, 1,
    10), gamma = c(10^-2, 10^-1, 1))
raderror <- radres$best.performance
radparams <- radres$best.parameters
```

The best achieved training error rate for the support vector classifier is 0.0043478 with a cost parameter of 0.1‘ and for the support vector machine 0‘ with the parameter combination of $0.1, 10$‘.

(ii) (1P) Report the confusion tables and misclassification error rates for the test set in both cases, using the best parameters you found in (i).

```r
linpred <- predict(linres$best.model, test)
confmlin <- confusionMatrix(linpred, test$species)$table
missclin <- mean(linpred != test$species)
radpred <- predict(radres$best.model, test)
confmrad <- confusionMatrix(radpred, test$species)$table
misscrad <- mean(radpred != test$species)
```

The confusion matrix for the support vector classifier is $42, 0, 0, 0, 20, 0, 0, 0, 38$, and for the support vector machine is $41, 1, 0, 1, 19, 0, 0, 0, 38$. The misclassification rate is 0 in the former case and 0.02 in the latter.

(iii) (1P) Which classifier do you prefer and why? In the current scenario, the support vector classifier (linear boundary) is to be preferred. The reason for this is twofold: Firstly, the support vector classifier is a simpler classifier with only one hyperparameter, which makes the tuning and interpretation easier for the analyst and the respective audience. Secondly, the support vector classifier has actually been shown to outperform the support vector machine on basis of the test misclassification error, as shown above.

Problem 6 (12P)

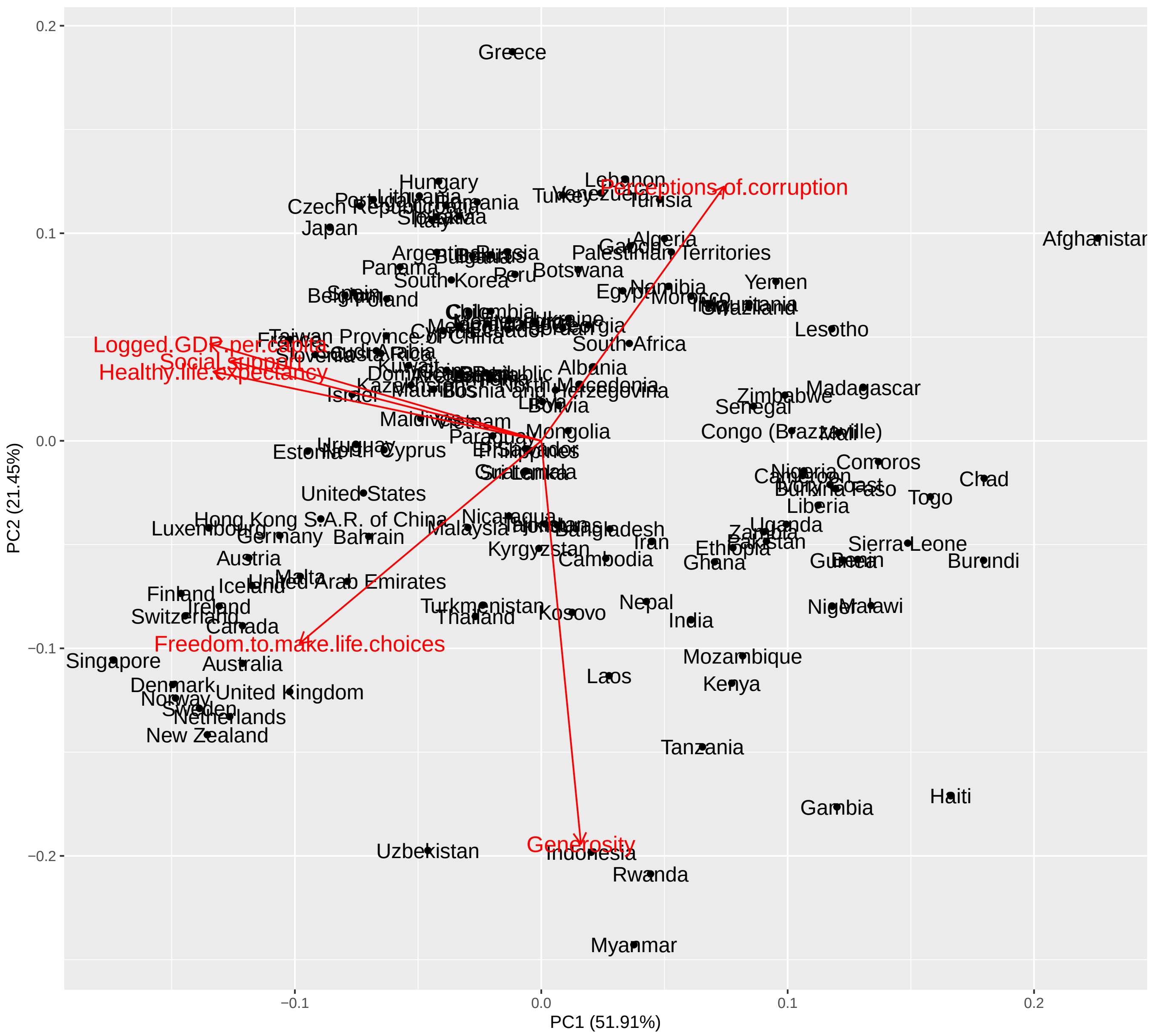
```r
# load a synthetic dataset
id <- "1NJ1SuUBebl5P8rMSIwm_n3S8a7K43yP4" # google file ID
happiness <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id),
                      fileEncoding="UTF-8-BOM")

colnames(happiness)
```

```
##  [1] "Country.name"
##  [2] "Regional.indicator"
##  [3] "Ladder.score"
##  [4] "Standard.error.of.ladder.score"
##  [5] "upperwhisker"
##  [6] "lowerwhisker"
##  [7] "Logged.GDP.per.capita"
##  [8] "Social.support"
##  [9] "Healthy.life.expectancy"
## [10] "Freedom.to.make.life.choices"
## [11] "Generosity"
## [12] "Perceptions.of.corruption"
## [13] "Ladder.score.in.Dystopia"
## [14] "Explained.by..Log.GDP.per.capita"
## [15] "Explained.by..Social.support"
## [16] "Explained.by..Healthy.life.expectancy"
## [17] "Explained.by..Freedom.to.make.life.choices"
## [18] "Explained.by..Generosity"
## [19] "Explained.by..Perceptions.of.corruption"
## [20] "Dystopia...residual"
```

```
cols = c('Country.name',
         'Ladder.score',  # happiness score
         'Logged.GDP.per.capita',
         'Social.support',
         'Healthy.life.expectancy',
         'Freedom.to.make.life.choices',
         'Generosity',  # how generous people are
         'Perceptions.of.corruption')

# We continue with a subset of 8 columns:
happiness = subset(happiness, select = cols)
rownames(happiness) <- happiness[, c(1)]

# And we creat an X and a Y matrix
happiness.X = happiness[, -c(1, 2)]
happiness.Y = happiness[, c(1, 2)]
happiness.XY = happiness[, -c(1)]

# scale
happiness.X = data.frame(scale(happiness.X))


str(happiness)
```

```
## 'data.frame':    149 obs. of  8 variables:
##  $ Country.name                : chr  "Finland" "Denmark" "Switzerland" "Iceland" ...
##  $ Ladder.score                : num  7.84 7.62 7.57 7.55 7.46 ...
##  $ Logged.GDP.per.capita       : num  10.8 10.9 11.1 10.9 10.9 ...
##  $ Social.support              : num  0.954 0.954 0.942 0.983 0.942 0.954 0.934 0.908 0.948 0.934 ..
##  $ Healthy.life.expectancy     : num  72 72.7 74.4 73 72.4 73.3 72.7 72.6 73.4 73.3 ...
##  $ Freedom.to.make.life.choices: num  0.949 0.946 0.919 0.955 0.913 0.96 0.945 0.907 0.929 0.908 ...
##  $ Generosity                  : num  -0.098 0.03 0.025 0.16 0.175 0.093 0.086 -0.034 0.134 0.042 ..
##  $ Perceptions.of.corruption   : num  0.186 0.179 0.292 0.673 0.338 0.27 0.237 0.386 0.242 0.481 ...
```

```
pca_mat = prcomp(happiness.X, center = T, scale = T)

# Score and loadings plot:
autoplot(pca_mat, data = happiness.X, colour = "Black", loadings = TRUE, loadings.colour = "red",
    loadings.label = TRUE, loadings.label.size = 5, label = T, label.size = 4.5)
```



a) (3P)

(i) One obvious observation is the strong correlation between the variables `Logged.GDP.per.capita`, `Social.support` and `Healthy.life.expectancy` as shown by the small angle between these. It appears that these variables go hand-in-hand with each other, e.g. that a country with high levels of social support will most likely also have a good economic outlook in terms of GDP per capita. Conversely, there seems to be a negative correlation between the `Freedom.to.make.life.choices` and `Perceptions.of.corruption` as indicated by the almost directly opposite angles of the two. Evidently, living in a corrupt country in which financial assets are required to buy favors will seriously hamper the own ability to pursue certain goals in life.

(ii) Afghanistan can be considered an outlier among the above given countries. It is at the most far right corner of the plot (highest value of PC1) and is not near any of the visible clusters.

b) (4P)

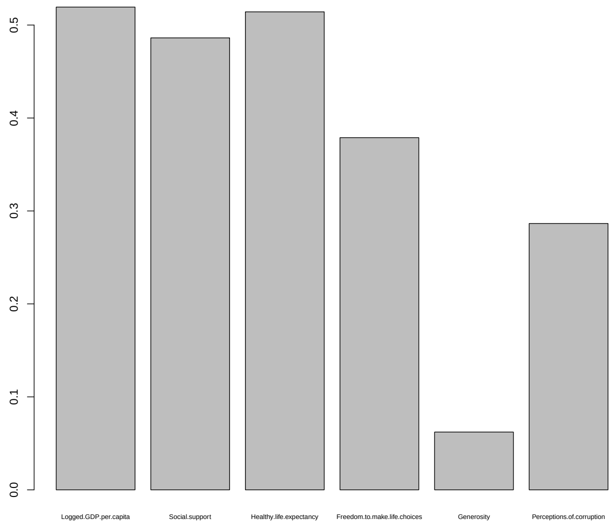
Here, we're going to find out which variables are important by principal component analysis (PCA) and partial least squares regression (PLSR).

Note that we can naturally assume the followings:

- PCA will find out important variables w.r.t explainability of the dataset of the predictors.
- PLSR can find out important variables w.r.t the response in the model, that is, happiness (= `Ladder.score`).

(i) Make a graphical description of the absolute values of the first principal component (= `PC1`) by PCA. You can use a bar plot, or any other graphical description of your choice (see R-hints below). (1P)

```
vals <- abs(data.frame(pca_mat$rotation)$PC1)
barplot(vals, width = 3, names.arg = c(cols[3:8]), cex.names = 0.6)
```



(ii) Fit PLSR on `happiness.XY` with a response of `Ladder.score` (= happiness score) and all the remaining variables in that dataset as predictors. (1P)

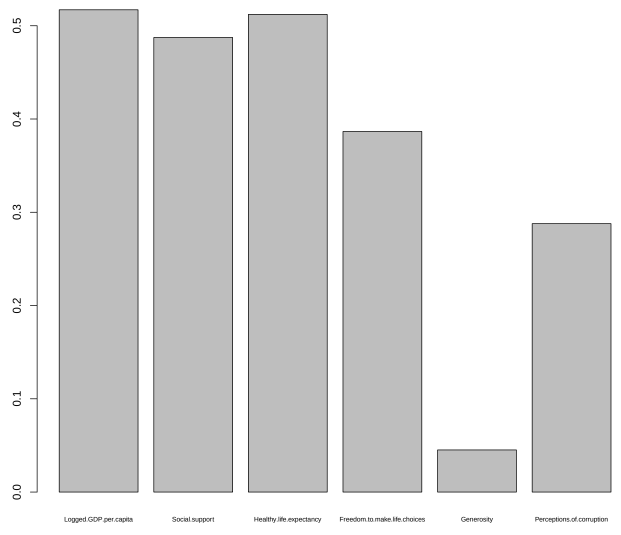
```
plsr_model <- plsr(happiness.XY$Ladder.score ~ ., data = happiness.XY, scale = T)
summary(plsr_model)
```

```
## Data:    X dimension: 149 6
##  Y dimension: 149 1
## Fit method: kernelpls
## Number of components considered: 6
## TRAINING: % variance explained
##                            1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X                            51.87    68.64    84.48    88.10    94.12   100.00
## happiness.XY$Ladder.score    75.10    75.50    75.55    75.58    75.58    75.58
```

```
vals_plsr <- plsr_model$loadings[, c("Comp 1")]
```

(iii) Plot a bar graph of the absolute values of the first principal component for `X` (= predictors of `happiness.XY`) by PLSR. Use the same type of plot as in (i) in order to compare. (1P)

```
barplot(abs(vals_plsr), width = 3, cex.names = 0.6)
```



(iv) What are the three most important predictor to predict the happiness score based on the PLSR bar graph from (iii)? (1P) Based on the barplot in (iii), the three most important predictors to assess the happiness score are:
  (a) `Logged.GDP.per.capita`
  (b) `Social.support`
  (c) `Healthy.life.expectancy`

c) (2P) - Multiple choice

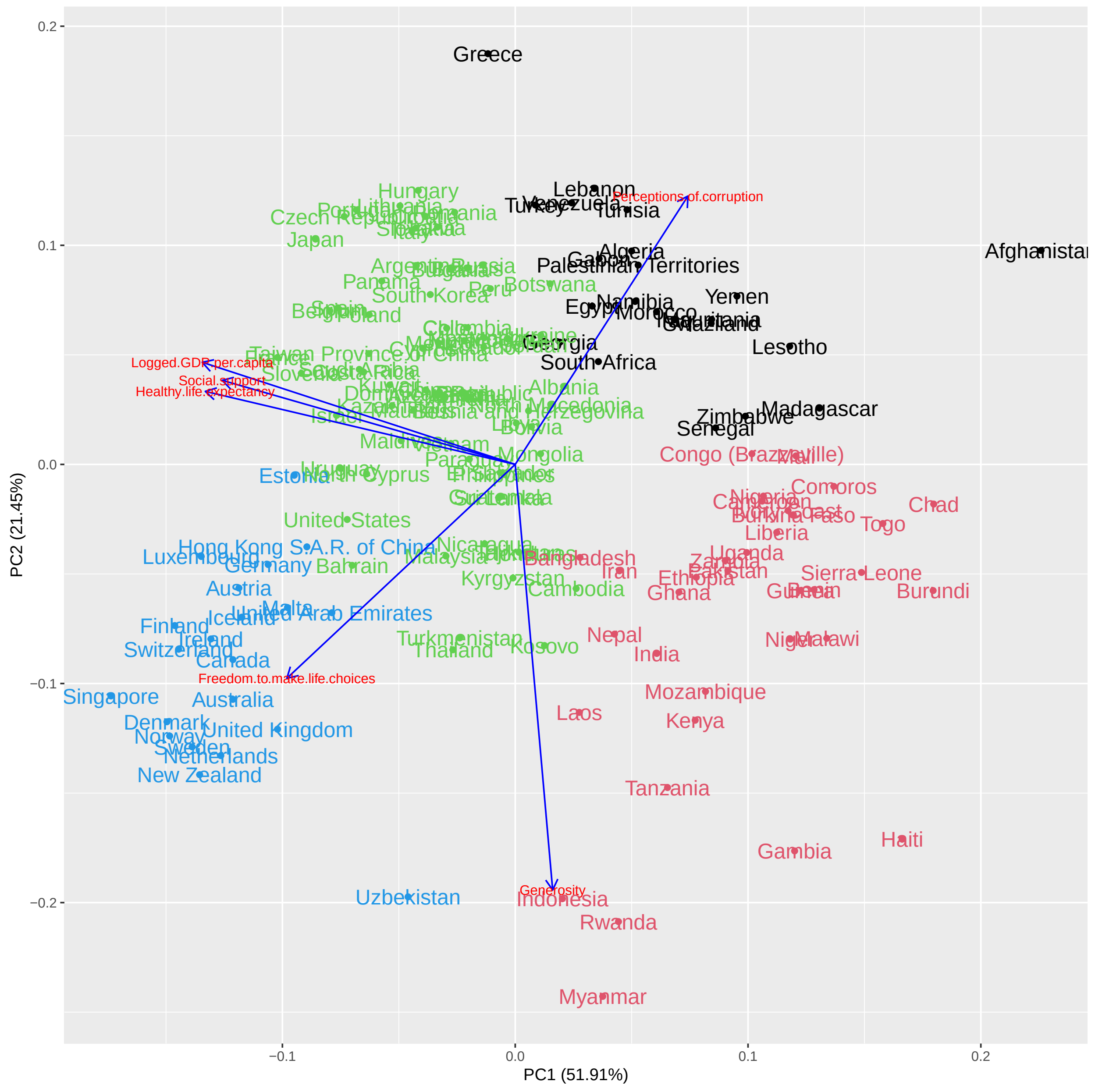
Say for *each* of them if it is true or false.

1. K-means is optimizing clusters such that the within-cluster variance becomes large: False
2. No matter how many times you run K-means clustering, its cluster centroids will always end up in the same locations: False
3. Strong correlation between predictors allows PCR to be more effective for predicting a response when prediction is made based on the first two principal components: False
4. We can do outlier/anomaly detection with PCA: True

d) (3P)

(i) One easy way to achieve the desired clustering is by setting K = 4.

```
K = 4  # your choice
km.out = kmeans(happiness.X, K)

autoplot(pca_mat, data = happiness.X, colour = km.out$cluster, label = T, label.size = 5,
    loadings = T, loadings.colour = "blue", loadings.label = T, loadings.label.size = 3)
```



(ii) In a first instance we calculate the mean Ladder scores for each of the clusters, to later interpret them in the light of the previous observations.

```
# Sort the output according to the cluster
clusters.sort <- sort(km.out$cluster)
# Extract indexes for the clusters
clusters.score <- vector(, K)
Ind <- vector(, K)
Ind[1] = 1
for (i in 1:3) {
    Ind[i + 1] <- min(which(clusters.sort > i))
    names <- names(clusters.sort[Ind[i]:Ind[i + 1] - 1])
    clusters.score[i] <- mean(happiness.XY[names, "Ladder.score"])
}
names <- names(clusters.sort[Ind[i + 1]:149])
```

```r
clusters.score[4] <- mean(happiness.XY[names, "Ladder.score"])
print(clusters.score)
```

```
## [1] 4.463091 4.516514 5.841301 7.030952
```

It becomes apparent that there is a clear hierarchy in the ladder scores for the four produced clusters. The cluster which is centered at low values of both principal components and which includes countries such as Denmark and Norway has the highest ladder score, while the cluster at the far most right of the plot, at high values of PC1, has the lowest ladder score. The second happiest countries are those which are centered near moderately low values of PC1 and moderately high values of PC2, including countries such as South Korea and Japan. Countries which are centered around low values of PC2, such as e.g. Kosovo, are the third happiest in the presented clustering.

These results can be interpreted when considering the loading directions shown in the above plot. It can be argued based on the mean ladder score of the second happiest cluster together with the loading directions of GDP per capita, social support and healthy life expectancy that "hard factors" (e.g. good personal economic outlook) provide a certain level of minimal satisfaction across countries. However, cultural "soft factors" may also play a significant role, which can be exhibited when analyzing the happiest cluster in the lower left corner of the plot: Clearly, the countries present therein also have very good economic conditions (as indicated by low values of PC1), however, as evident by the low values of PC2 they also provide some combination of generosity and, importantly, the freedom to pursue one's own life choices. The impact of social aspects on the happiness of the poorer countries is not as straightforward to assess, as here the economic outlook seems to be most important. As such, the unhappiest countries exhibit highest values of PC1, and coincidentally among those are some of the poorest countries of the world (e.g. Burundi, Sierra Leone, Madagascar and Afghanistan). The above interpretations can hence be summarized as follows: While improving the economic conditions for poor countries will provide highest improvement in overall life satisfaction, richer countries with a certain minimal level of wealth will most likely respond to improvements in social aspects, such as the freedom to make life choices.