Compulsory exercise 2: Group 37

TMA4268 Statistical Learning V2022

Oskar Jørgensen & Halvor L. Henriksen

04 april, 2022

Problem 1

```
library(MASS)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
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, ])
boston.test = data.frame(boston[-train.ind, ])
```

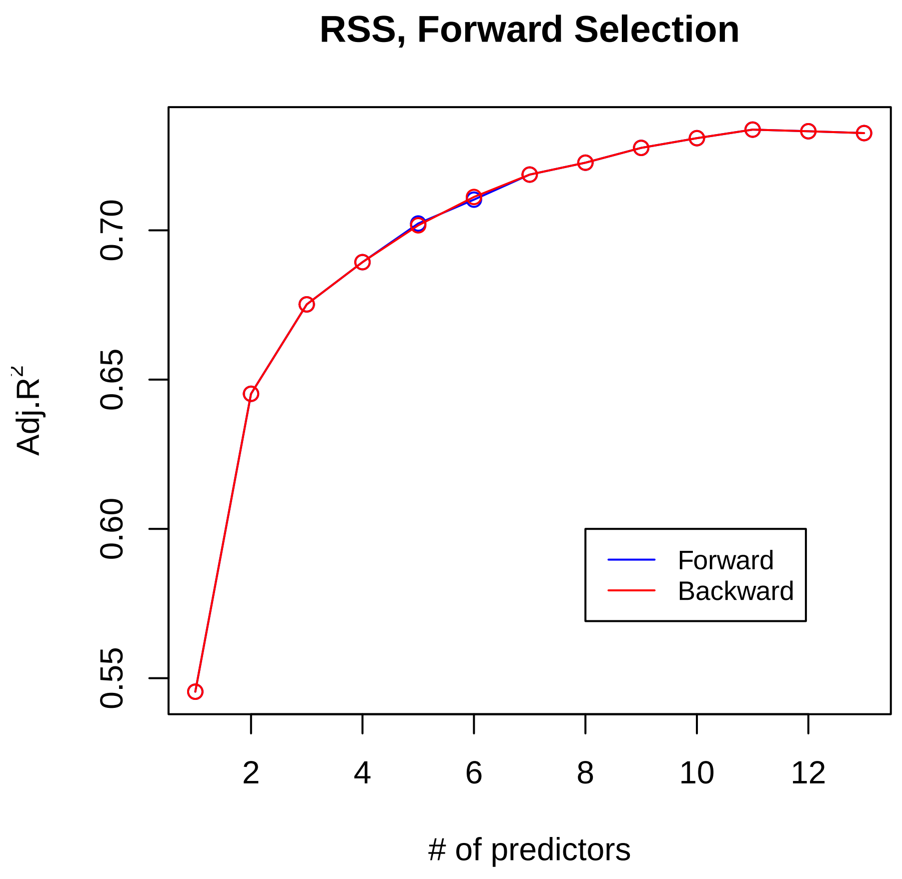
a)

```
library(leaps)
forward_selection = regsubsets(medv~., data=boston.train,
                               method="forward", nvmax = ncol(boston.train) - 1)
forward_summary = summary(forward_selection)
backward_selection = regsubsets(medv~., data=boston.train,
                                method="backward", nvmax = ncol(boston.train) - 1)
```

```r
backward_summary = summary(backward_selection)
forward_summary$adjr2
```

```
##  [1] 0.5454616 0.6452468 0.6751936 0.6893398 0.7022859 0.7102751 0.7186754
##  [8] 0.7226542 0.7276173 0.7308733 0.7337328 0.7331813 0.7325720
```

```r
library(latex2exp)
plot(c(1:(ncol(boston.train)-1)), forward_summary$adjr2,
     xlab = "# of predictors", ylab = TeX(r'($Adj. R^2$)'),
     col="blue", main = "RSS, Forward Selection")
lines(c(1:(ncol(boston.train)-1)), forward_summary$adjr2, col="blue")
points(c(1:(ncol(boston.train)-1)), backward_summary$adjr2,
       xlab = "# of predictors", ylab = TeX(r'($Adj. R^2$)'),
       col="red", main = "RSS, Backward Selection")
lines(c(1:(ncol(boston.train)-1)), backward_summary$adjr2, col="red")
legend(8, 0.6, legend=c("Forward", "Backward"),
       col=c("blue", "red"), lty=c(1,1), cex=0.8)
```



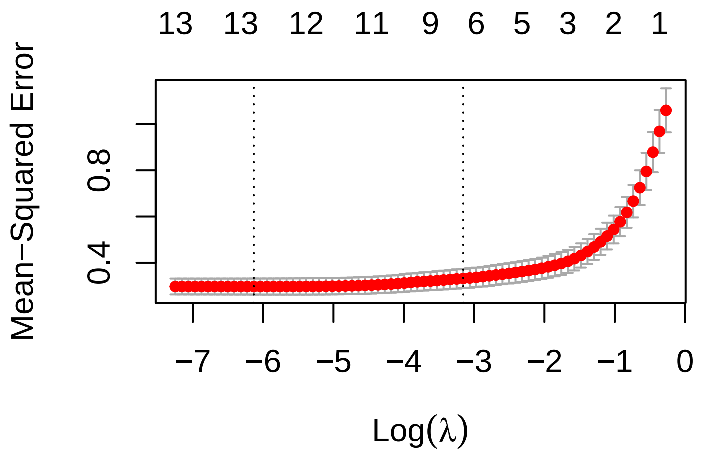
b)

```r
# According to F-selection, the 4 best predictors are lstat, rm, ptratio and dis
# We also must include the response, medv
best_pred = names(coef(forward_selection, 4))
best_pred = best_pred[2:5] # Remove the intercept
new_boston.train = boston.train[c(best_pred, "medv")]
```

c)

(i)

```
library(glmnet)
set.seed(1)
x_train <- model.matrix(medv~.,boston.train)[,-1]
y_train <- boston.train$medv
cv.out= cv.glmnet(x_train, y_train, alpha=1, nfolds = 5)
plot(cv.out)
```



(ii)

```
best_lambda_lasso <- cv.out$lambda.min
cat("Best lambda:", best_lambda_lasso)
```

```
## Best lambda: 0.003795687
```

(iii)

```
coef(cv.out, s = "lambda.min")
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.023602341
## crim        -0.076458877
## zn           0.088923654
## indus        .
## chas         0.087118503
## nox         -0.167421957
## rm           0.315045802
## age         -0.007962884
## dis         -0.304142295
## rad          0.247156468
## tax         -0.188382376
## ptratio     -0.201958534
## black        0.101646634
## lstat       -0.429076669
```

d)

1. True
2. False
3. False
4. True

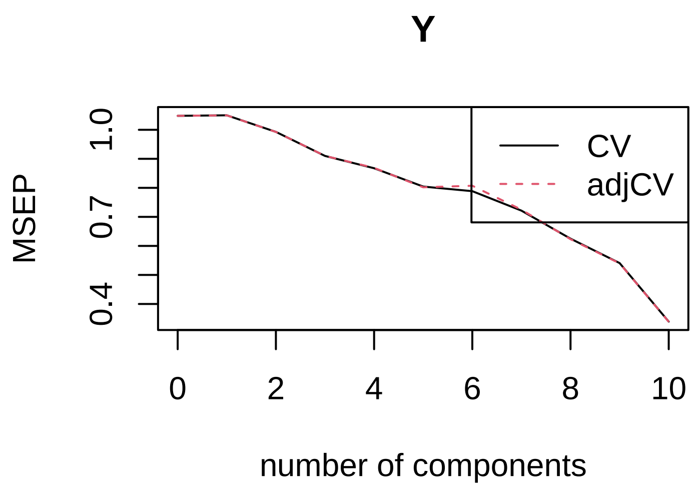
Problem 2

```
library(MASS)
set.seed(1)
# 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, ])
# show head(..)
# Y: response variable; X: predictor variable
head(synthetic)
```

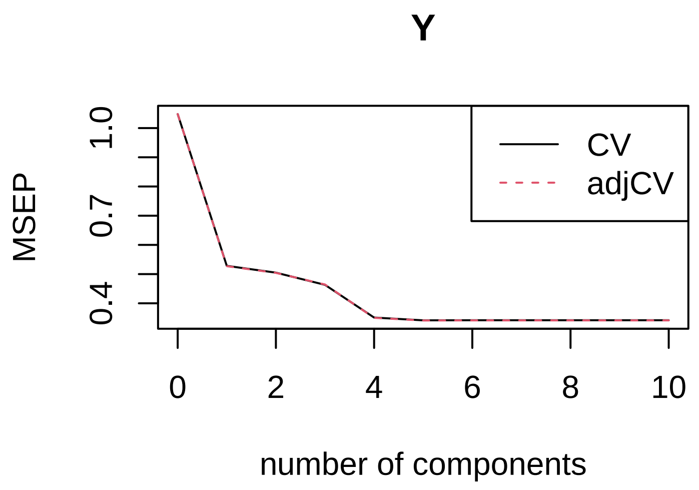
```
##             Y          X1          X2         X3         X4         X5
## 1 -1.43753239 -0.75905055 -0.69720326 -0.3016852 -0.7434697  0.8807558
## 2 -1.70972989 -0.28635632  0.04809182  0.5791725 -0.7446170  0.9935311
## 3  1.33931240  0.09574117 -0.89605758 -0.9636347  0.5554647 -0.5341800
## 4  0.20354906 -0.28702695  1.72952687  1.4289705 -0.1596993 -0.7161976
## 5 -0.09261896  0.02345825  0.51201583  0.1544345  0.4318039 -0.8674060
## 6  1.69952325  1.19231791 -0.98179754 -0.9567773 -0.6933918  0.4656891
##           X6         X7         X8          X9        X10
## 1 -0.8705750 -0.7448252 -0.4639697  0.62502272 -0.8149674
## 2  0.3532248 -0.5860332 -0.7964403  0.84868110 -0.1065119
## 3  0.4707434 -0.6588069 -0.7327518 -0.29429307  0.6588927
## 4 -0.7774007  0.2502145  0.5987052 -0.04428773  0.6247479
## 5 -0.9066908  0.8946086 -0.9700185  0.09082626  0.6102134
## 6 -0.7381794  0.8650175  0.4108119  0.75677429 -0.2281439
```

a)

```
set.seed(1)
library(pls)
pcr_fit = pcr(Y~., data=synthetic.train, scale=TRUE, validation="CV")
validationplot(pcr_fit, val.type = "MSEP", legendpos = "topright")
```



```
pls_fit = plsr(Y~., data=synthetic.train, scale=TRUE, validation="CV")
validationplot(pls_fit, val.type = "MSEP", legendpos = "topright")
```



b)

```
library(GGally)
cor_matrix = round(cor(synthetic), 4)
cor_matrix
```

```
##          Y      X1      X2      X3      X4      X5      X6      X7      X8
## Y   1.0000  0.7005  0.1130  0.0149  0.0195  0.0007 -0.0097 -0.0202 -0.0109
## X1  0.7005  1.0000 -0.0750 -0.0267 -0.0249 -0.0125  0.0204 -0.0341 -0.0058
## X2  0.1130 -0.0750  1.0000  0.9505  0.0197  0.0013 -0.0645  0.0475  0.0034
## X3  0.0149 -0.0267  0.9505  1.0000 -0.0117 -0.0092 -0.0637  0.0334 -0.0008
## X4  0.0195 -0.0249  0.0197 -0.0117  1.0000  0.0168 -0.0065 -0.0717 -0.0153
## X5  0.0007 -0.0125  0.0013 -0.0092  0.0168  1.0000  0.0264 -0.0242 -0.0391
## X6  -0.0097  0.0204 -0.0645 -0.0637 -0.0065  0.0264  1.0000  0.0088 -0.0481
## X7  -0.0202 -0.0341  0.0475  0.0334 -0.0717 -0.0242  0.0088  1.0000  0.0645
## X8  -0.0109 -0.0058  0.0034 -0.0008 -0.0153 -0.0391 -0.0481  0.0645  1.0000
## X9   0.0219  0.0479 -0.0682 -0.0531 -0.0027  0.0113 -0.0820  0.0007 -0.0322
## X10 -0.0147  0.0229  0.0160  0.0331  0.0022  0.0162  0.0123 -0.0320 -0.0269
##          X9     X10
## Y    0.0219 -0.0147
## X1   0.0479  0.0229
## X2  -0.0682  0.0160
## X3  -0.0531  0.0331
## X4  -0.0027  0.0022
## X5   0.0113  0.0162
## X6  -0.0820  0.0123
## X7   0.0007 -0.0320
## X8  -0.0322 -0.0269
## X9   1.0000  0.0644
## X10  0.0644  1.0000
```

```r
synthetic_princomp_1 = pcr_fit$loadings[,c('Comp 1')]
synthetic_plsr_princomp_1 = pls_fit$loadings[,c('Comp 1')]

print("Coefficients of principal component 1 (PCR): ")
```

```
## [1] "Coefficients of principal component 1 (PCR): "
```

```r
synthetic_princomp_1
```

```
##           X1           X2           X3           X4           X5           X6 
##  0.084523270 -0.699026279 -0.694896358 -0.028811471  0.003029199  0.105357979 
##           X7           X8           X9          X10 
## -0.065309349 -0.019813467  0.061482035 -0.030977784
```

```r
print("Coefficients of principal component 1 (PLSR): ")
```

```
## [1] "Coefficients of principal component 1 (PLSR): "
```

```r
synthetic_plsr_princomp_1
```

```
##          X1          X2          X3          X4          X5          X6 
##  0.96793668  0.15173876  0.20155683 -0.04877655 -0.04264084 -0.07245705 
##          X7          X8          X9         X10 
## -0.09145116 -0.03836298  0.11491663  0.05499069
```

Difference between PCR and PSLR

From the plots, we can see that PLSR performs substantially better than PCR. It captures most of the variability of the response using only 4 components. PCR, on the other hand, requires 10 components to match the result of PLSR, failing to reduce the dimensionality of the feature space. From the correlation matrix (provided above), we see that the covariates are highly uncorrelated, with the exception of X2 and X3 (rho = 0.95), and X1 and the response (rho = 0.70). This helps us explain why the PLSR is able to reduce the variance while also reducing the dimensionality. As this method considers the response when constructing a principal component, it uses the correlation between X1 and Y to capture most of the variability of the response in the first principal component. From the decomposition of the first principal component of PCA and PLSR, we do indeed see that the PLSR model has a large coefficient for X1, while the PCA has large coefficients for X2 and X3.

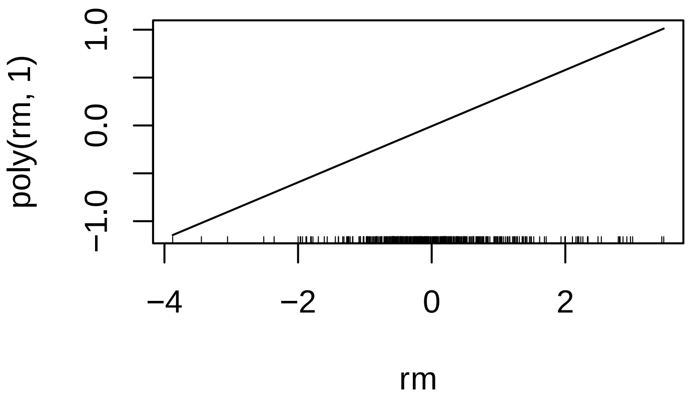
Problem 3

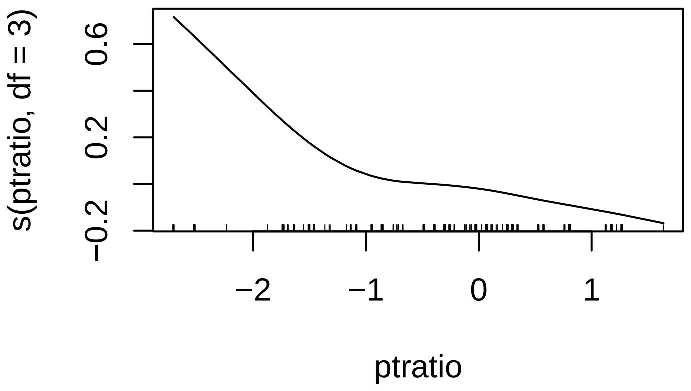
a)

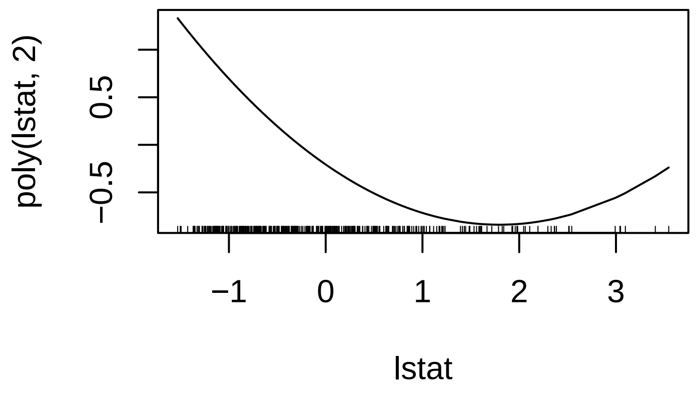
1. True
2. False
3. False
4. True

b)

```
library(gam)
gam_fit = gam(medv~poly(rm, 1)+s(ptratio, df = 3)+poly(lstat, 2), data = boston.train)
plot(gam_fit, pages = 1)
```







```
# summary(gam_fit)
```

Problem 4

a)

1. False
2. True
3. True
4. True

b)

See figure 1

c)

```
library(tidyverse)
library(palmerpenguins) # Contains the data set "penguins".
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()
# 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, ]
```

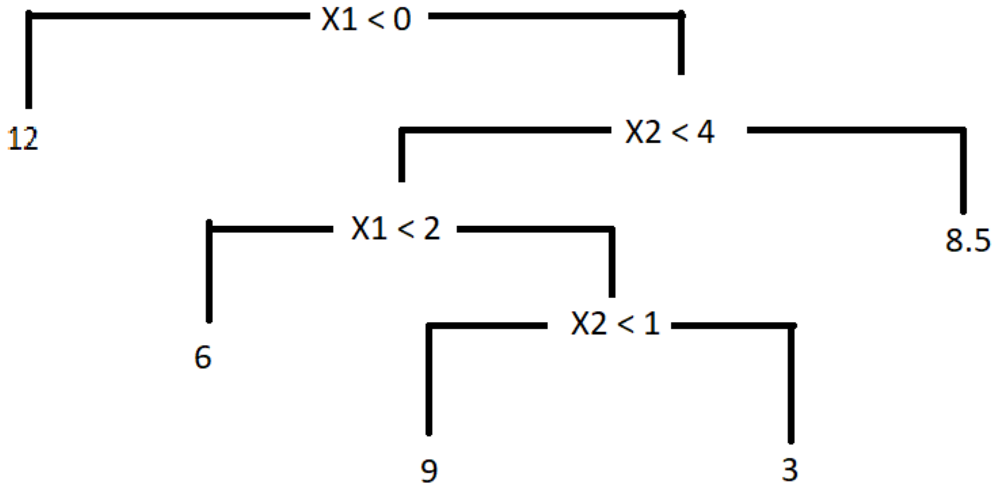
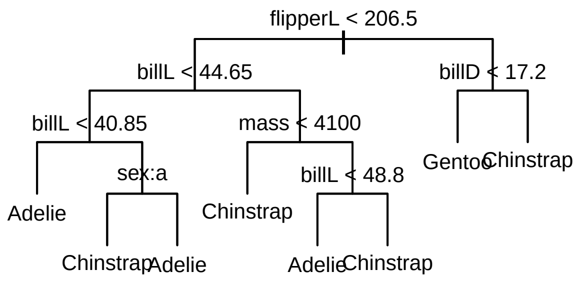


Figure 1: Tree from task 4b)

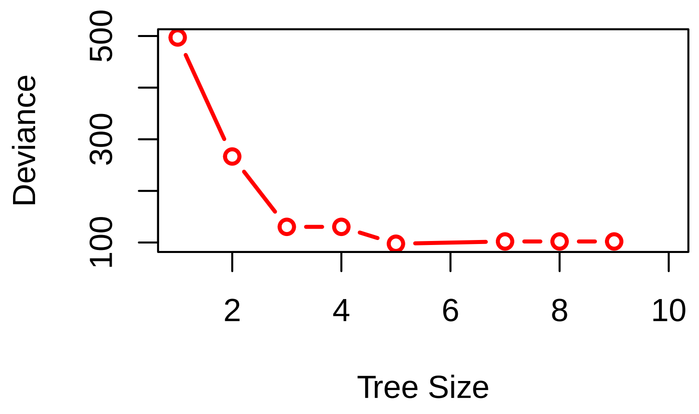
(i)

```r
set.seed(123)
library(tree)
penguin_simple_tree = tree(species~., data = train, method = "gini")
plot(penguin_simple_tree, type="uniform")
text(penguin_simple_tree, cex=0.7)
```



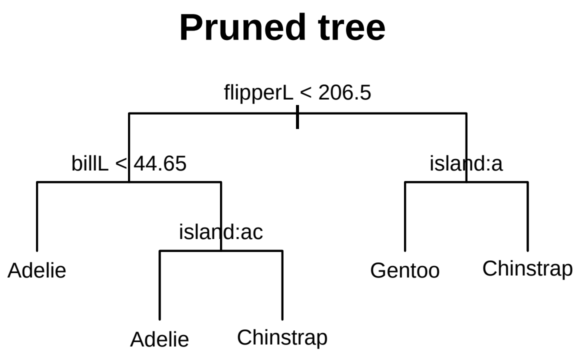
(ii)

```r
set.seed(123)
penguin_tree_huge = tree(species~., data = train,
                         control = tree.control(nrow(train),
                                   mincut = 2, minsize = 4, mindev = 0.001))
cv_tree_huge = cv.tree(penguin_tree_huge, K = 10)
plot(cv_tree_huge$dev ~  cv_tree_huge$size,type= "b",
     lwd=2, col="red", xlab="Tree Size", ylab="Deviance", xlim=c(1, 10))
```



(iii)

```r
set.seed(123)
min_dev = which.min(rev(cv_tree_huge$dev))
prune_penguins = prune.tree(penguin_tree_huge, best = min_dev)
plot(prune_penguins, type="uniform")
text(prune_penguins, cex=0.7)
title("Pruned tree")
```



```
# We use majority vote to predict species (removing type = "class" will get
# you the probabilities for each species for each observation)
simple_penguin_pred = predict(penguin_simple_tree, newdata = test, type = "class")
prune_penguin_pred = predict(prune_penguins, newdata = test, type = "class")

library(caret)
conf_mat_simple_penguin <- confusionMatrix(simple_penguin_pred,
                                           reference = test$species)$table
conf_mat_simple_penguin
```

```
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        41         2      1
##   Chinstrap      1        18      0
##   Gentoo         0         0     37
```

```
conf_mat_prune_penguin <- confusionMatrix(prune_penguin_pred,
                                          reference = test$species)$table
conf_mat_prune_penguin
```

```
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        42         4      0
##   Chinstrap      0        16      0
##   Gentoo         0         0     38
```

```
cat("Misclassification rate (simple tree):",
    (1 - sum(diag(conf_mat_simple_penguin))/sum(conf_mat_simple_penguin[1:3, 1:3]))*100, "%\n")
```

```
## Misclassification rate (simple tree): 4 %
```

```
cat("Misclassification rate (pruned tree):",
    (1 - sum(diag(conf_mat_prune_penguin))/sum(conf_mat_prune_penguin[1:3, 1:3]))*100, "%\n")
```

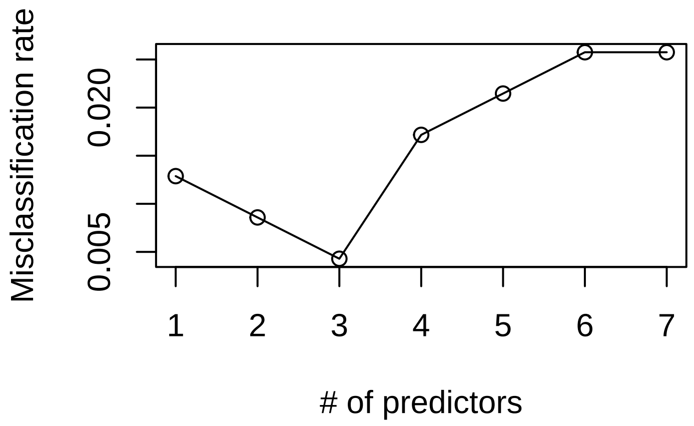
```
## Misclassification rate (pruned tree): 4 %
```

d)

```
library(randomForest)
set.seed(4268)
misclass_rate = c()

for(i in 1:(ncol(train))) {
  rf_penguin = randomForest(species~., data = train, mtry = i,
                            ntree = 500, importance = TRUE)
  rf_predict = predict(rf_penguin, type = "class")
  conf_penguin <- confusionMatrix(rf_predict, reference = train$species)$table
  misclass_rate[i] =  1 - sum(diag(conf_penguin))/sum(conf_penguin[1:3, 1:3])
}

plot(c(1:(ncol(train))), misclass_rate, type = "o", xlab = "# of predictors",
     ylab = "Misclassification rate")
```



```
best_mtry = which.min(misclass_rate)
cat("Best value for mtry:", best_mtry)
```

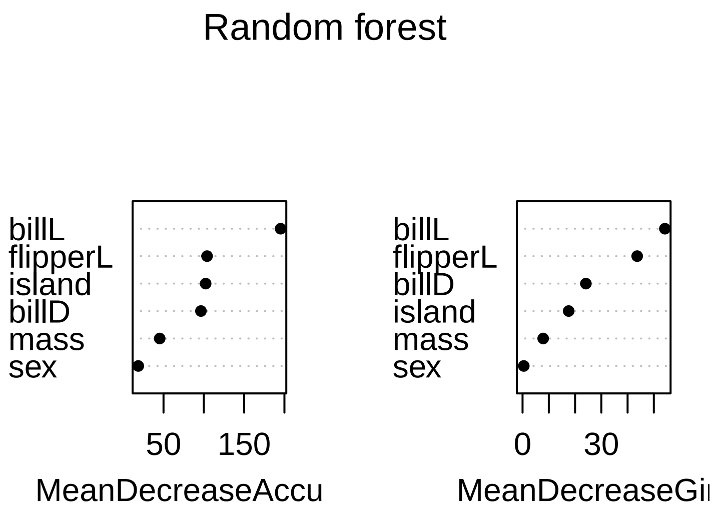
```
## Best value for mtry: 3
```

```
best_rf_penguin = randomForest(species~., data = train, mtry = best_mtry,
                               ntree = 5000, importance = TRUE)
best_rf_predict = predict(rf_penguin, newdata = test, type = "class")
conf_test_penguin <- confusionMatrix(best_rf_predict, reference = test$species)$table
best_misclass_rate = 1 - sum(diag(conf_test_penguin))/sum(conf_test_penguin[1:3, 1:3])

cat("Misclassification error (mtry=3):", best_misclass_rate)
```

```
## Misclassification error (mtry=3): 0.02
```

```
varImpPlot(best_rf_penguin, pch=20, main = "Random forest")
```



We use the out-of-bag (OOB) error rate to select the best value for mtry, the number of unique predictors considered at each split in a decision tree. When looping over the different values for mtry, we construct 500 trees. When constructing the final forest, we increase the number of trees to 5000 as this calculation only needs to be performed once. We found that the best value for m was 3, which is not surprising as $\sqrt{p} = \sqrt{7} \approx 3$. From the variable-importance plots we can see that billL (bill length) and flipperL (flipper length) are the two most influential variables.

Problem 5

a)

1. False
2. True
3. True
4. True

b)

(i)

```
library(e1071)
set.seed(4268)
cv_svm_linear = tune(svm, species ~ ., data=train, kernel="linear",
                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100)))

cv_svm_radial = tune(svm, species ~ ., data=train, kernel="radial",
                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100),
                                   gamma = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100)))

cv_svm_linear$best.parameters
```

```
##   cost
## 5    1
```

```
cv_svm_radial$best.parameters
```

```
##   cost gamma
## 9  100  0.01
```

```
cat("Error rate training (linear):", cv_svm_linear$best.performance, "\n")
```

```
## Error rate training (linear): 0
```

```
cat("Error rate training (radial):", cv_svm_radial$best.performance, "\n")
```

```
## Error rate training (radial): 0
```

(ii)

```
library(caret)
# We get the models with the optimal paramaters using the best.model attribute
best_linear = cv_svm_linear$best.model
best_radial = cv_svm_radial$best.model

best_linear_svm_predict = predict(best_linear, newdata = test, type = "class")
conf_test_penguin_linear <- confusionMatrix(best_linear_svm_predict, reference = test$species)$table
best_misclass_rate_lin=1-sum(diag(conf_test_penguin_linear))/sum(conf_test_penguin_linear[1:3,1:3])

best_radial_svm_predict = predict(best_radial, newdata = test, type = "class")
```

```r
conf_test_penguin_radial <- confusionMatrix(best_radial_svm_predict, reference = test$species)$table
best_misclass_rate_rad=1-sum(diag(conf_test_penguin_radial))/sum(conf_test_penguin_radial[1:3,1:3])

print("Confusion matrix (linear):")
```

```
## [1] "Confusion matrix (linear):"
```

```r
conf_test_penguin_linear
```

```
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        42         0      0
##   Chinstrap      0        20      0
##   Gentoo         0         0     38
```

```r
print("Confusion matrix (radial):")
```

```
## [1] "Confusion matrix (radial):"
```

```r
conf_test_penguin_radial
```

```
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        42         0      0
##   Chinstrap      0        20      0
##   Gentoo         0         0     38
```

```r
cat("Misclassification rate (linear):", best_misclass_rate_lin, "\n")
```

```
## Misclassification rate (linear): 0
```

```r
cat("Misclassification rate (radial):", best_misclass_rate_rad, "\n")
```

```
## Misclassification rate (radial): 0
```

(iii) Both models performed extremely well, with a test error rate of 0. Because we are unable to distinguish the models based on performance, we select the most interpretable model, the SVM with a linear kernel. This model is in essence a p-dimensional hyperplane, and while it may not be visualized easily, the coefficients of the hyperplane can be interpreted.

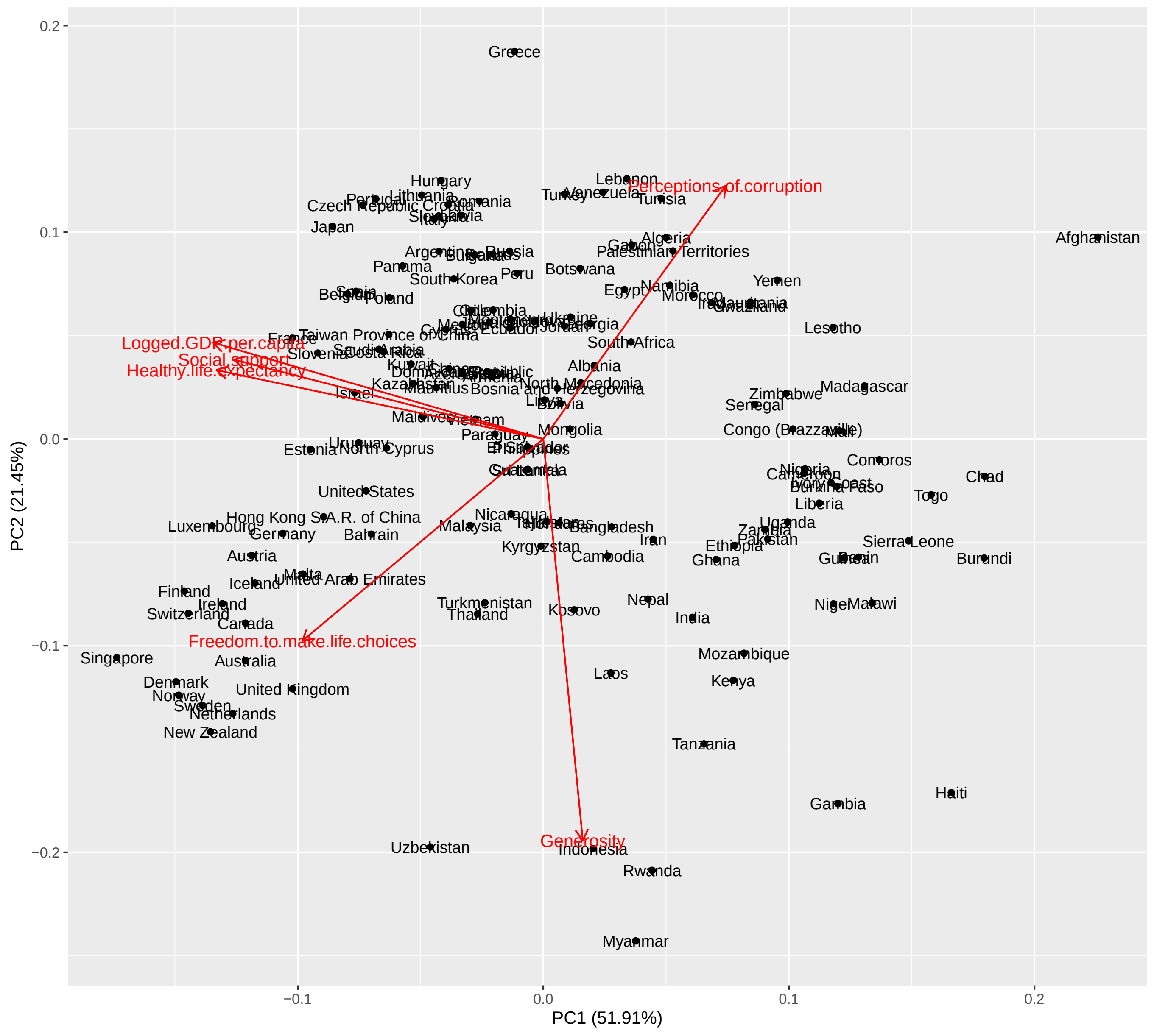
```r
# load a synthetic dataset
id <- "1NJ1SuUBebl5P8rMSIwm_n3S8a7K43yP4" # google file ID
happiness <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id)
                      ,fileEncoding="UTF-8-BOM")
colnames(happiness)
```

```
##  [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 create 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 ...
```

```
library(ggfortify)
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 = 4,
         label=T, label.size=3.5)
```



a)

(i)

From the plot we see that GDP per capita, social support and and healthy life expcentacy are highly correlated. We can also observe that freedom to make life choices and perception of corruption are negatively correlated (an increase in one, would lead to a decrease of the other).

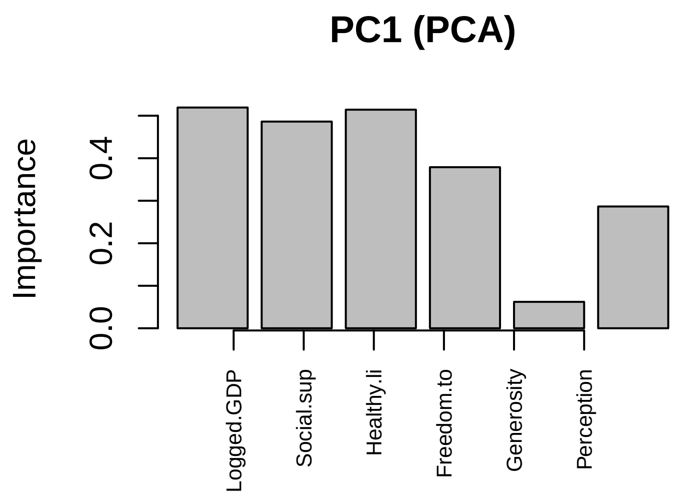
(ii)

Afghanistan is the outlier, as it is far from any other country.

b)

(i)

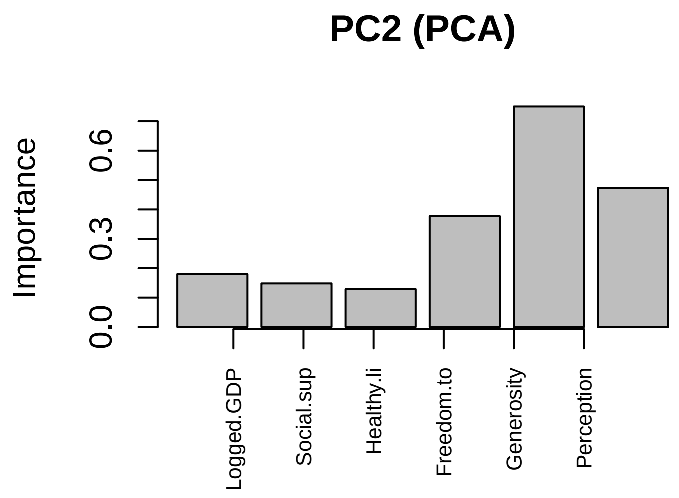
```r
set.seed(123)
princomp_1 = data.frame(pca_mat$rotation)$PC1
df = data.frame(pca_mat$rotation[, 1])
names = rownames(df)
labs <- substr(sapply(strsplit(names, " "),
                      function(x) x[1]), 1, 10)
barplot(abs(princomp_1), xaxt = "n", xlab = "", ylab = "Importance", main = "PC1 (PCA)")
axis(1, at = 1:6, labels = labs, las = 2, cex.axis = 0.7)
```



```r
princomp_1
```

```
## [1] -0.51930732 -0.48621110 -0.51421885 -0.37887095  0.06213545  0.28651988
```

```r
set.seed(123)
princomp_2 = data.frame(pca_mat$rotation)$PC2
barplot(abs(princomp_2), xaxt = "n", xlab = "", ylab = "Importance", main = "PC2 (PCA)")
axis(1, at = 1:6, labels = labs, las = 2, cex.axis = 0.7)
```



```
princomp_2
```

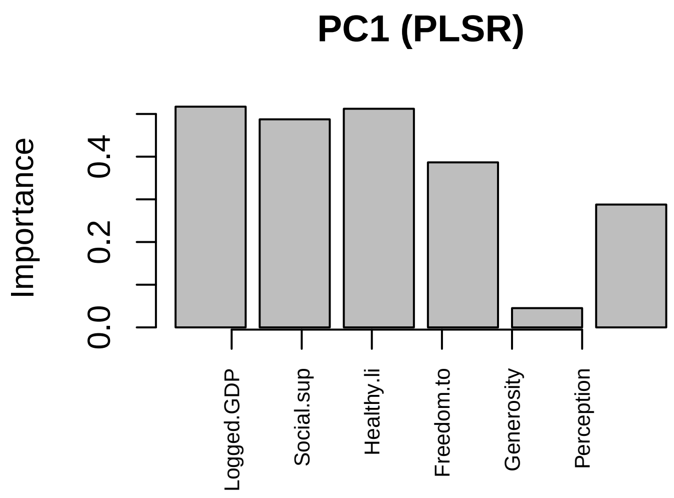
```
## [1]  0.1800485  0.1482583  0.1287626 -0.3770901 -0.7503368  0.4730962
```

(ii)

```
set.seed(123)
library(pls)
plsr_model = plsr(Ladder.score~., data=happiness.XY, scale=TRUE, validation="CV")
```

(iii)

```
library(pls)
plsr_princomp_1 = plsr_model$loadings[,c('Comp 1')]
barplot(abs(plsr_princomp_1), xaxt = "n", xlab = "", ylab = "Importance", main = "PC1 (PLSR)")
axis(1, at = 1:6, labels = labs, las = 2, cex.axis = 0.7)
```



```
plsr_princomp_1
```

```
##       Logged.GDP.per.capita               Social.support
##                  0.51710666                   0.48740123
##     Healthy.life.expectancy Freedom.to.make.life.choices
##                  0.51212197                   0.38659271
##                  Generosity    Perceptions.of.corruption
##                 -0.04516483                  -0.28779558
```

When plotting the principal component of the PCA model and the PLSR model, the bar graphs were quite similar. Although the signage of each coefficient was flipped, this doesn't matter as the variance captured by each principal component in both models is invariant up to sign.

(iv)

From the graph we see that GDP per capita, social support and healthy life expectancy are the three most important predictors to predict happiness in the PLSR model.

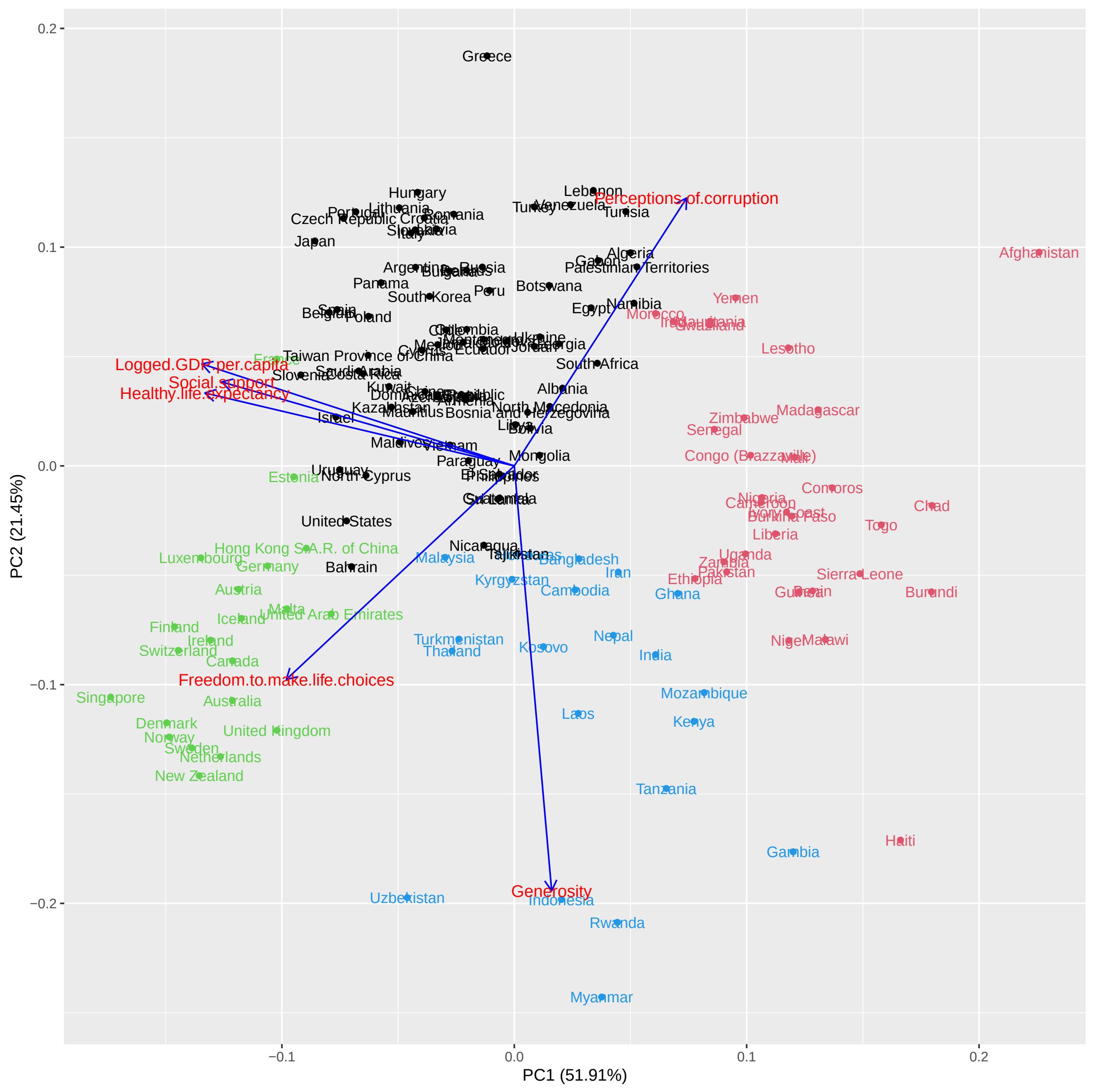
c)

1. False
2. False

3. True
4. True

d)

```
K = 4
km.out = kmeans(happiness.X, K)
autoplot(pca_mat, data = happiness.X, colour=km.out$cluster,
         label=T, label.size=3.5,
         loadings = T, loadings.colour = 'blue',
         loadings.label = T, loadings.label.size = 4)
```



We assume that high GDP, social support, healthy life expectancy, freedom of choice and generosity contribute to a higher happiness, while corruption would lead to a decrease in happiness. From PC1 we see that GDP, social support, life expectancy, freedom of choice and corruption all have a large impact on the principal component. Furthermore, we see that the coefficient of the first 4 are all negative, while corruption has a positive coefficient. This suggests to us that a negative PC1 is correlated with a high happiness score.

Looking at principal component 2, we see that three most influential features are freedom of choice, generosity and perceived corruption, with generosity and freedom having negative coefficients, and corruption having a positive coefficient. This suggests that a negative value for PC2 indicates increased happiness, assuming our prior assumption about which features contribute to happiness holds.

From the PLSR model (which is fitted with regards to the response) we see however that generosity does not seem important when determining the happiness of a country. This might indicate that the importance of principle component 2 is slightly lower than that of principal component 1. Other than our assumption about generosity, the PLSR model also confirms our assumption about which features increase happiness, and which don't.

- The black cluster has a low value for both principal components, suggesting a very high happiness score.
- The red cluster generally has a low value for PC1, with a mid to high values for PC2. Since we believe that PC1 is more important with regards to happiness than PC2, this cluster seems to contain countries with a mid to high happiness score.
- The blue cluster has average values for PC1 and high values for PC2. Again, since we regard PC1 as more important when determining happiness, these countries seem to have a mid happiness score.
- The green cluster has high values for PC1 and average values for PC2. Again, the importance of PC1 outweighs PC2, and suggests that these countries have the lowest happiness score.