TMA4268 Compulsory Exercises 2

Group 25

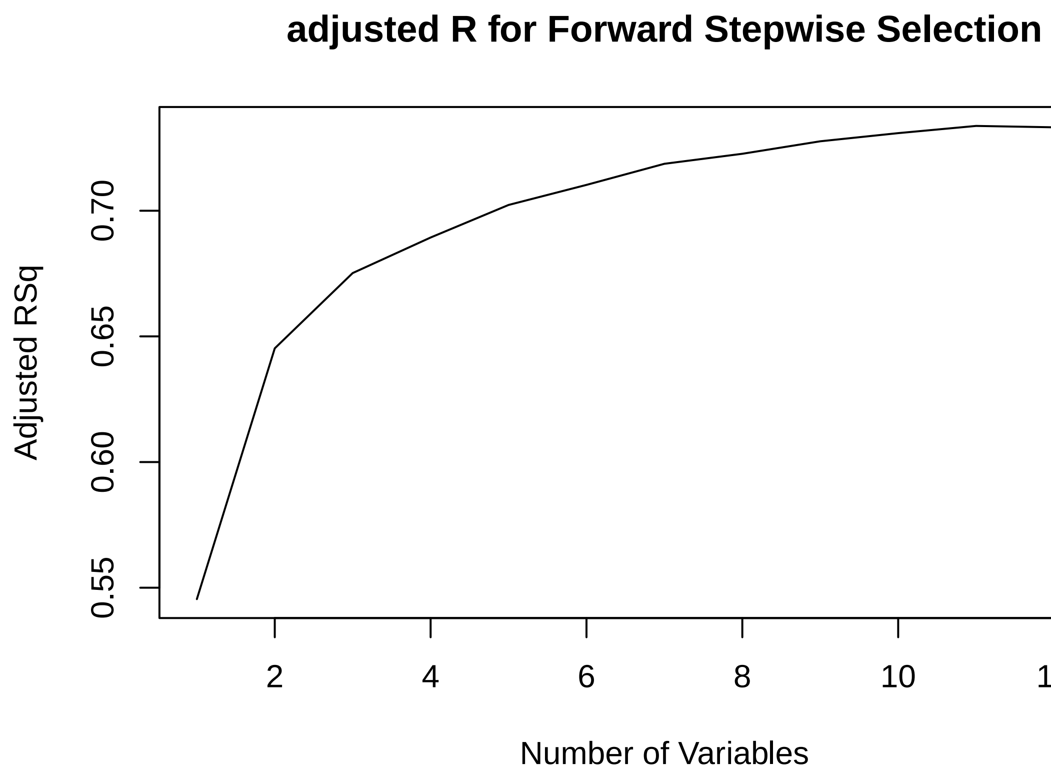
1 4 2022

Problem 1

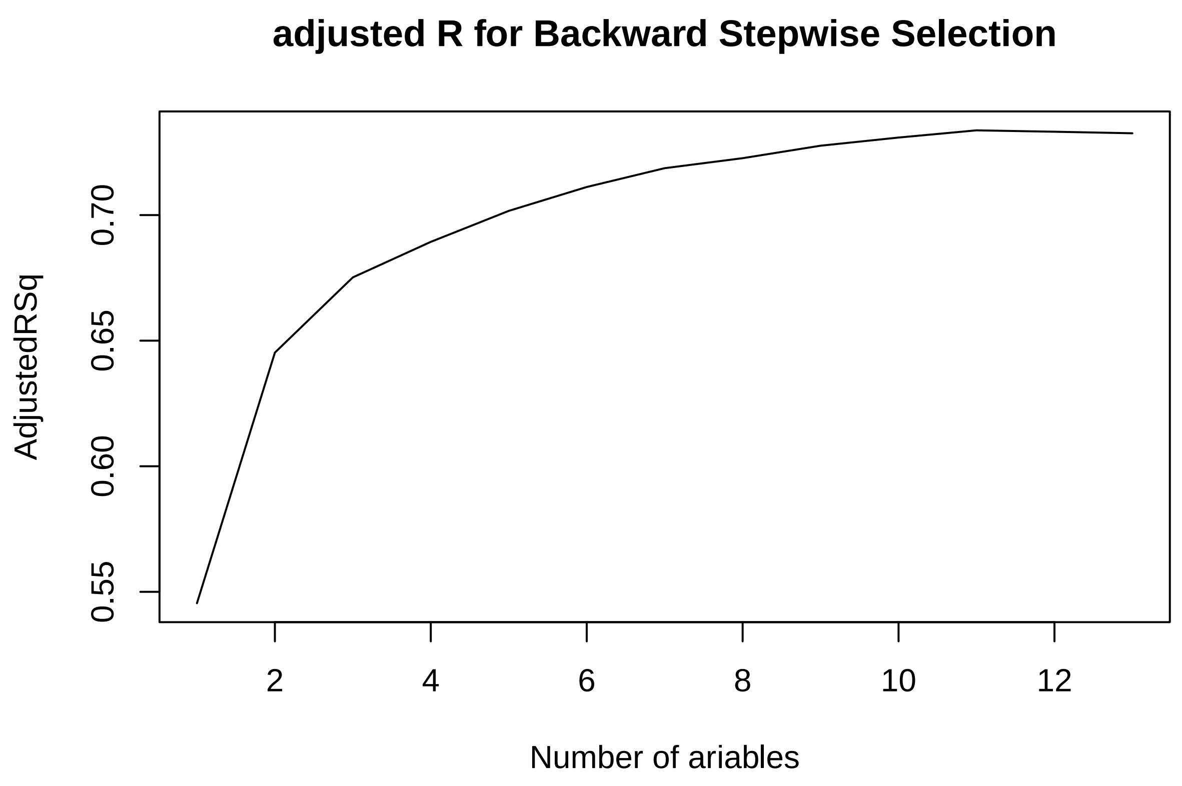
```r
set.seed(1)
boston <- scale(Boston, center = T, scale = T)
# split into training and test sets
train.ind = sample(1:nrow(boston), 0.8 * nrow(boston))
boston.train = data.frame(boston[train.ind, ])
boston.test = data.frame(boston[-train.ind, ])
```

a)

```r
regfit.fwd = regsubsets(medv ~. , data=boston.train,nvmax=14, method ="forward")
regfit.bwd = regsubsets(medv ~. , data=boston.train,nvmax=14, method ="backward")
reg.summary.fwd <- summary(regfit.fwd)
plot(reg.summary.fwd$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
title(main= "adjusted R for Forward Stepwise Selection")
```



```
reg.summary.bwd <- summary(regfit.bwd)
plot(reg.summary.bwd$adjr2,xlab="Number of ariables",ylab="AdjustedRSq",type="l")
title(main= "adjusted R for Backward Stepwise Selection")
```



b)

```
reg.summary.fwd$outmat
```

```
##           crim zn  indus chas nox rm  age dis rad tax ptratio black lstat
## 1  ( 1 )  " "  " " " "   " "  " " " " " " " " " " " " " "     " "   "*"
## 2  ( 1 )  " "  " " " "   " "  " " "*" " " " " " " " " " "     " "   "*"
## 3  ( 1 )  " "  " " " "   " "  " " "*" " " " " " " " " "*"     " "   "*"
## 4  ( 1 )  " "  " " " "   " "  " " "*" " " "*" " " " " "*"     " "   "*"
## 5  ( 1 )  " "  " " " "   " "  " " "*" " " "*" " " " " "*"     "*"   "*"
## 6  ( 1 )  " "  " " " "   " "  "*" "*" " " "*" " " " " "*"     "*"   "*"
## 7  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" " " " " "*"     "*"   "*"
## 8  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" "*" " " "*"     "*"   "*"
## 9  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"
## 10  ( 1 ) " "  "*" " "   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"
## 11  ( 1 ) "*"  "*" " "   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"
## 12  ( 1 ) "*"  "*" "*"   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"
## 13  ( 1 ) "*"  "*" "*"   "*"  "*" "*" "*" "*" "*" "*" "*"     "*"   "*"
```

We can from this summary see that we should choose the four predictors: rm, dis, ptratio and lstat.

```r
reg.summary.bwd$outmat
```

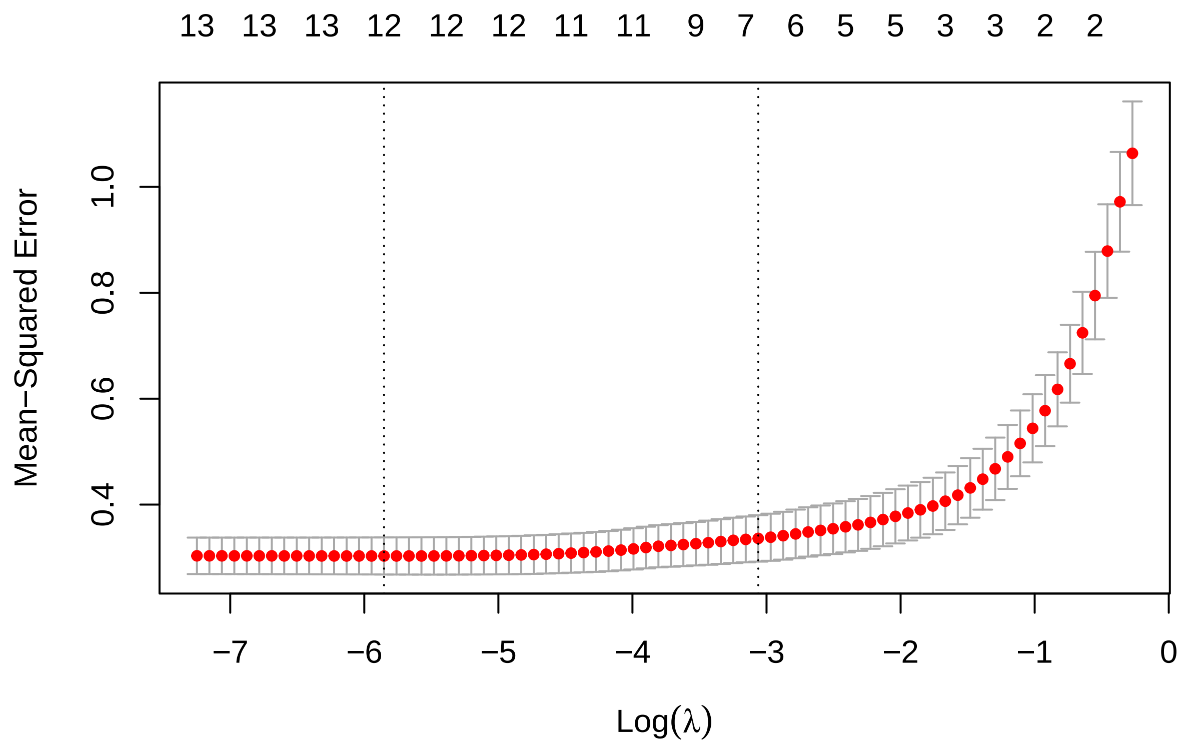
```
##            crim zn  indus chas nox rm  age dis rad tax ptratio black lstat
## 1  ( 1 )  " "  " " " "   " "  " " " " " " " " " " " " " "     " "   "*"  
## 2  ( 1 )  " "  " " " "   " "  " " "*" " " " " " " " " " "     " "   "*"  
## 3  ( 1 )  " "  " " " "   " "  " " "*" " " " " " " " " "*"     " "   "*"  
## 4  ( 1 )  " "  " " " "   " "  " " "*" " " "*" " " " " "*"     " "   "*"  
## 5  ( 1 )  " "  " " " "   " "  "*" "*" " " "*" " " " " "*"     " "   "*"  
## 6  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" " " " " "*"     " "   "*"  
## 7  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" " " " " "*"     "*"   "*"  
## 8  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" "*" " " "*"     "*"   "*"  
## 9  ( 1 )  " "  " " " "   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"  
## 10  ( 1 ) " "  "*" " "   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"  
## 11  ( 1 ) "*"  "*" " "   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"  
## 12  ( 1 ) "*"  "*" "*"   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"  
## 13  ( 1 ) "*"  "*" "*"   "*"  "*" "*" "*" "*" "*" "*" "*"     "*"   "*"  
```

We get the same result for the summary from the backward stepwise selection.

c)

- i)

```r
x.train <- model.matrix(medv ~ ., data = boston.train)
y.train <- boston.train$medv
set.seed(1)
cv.lasso <- cv.glmnet(x.train, y.train, alpha = 1)
plot(cv.lasso)
```



```
cat("he minimal lambda value is:",cv.lasso$lambda.min)
```

```
## he minimal lambda value is: 0.002871298
```

- ii)

```
cat("The best lambda value is:",cv.lasso$lambda.1se)
```

```
## The best lambda value is: 0.046795
```

- iii)

```
medv.lasso <- glmnet(x.train, y.train, alpha = 1, lambda = cv.lasso$lambda.1se)
coef(medv.lasso)
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)  0.02479037
## (Intercept)  .
## crim         .
## zn           .
## indus        .
## chas         0.06814148
## nox           .
## rm            0.34970663
## age           .
## dis          -0.04924090
## rad           .
## tax           .
## ptratio      -0.16212612
## black         0.07797455
## lstat        -0.42005208
```

d)

For this problem we get the following:

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

Problem 2

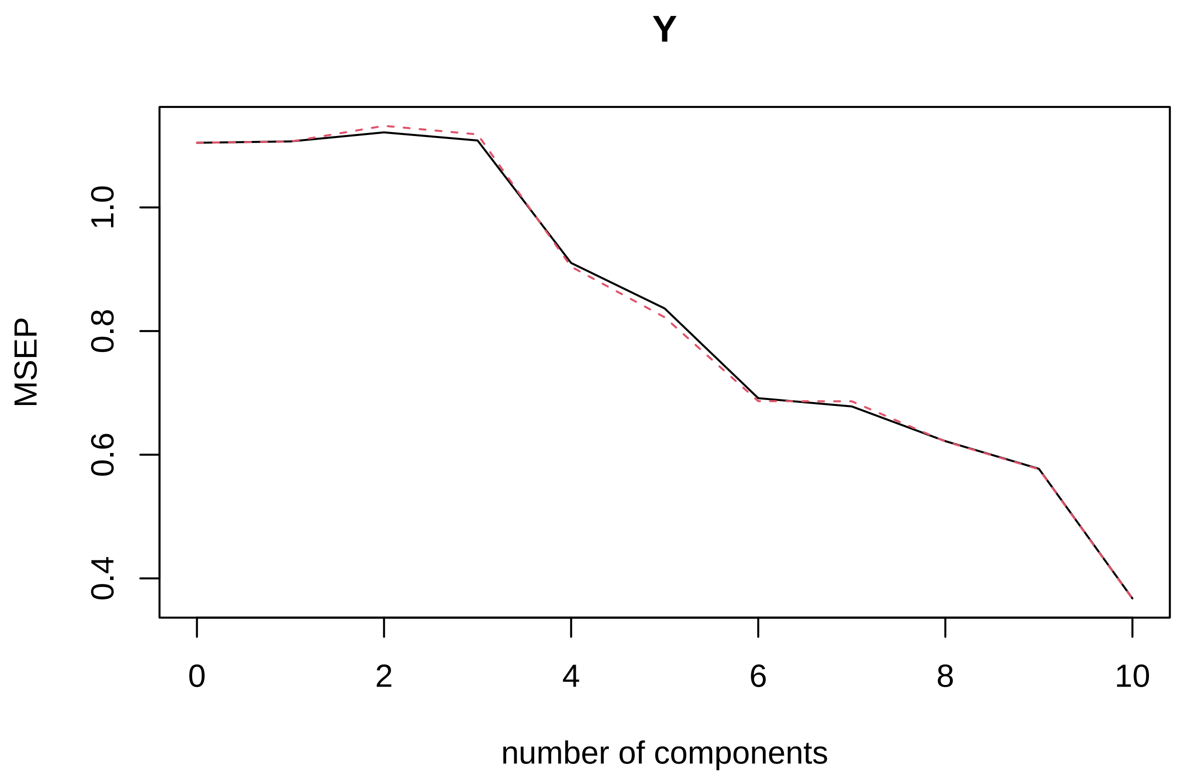
a)

First we perform pcr

```
set.seed(1)
pcr.fit<-pcr(Y~., data = synthetic.train, scale = TRUE, validation ="CV")
pcr.pred<-predict(pcr.fit,synthetic.test,ncomp = 5)
cat("The MSE relative to the test set is:", mean((pcr.pred- synthetic.test$Y)^2))
```

```
## The MSE relative to the test set is: 0.652026
```

```
validationplot(pcr.fit,val.type = "MSEP")
```

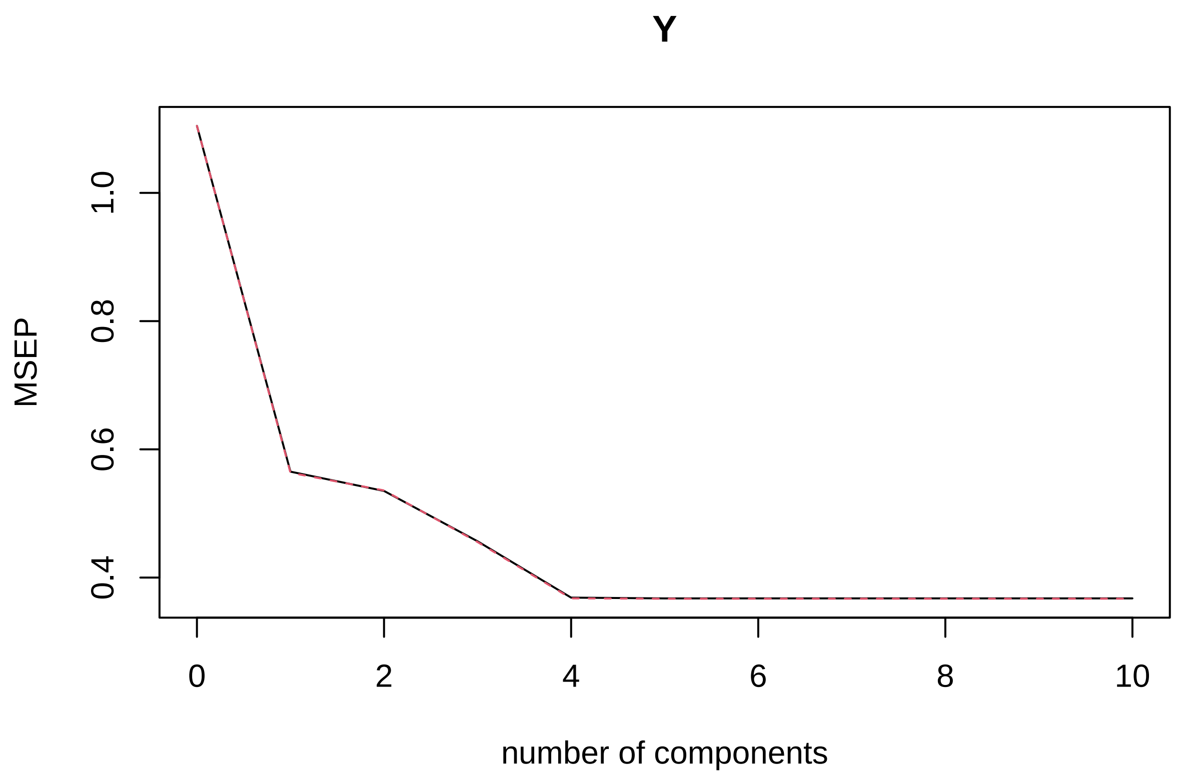


As we observe in the plot, the first few principle components has little effect on the MSEP. This is a good indicator that pcr is not effective when used on this particular data set. pcr usually preforms well when the first components contributes significantly to the reduction of MSEP Then perform plsr

```r
set.seed(1)
plsr.fit<-plsr(Y~.,data = synthetic.train,scale = TRUE, validation = "CV")
plsr.pred<-predict(plsr.fit,synthetic.test, ncomp = 5)
cat("The MSE relative to the test set is:", mean((plsr.pred-synthetic.test$Y)^2))
```

```
## The MSE relative to the test set is: 0.261249
```

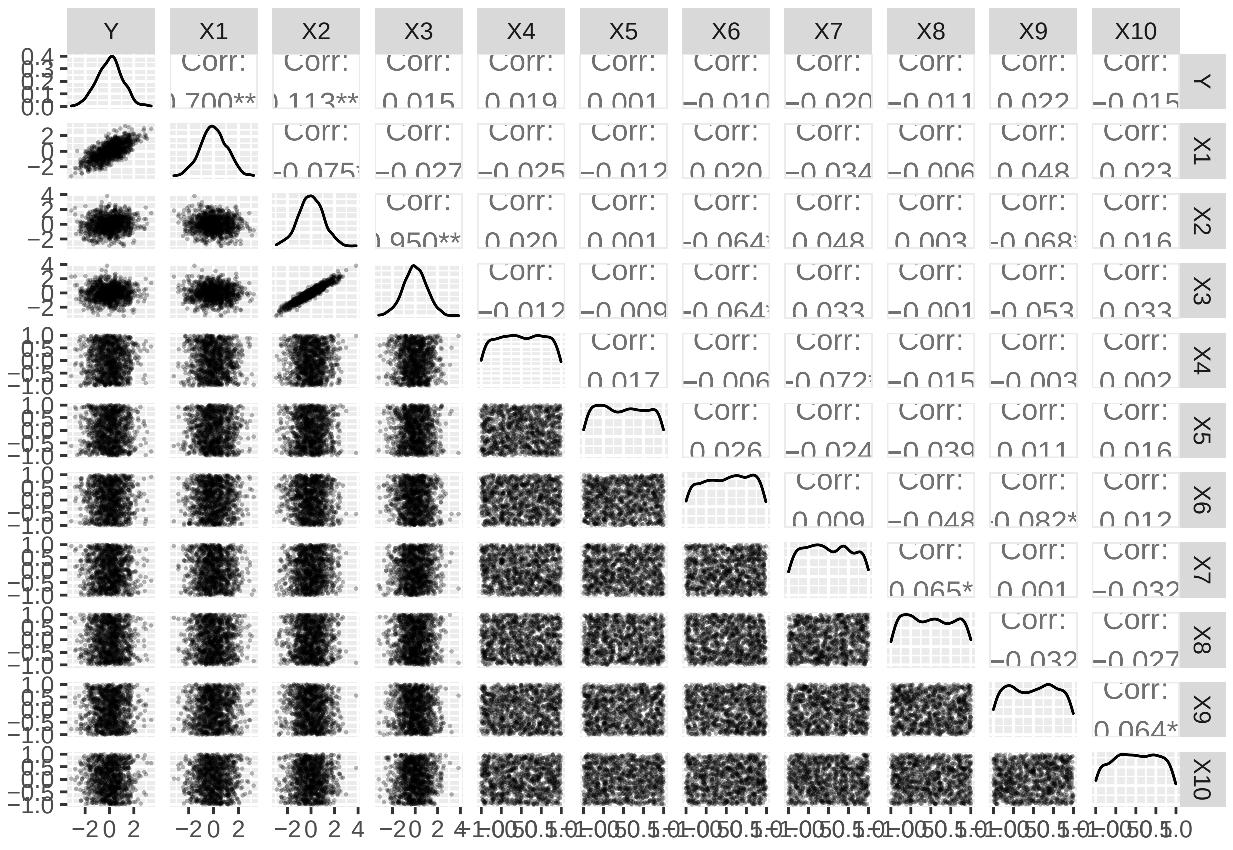
```r
validationplot(plsr.fit,val.type = "MSEP")
```



b)

We see from the plot that the first partial least squre component reduces the majority of the mean square error. After only 4 components, next to all the MSE is accounted for. To understand why plsr performce so much better than pcr, we need to understand the difference between pcr and plsr. Pcr is unsupervised, it simply finds the directions relative to the covariates for which the variance is the highest. On the other hand, plsr is supervised. The first components is found by fitting a linear regression to the data set. Because of this, plsr chooses the most predictive component in the data set. Plsr outperformce pcr significantly when directions along which there is low variance is highly predictive relative to the response. Pcr, mistakenly, drops these components, while plsr first few components is based on these directions.

```
ggpairs(synthetic, lower = list(continuous = wrap("points", alpha = 0.3, size=0.1), combo = wrap("dot",
```



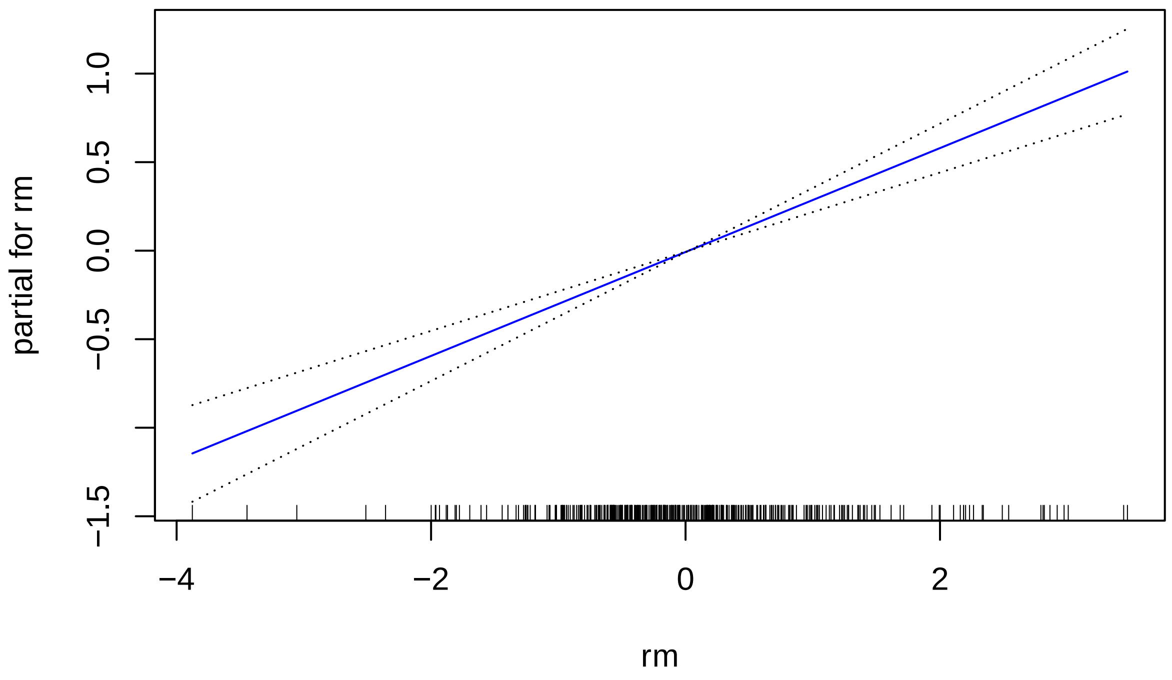
From this plot we se that y, x1, x2 and x3 is approximately normal distributed and that x4-10 is approximately uniform distributed. Moreover, the x1 seems to be the only covariate that is strongly correlated with y, so plsr would probably make a linear combination of mainly x1. Pcr would not pick up on this fact, but would rather make linear combinations of mainly x4-10, since these has higher variance compared to the normal distributed covariates.

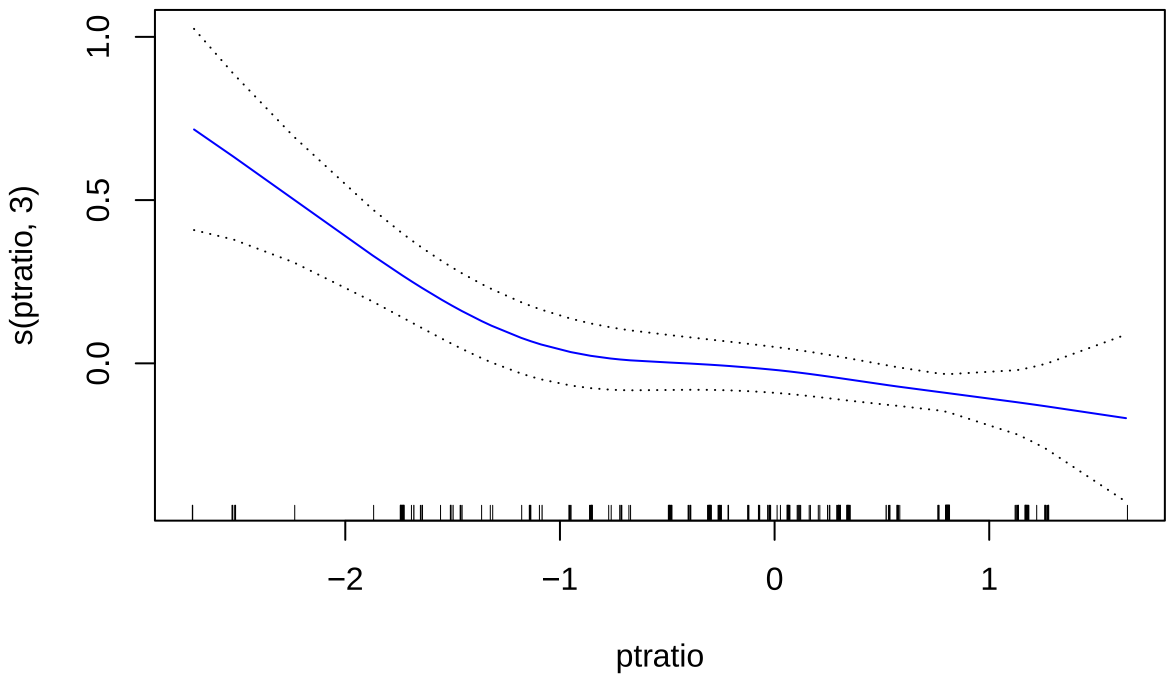
Problem 3

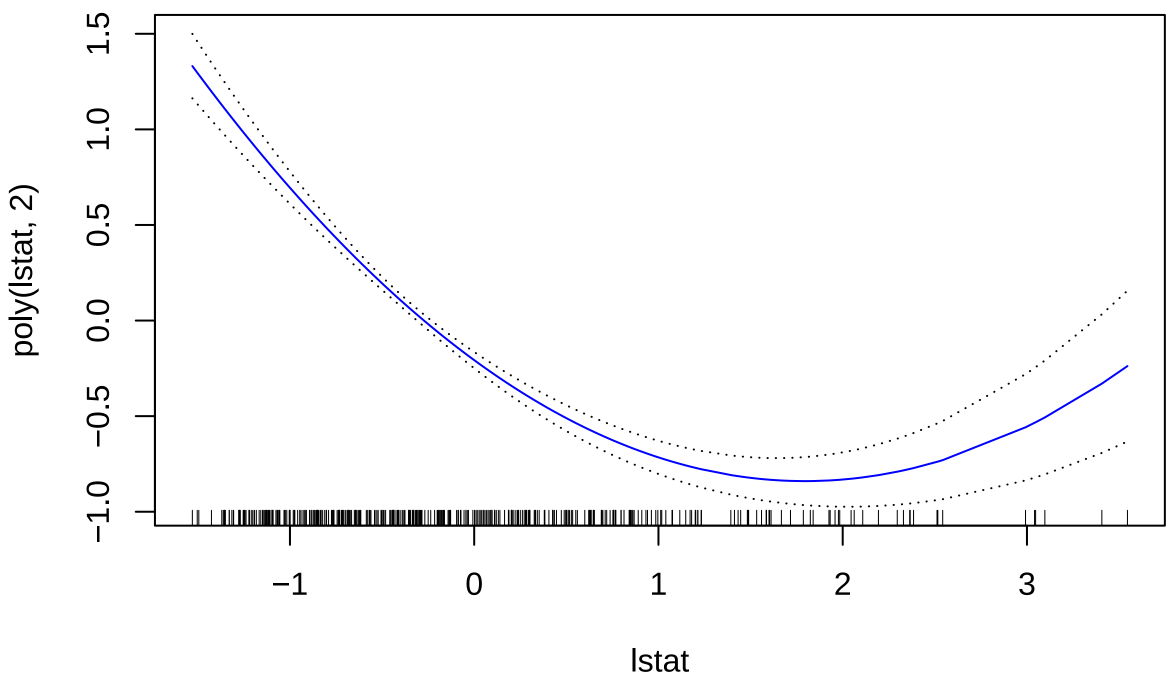
Multiple choice:

- i) True
- ii) False
- iii) False, it's twice differentiated.
- iv) True. Less variance (closer to a constant fuction), but higher bias.

```
fit.gam<-gam(medv~rm+s(ptratio,3)+poly(lstat,2), data = boston.train)
plot.Gam(fit.gam, se = TRUE, col = "blue")
```


partial for rm
1.0
0.5
0.0
−0.5
−1.5
−4
−2
0
2
rm


s(ptratio, 3)
1.0
0.5
0.0
−2
−1
0
1
ptratio



Problem 4

a)

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

b)

c)

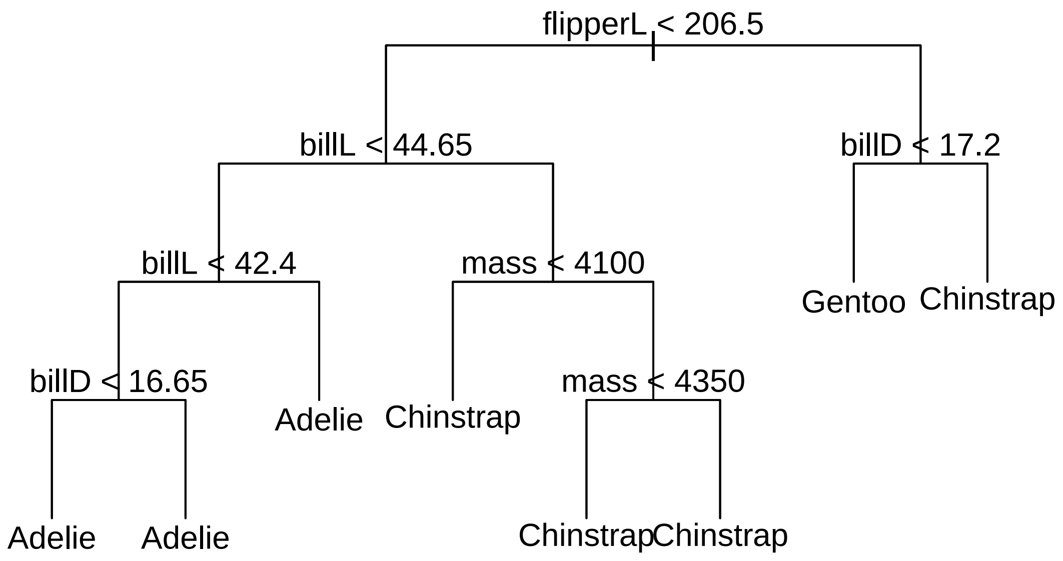
- i) We create the tree and use the train data.

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

```
##
## Classification tree:
```

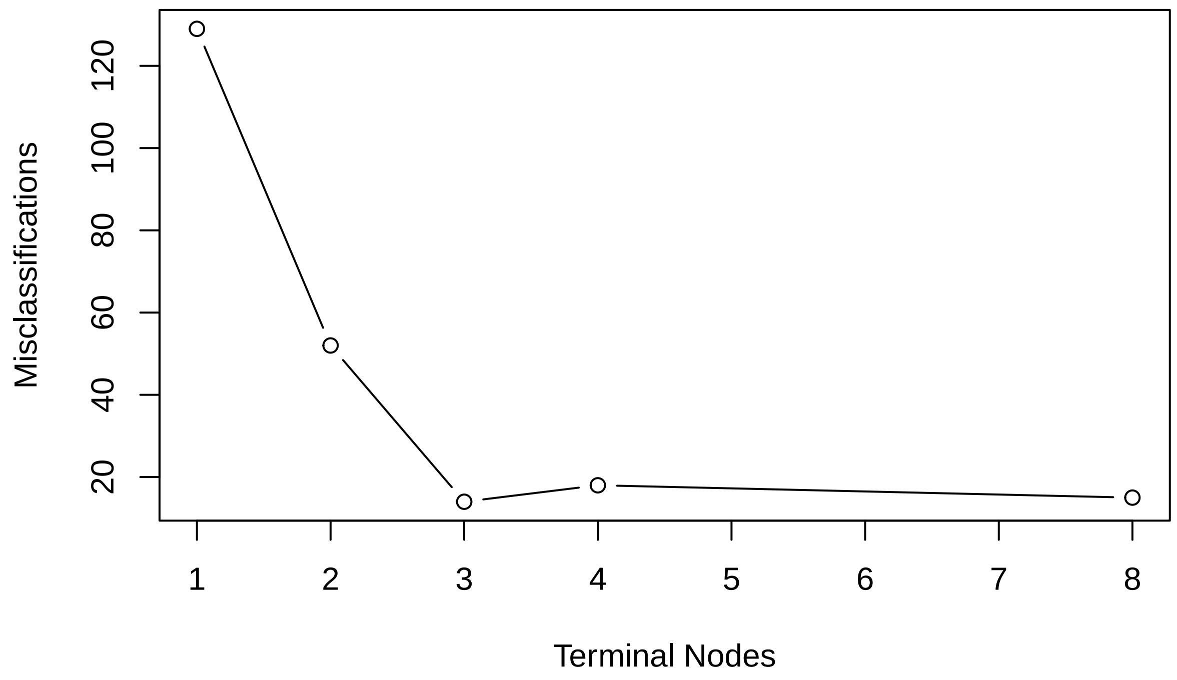
```
## tree(formula = species ~ ., data = train, split = "gini")
## Variables actually used in tree construction:
## [1] "flipperL" "billL"    "billD"    "mass"
## Number of terminal nodes:  8
## Residual mean deviance:  0.1869 = 42.06 / 225
## Misclassification error rate: 0.04292 = 10 / 233
```

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



- ii) We use the prune function to apply cost complexity pruning. The cv.tree function does automatically 10-fold cross validation. Moreover, we plot the misclassifications.

```
set.seed(123)
cv.head = cv.tree(tree.HIClass, FUN = prune.misclass)
plot(cv.head$size, cv.head$dev, type="b", xlab = "Terminal Nodes", ylab = "Misclassifications")
```



- iii) We will print the results from previously in order to see which number of nodes we will choose.

```
print(cv.head)
```

```
## $size
## [1] 8 4 3 2 1
##
## $dev
## [1]  15  18  14  52 129
##
## $k
## [1] -Inf    0    1   40   78
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

We choose the number of the nodes with the least misclassifications (=14) so we choose nodes = 3.

```
prune.HIClass = prune.misclass(tree.HIClass, best = 3)
tree.pred.prune <- predict(prune.HIClass, test, type="class")
confMat<-confusionMatrix(tree.pred.prune,test$species)
```

```r
error_rate<- 1 - sum(diag(confMat$table))/sum(confMat$table)
confMat$table
```

```
##             Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        42         3      0
##   Chinstrap      0        14      0
##   Gentoo         0         3     38
```

```r
cat("Error rate:",error_rate)
```

```
## Error rate: 0.06
```

d)

The basic parameters that could be tuned in Random Forest are the number of trees -ntree- and the number of variables tested at each split -mtry-. The number of trees is not really significant for tuning, it is enough if we have a large number of trees so we will leave that at the default value which is 500. The number of variables tested at each split though should be tuned and we will do that below using the model accuracy.

```r
control <- trainControl(method='repeatedcv', number = 10, repeats=3, search = 'random')
rf_random<- train(species~., data = train, method = 'rf', metric = 'Accuracy', tuneLength = 15, trContro
print(rf_random)
```

```
## Random Forest
##
## 233 samples
##   6 predictor
##   3 classes: 'Adelie', 'Chinstrap', 'Gentoo'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 210, 210, 209, 209, 210, 209, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.9942633  0.9908428
##   2     0.9942633  0.9908428
##   3     0.9884607  0.9816481
##   4     0.9842336  0.9750246
##   5     0.9814559  0.9706314
##   7     0.9800066  0.9684583
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 1.
```

According to the model accuracy the best mtry is 1 so we will use that.

```r
penguins.rf <- randomForest(species~., data = train, mtry = 1)
print(penguins.rf)
```

```
##
## Call:
##  randomForest(formula = species ~ ., data = train, mtry = 1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 0.86%
## Confusion matrix:
##           Adelie Chinstrap Gentoo class.error
## Adelie       104         0      0  0.00000000
## Chinstrap      2        46      0  0.04166667
## Gentoo         0         0     81  0.00000000
```

Now we will calculate the misclassification error rate for the test data.

```
pred.rf <- predict(penguins.rf, test)
confMat2 <- confusionMatrix(pred.rf, test$species)$table
confMat2
```

```
##            Reference
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        42         1      0
##   Chinstrap      0        19      0
##   Gentoo         0         0     38
```

```
error_rate2<-1 - sum(diag(confMat2))/sum(confMat2)
cat("Error rate:", error_rate2)
```

```
## Error rate: 0.01
```

Lastly, we calculate the most important predictors using the varImp function which returns the variable importance based on mean decrease in accuracy

```
varImp(penguins.rf)
```

```
##            Overall
## island   20.093523
## billL    36.652387
## billD    21.418852
## flipperL 26.173950
## mass     22.224653
## sex       1.323577
```

We can see that billL is the most important predictor followed by flipperL.

Problem 5

a)

- i) False

- ii) True
- iii) True
- iv) True

b)

Firstly, we use cross validation to find values for the support vector machine

```
tune.svm<-tune(svm,species~., kernel = "radial",
               data = train,
               ranges = list(cost = 10^(-3:3),
            gamma= 10^(-3:3)))
par.svm<-tune.svm$best.parameters
cat("Best cost for svm:", par.svm$cost,"\n")
```

```
## Best cost for svm: 100
```

```
cat("Best gamme for svm:",par.svm$gamma,"\n")
```

```
## Best gamme for svm: 0.01
```

```
tune.svc<-tune(svm, species~., kernel = "linear",
               data = train,
               ranges = list(cost =
                                10^(-3:3)))
par.svc<-tune.svc$best.parameters
cat("Best cost for svc:", par.svc$cost,"\n")
```

```
## Best cost for svc: 1
```

Form this we see that the best parameters for the support vector machine is when the cost is 100 and gamma is 0.01. The support vector classifier is optimal when the cost i 0.1. Using this, we firstly fit the svm

```
svm.fit<-svm(species~.,data = train,
             kernel = "radial",cost = 100,
             gamma = 0.01)

pred.svm<-predict(svm.fit,test)

table.svm<-table(test$species,pred.svm)
msrate.svm<-1-sum(diag(table.svm))/sum(table.svm)
cat("The missclassification rate is:",msrate.svm)
```

```
## The missclassification rate is: 0
```

```
table.svm
```

```
##            pred.svm
##             Adelie Chinstrap Gentoo
##   Adelie        42         0      0
##   Chinstrap      0        20      0
##   Gentoo         0         0     38
```

So we get perfect classification. For the svc

```
sv.fit<-svm(species~.,data = train, kernel = "linear", cost = 0.1, scale = FALSE)

pred.sv<-predict(sv.fit,test)
table.sv<-table(test$species,pred.sv)
msrate.sv<-1-sum(diag(table.sv))/sum(table.sv)
cat("The missclassification rate is:",msrate.sv)
```

```
## The missclassification rate is: 0.01
```

```
table.sv
```

```
##            pred.sv
##             Adelie Chinstrap Gentoo
##   Adelie        42         0      0
##   Chinstrap      1        19      0
##   Gentoo         0         0     38
```

So the scv has only one miss classification. Both methods preforms therefor very similar. Since svc is a simpler method than svm, but has very similar preformance, it is the recommended algorithm for this data set.

Problem 6

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

a)

- i) The variables "freedom to make life choices" and "corruption" point in opposite directions. The variables "logged GDR per capita", "healthy life expectancy" and "social support" point in very similar directions.
- ii) Afghanistan can be considered to be an outlier/anomaly.

b)

- i) Absolute values of first principal component by PCA

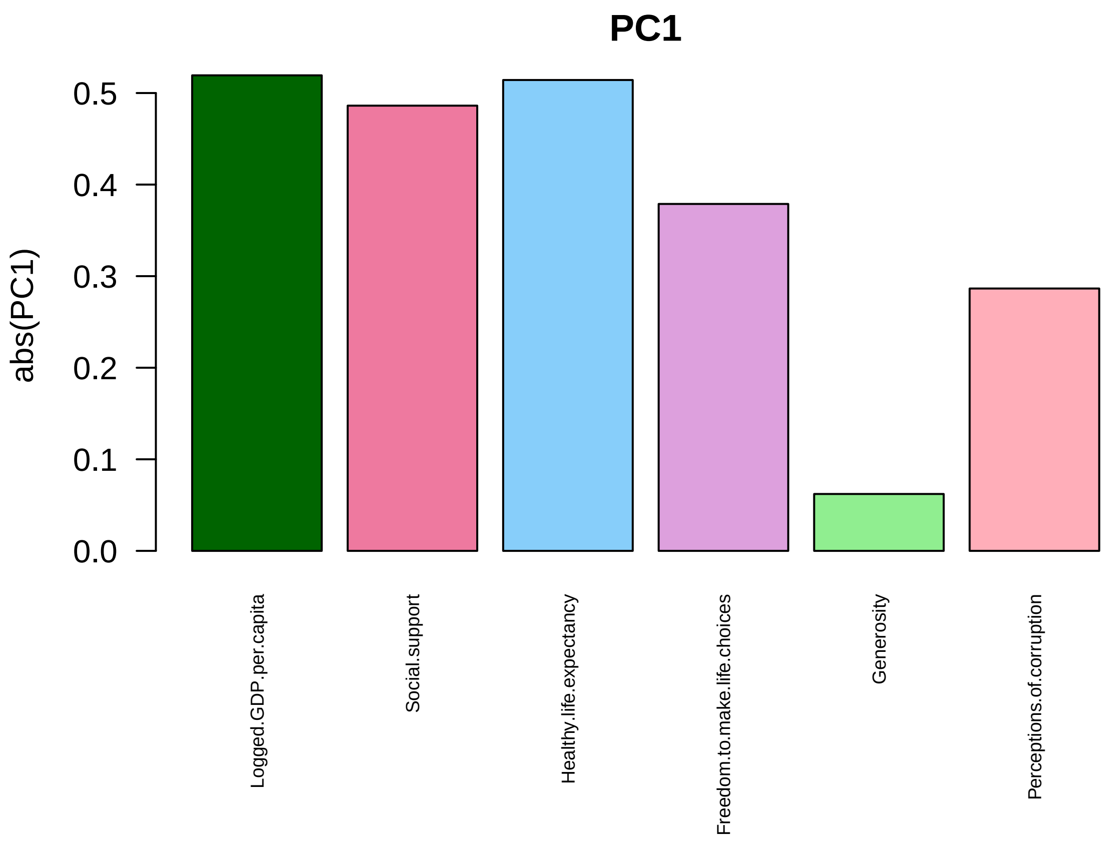
```
#Absolute values of first principal component PC1

PC1vals <- data.frame(pca_mat$rotation)$PC1
categories <- colnames(happiness)
vars <- categories[c(3:8)]
vars
```

```
## [1] "Logged.GDP.per.capita"        "Social.support"
## [3] "Healthy.life.expectancy"      "Freedom.to.make.life.choices"
## [5] "Generosity"                   "Perceptions.of.corruption"
```

```
absPC1vals<- c()
for(i in 1:length(PC1vals)){
  absPC1vals[i] = abs(PC1vals[i])
}
absPC1vals
```

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

```
par(mar = c(7, 4, 2, 2) + 0.5) #add room for the labels
barplot(absPC1vals,col = c("darkgreen", "palevioletred2", "lightskyblue", "plum", "palegreen2", "lightp
```



- ii) Fitting a partial least squares model (PLSR) on happiness.XY with response Ladder.score.

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

```
## Data:    X dimension: 149 6
##  Y dimension: 149 1
## Fit method: kernelpls
## Number of components considered: 6
```

```
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.078   0.5508   0.5569   0.5577   0.5612   0.5603   0.5601
## adjCV        1.078   0.5500   0.5555   0.5563   0.5594   0.5586   0.5584
##
## 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
## Ladder.score    75.10    75.50    75.55    75.58    75.58    75.58
```

- iii) Absolute values of first principal components for X in happiness.XY

```
#Absolute values of first principal component PC1
comp1vals<-plsr_model$loadings[,c('Comp 1')]
comp1vals
```

```
##        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
```

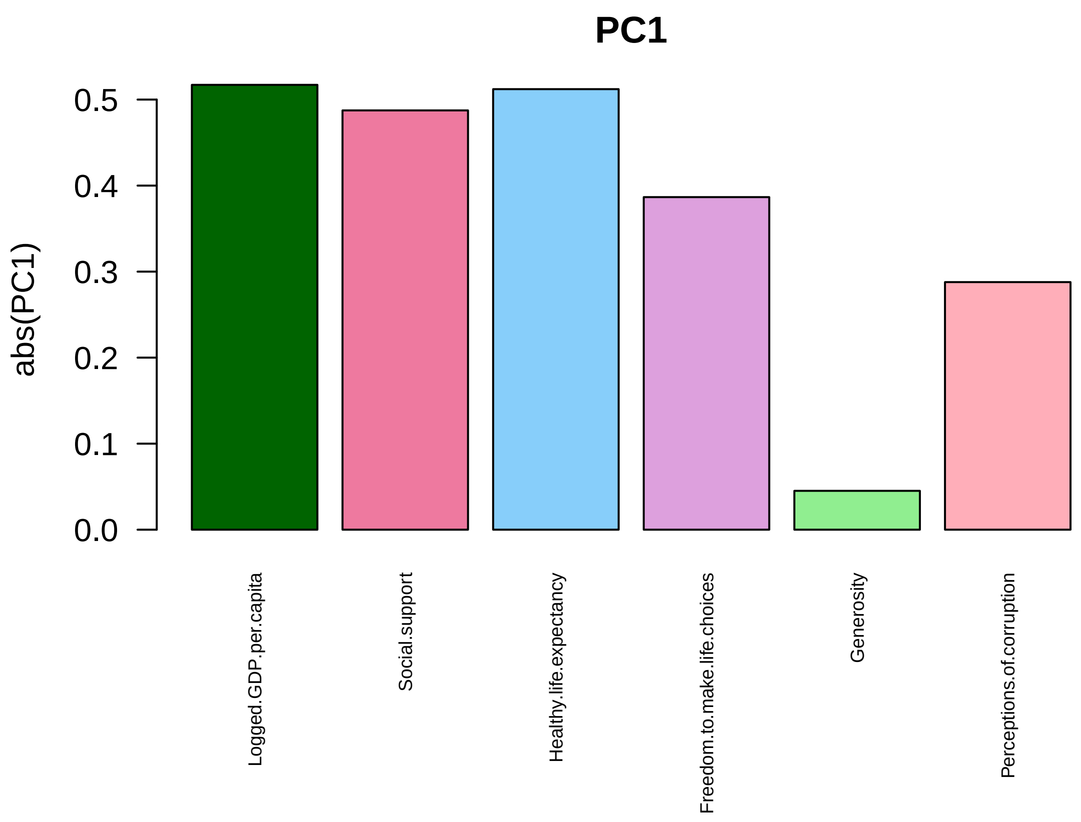
```
categories <- colnames(happiness)
vars <- categories[c(3:8)]
vars
```

```
## [1] "Logged.GDP.per.capita"        "Social.support"
## [3] "Healthy.life.expectancy"      "Freedom.to.make.life.choices"
## [5] "Generosity"                   "Perceptions.of.corruption"
```

```
abscomp1<- c()
for(i in 1:length(comp1vals)){
  abscomp1[i] = abs(comp1vals[i])
}
abscomp1
```

```
## [1] 0.51710666 0.48740123 0.51212197 0.38659271 0.04516483 0.28779558
```

```
par(mar = c(7, 4, 2, 2) + 0.9) #add room for the rotated labels
barplot(abscomp1, main="PC1",col = c("darkgreen", "palevioletred2", "lightskyblue", "plum", "palegreen2
```



Comparing the bar plot in **i)** with PCA to the bar plot above in **ii)** with PLSR, we see that they look very similar. It is difficult to spot any differences in the values from the bar plot, so we examine the absolute values of the PC1:

```
absPC1vals    #Abs values from PCA
```

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

```
abscomp1      #Abs values from PLSR
```

```
## [1] 0.51710666 0.48740123 0.51212197 0.38659271 0.04516483 0.28779558
```

All absolute values differ slightly from each other, but if we rank them by value the order is the same.

- iv) Based on the PLSR-plot, the three most important predictors to predict the happiness score are (from most to least) Logged.GDP.per.capita, Healthy.life.expectancy and Social.support. The values above confirms this.

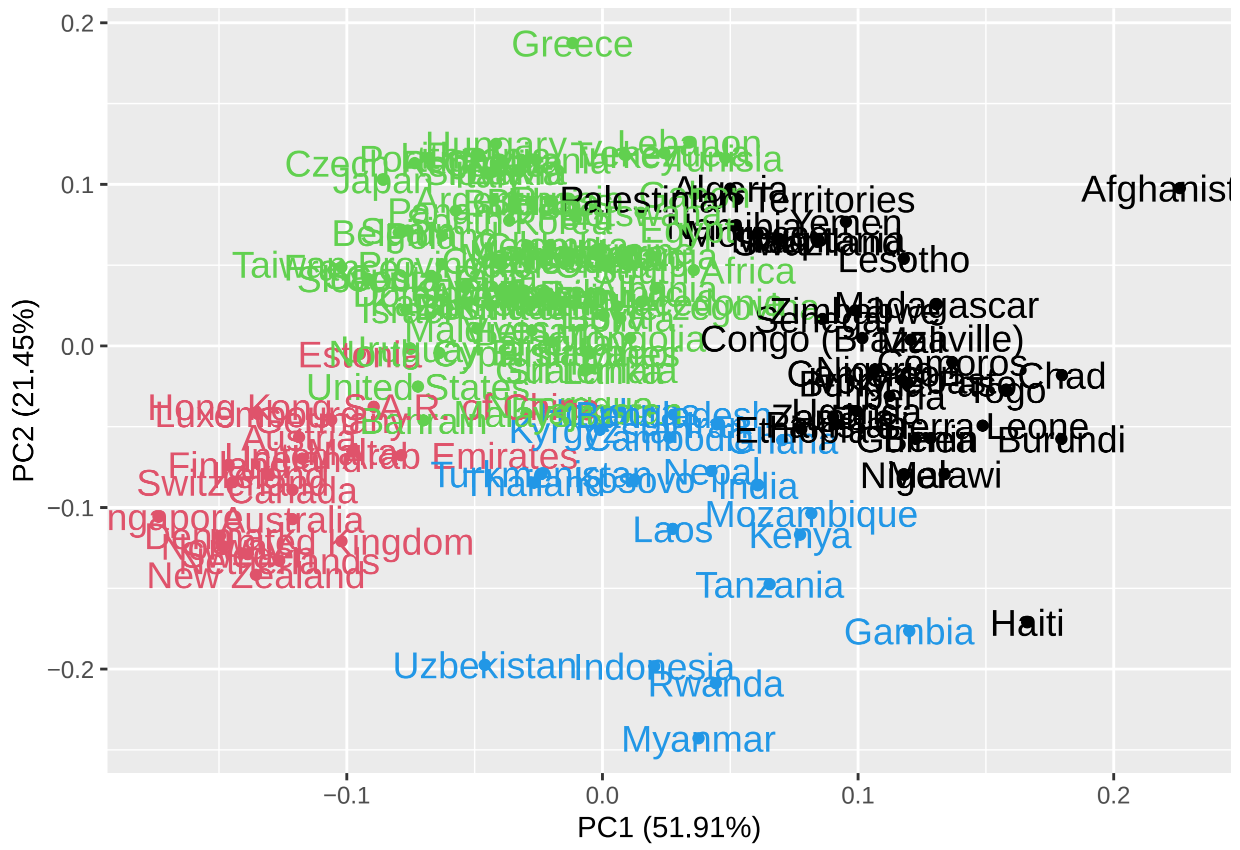
c)

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

d)

- i) K-means clusterization

```
K=4   #Number of clusters

km.out=kmeans(happiness.X, K)

autoplot(pca_mat, data=happiness.X, colour=km.out$cluster, label= T, label.size=5, loadings = F, loading
```



We see in the clusterization plot that our condition is satisfied. Norway, Sweden, Denmark and Finland are in the red cluster, while United states is in the green cluster.

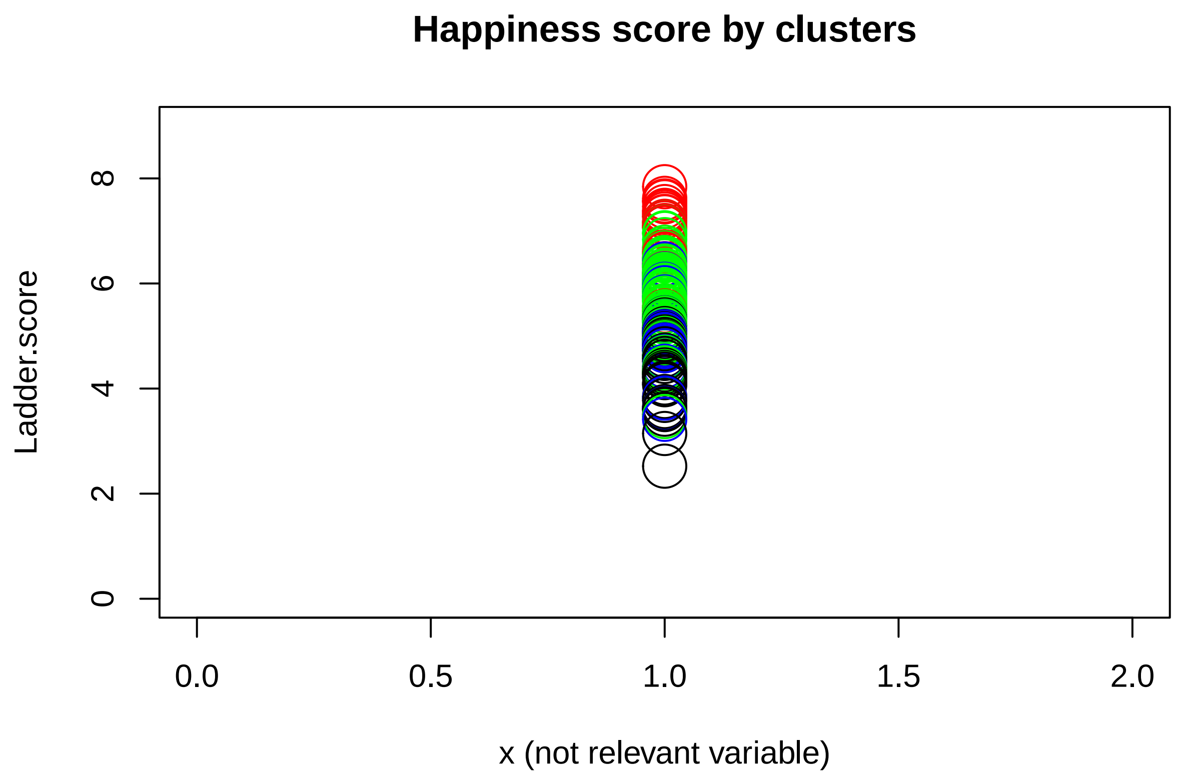
- ii)

  1. The four clusters are related to the happiness score, Ladder.score, where the red ranks highest and blue ranks lowest in Ladder.score. The happiest countries are placed in the red cluster and the least happy country, Afghanistan is in the black cluster. There is overlap between the happiness values of the countries in the clusters.

  2. The red cluster ranks highest on Freedom.to.make.life.choices, Social.support, Logged.GPD.per.capita and Healthy.life.expectancy. This suggests that high scores of these variables contribute to higher happiness overall.

  3. The black cluster ranks higher on corruption and lower on the variables mentioned in point 2. This suggests that high values of corruption contributes to a lower happiness scores.

```r
#Vector containing all happiness scores
lad<-c(happiness.Y[,2])

#Cluster colors by cluster index
cols=c("black", "red", "green", "blue")

clustercolvec<-km.out$cluster

#Vector of cluster numbers of all countries
newvec<-unname(clustercolvec)

#Vector for cluster indices of the countries
colr=c()
for(i in 1:149){
  colr[i]= cols[newvec[i]]
}

#Vector of ones
xvec<-rep(c(1),each=149)

#Making a scatterplot
plot(x = 1,
     type = "n",
     xlim = c(0, 2),
     ylim = c(0, 9),
     pch = 30,
     xlab = "x (not relevant variable)",
     ylab = "Ladder.score",
     main = "Happiness score by clusters")


points(x = xvec,
       y = lad,
       pch = 1,
       cex=3,
       col = colr)
```


Happiness score by clusters
Ladder.score
0
2
4
6
8
0.0
0.5
1.0
1.5
2.0
x (not relevant variable)