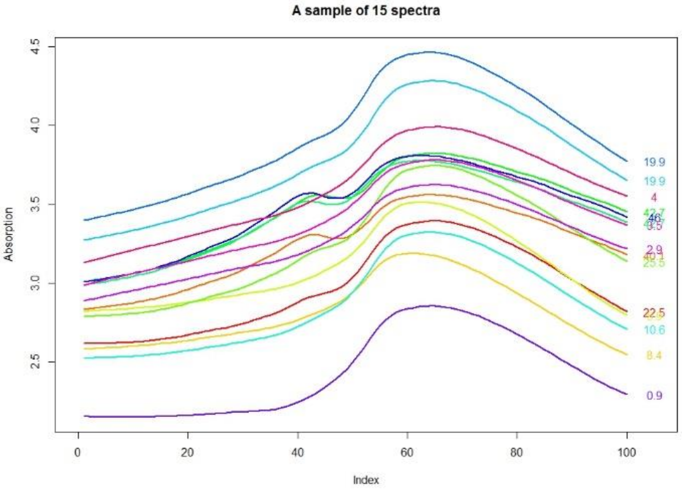
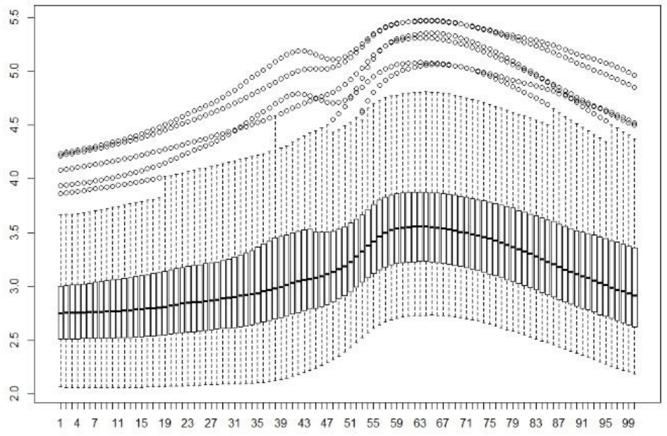
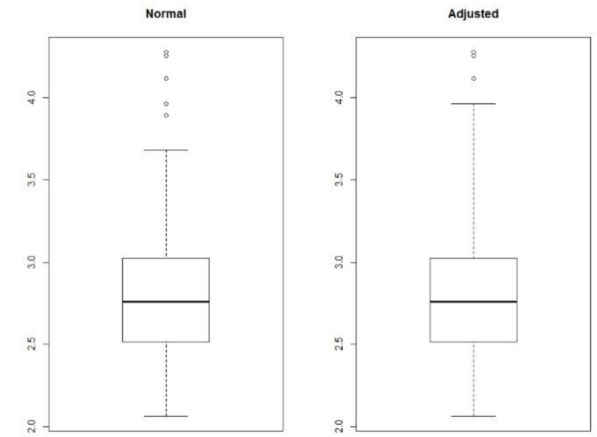
```r
#===============================================================================
#                             Using R: Infrared spectroscopy
#===============================================================================
#(a) load the data:
library(caret)
data(tecator)

#The matrix absorp contains the 100 absorbance values for the 215 samples, while matrix
#endpoints contains the percent of moisture, fat, and protein in columns 1-3,
respectively.
graph <- as.matrix(absorp)
plotColors <- rainbow(15)
plot(graph[1,], type="n", ylim=range(graph[1:15,]), xlim=c(0,105), ylab="Absorption")

for(i in 1:15){
  points(graph[i,], type = "l", col = plotColors[i], lwd = 2)
  text(105, graph[i,100], endpoints[i,2], col = plotColors[i])
}
title("A sample of 15 spectra")
boxplot(absorp)

#Below you can see the spectrum of the first 15 observation of data, which shows the
absorption. To explore all of the observations, we can use Boxplot.
```

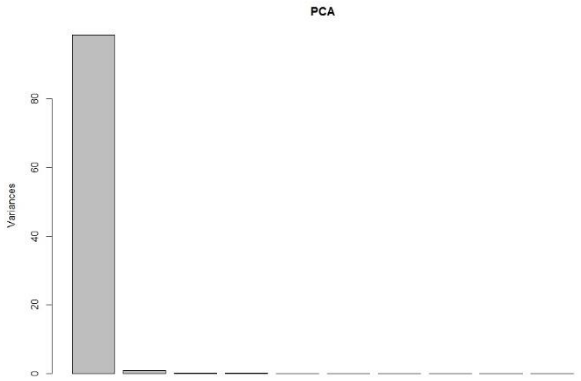




```r
par(mfrow =c(1,2))
boxplot(absorp[,5], main="Normal")
adjbox(absorp[,5], main="Adjusted")
par(mfrow =c(1,1))

# The boxplot shows a series of observations that could be potential outliers. In the
right hand side the boxplot and adjusted boxplot for the fifth predictor is shown. Both
of them show some potential outliers. However, in this problem we ignore the existence of
outliers. Overall, the data follow a smooth trend.
```

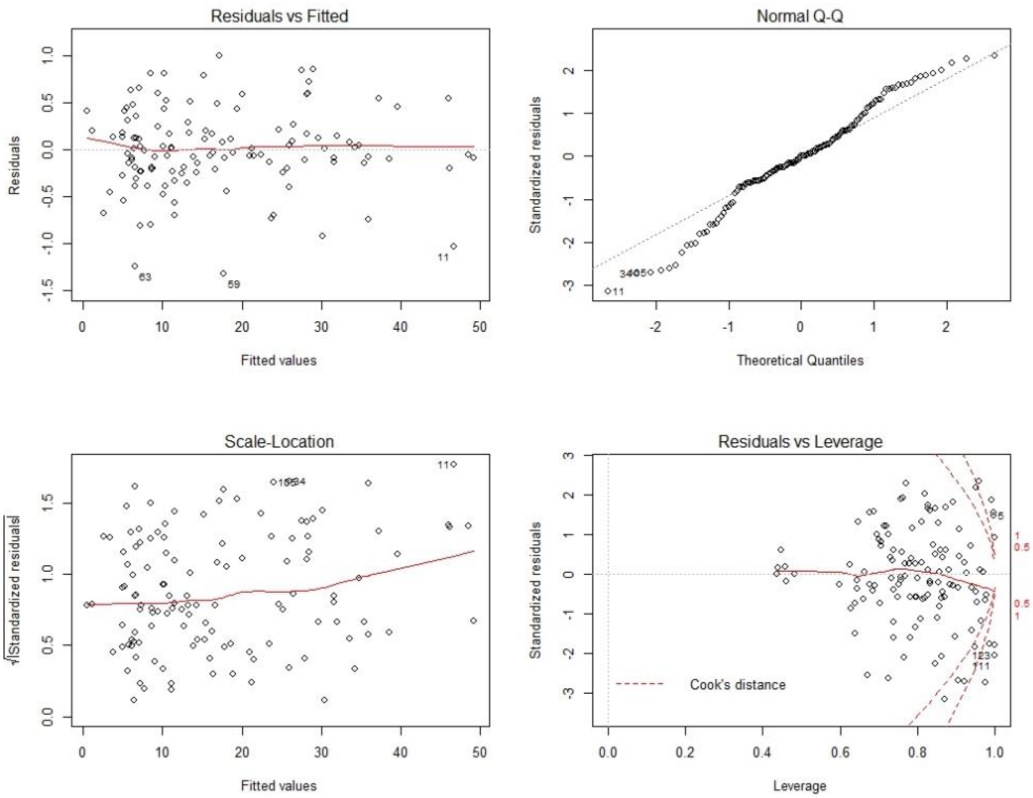


```
#---------------------------------------------------------------------------------------
#(b) Here the predictors are the measurements at the individual frequencies.
#Because the frequencies lie in a systematic order (850-1,050nm), the predictors have a
#high degree of correlation. Hence, the data lie in a smaller dimension than the total
#number of predictors (215). Use PCA to determine the effective dimension of these data.
#What is the effective dimension?
PCA <- prcomp(absorp, scale=T)
summary(PCA)
plot(PCA)

#As can be seen using only 1 PC, we can preserve almost more than 98 percent of the
information. The relative cumulative proportion of variance for the principal components
can be seen in this figure.
```

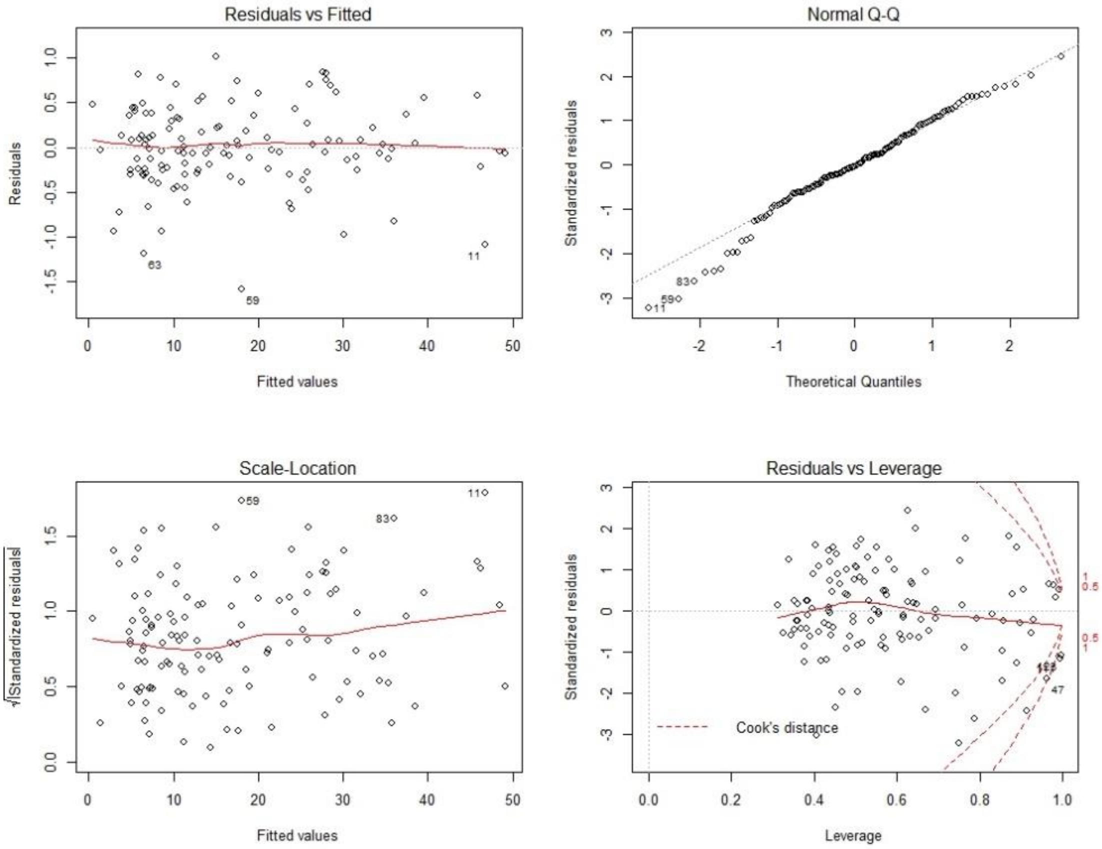


```
#---------------------------------------------------------------------------------------
#(c) Splitting the data into a training and a test set, pre-processing the data, and
building several models. For those models with tuning parameters, obtaining the optimal
values of the tuning parameter(s).
sum(is.na(absorp))
[1] 0
sum(is.na(endpoints))
[1] 0

#There is no missing data
#Using sample function, we split the data into two parts. 129 random observations from
215. The rest of the observations are kept for the test purpose.
```

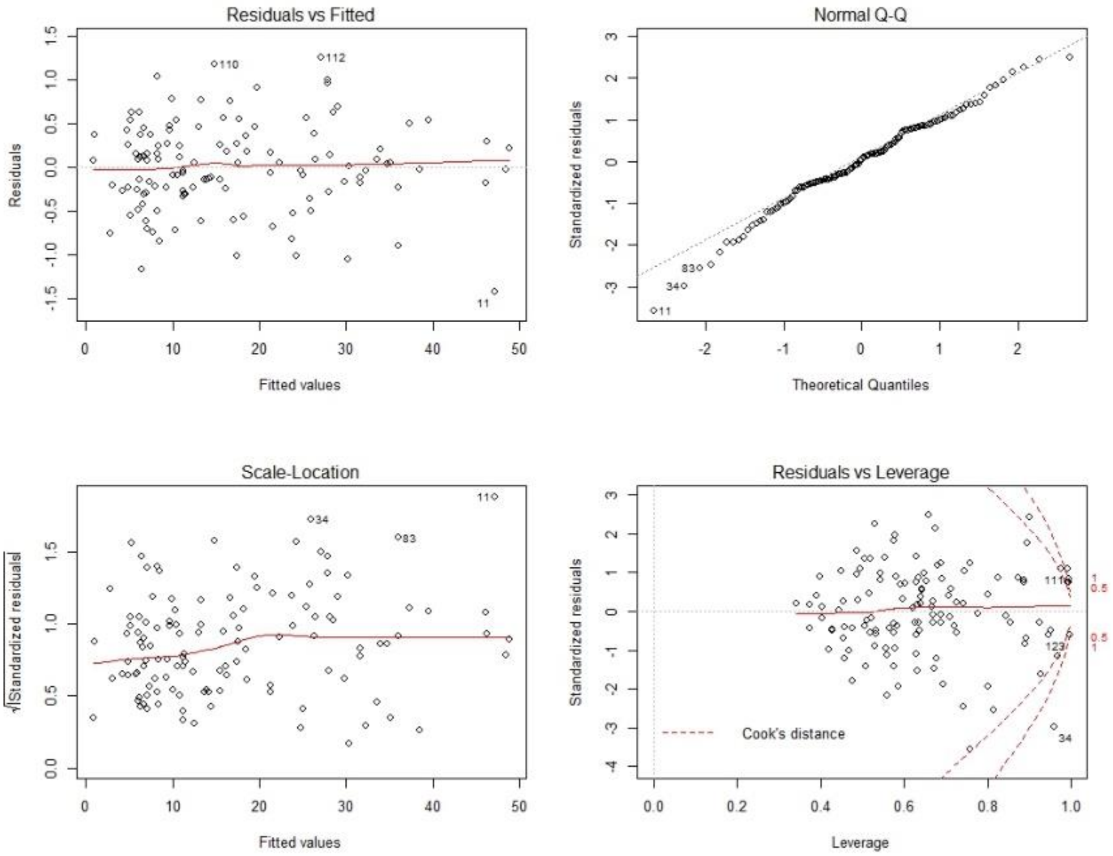
```
set.seed(100)
ss <- sample(215,129)
absorp.training <- data.frame(absorp[ss,])
absorp.test <- data.frame(absorp[-ss,])
endpoints.training <- data.frame(endpoints[ss,])
endpoints.test <- data.frame(endpoints[-ss,])

#Using the correlation command shows that the predictors are highly correlated, and
multicollinearity should be fixed.
round(cor(cbind(endpoints.training[,2],absorp.training)),3)

#For the purpose of comparing different method we use mean squared error

#Ordinary Least Squares (OLS)
fit.OLS <- lm(data=absorp.training, endpoints.training[,2] ~.)
summary(fit.OLS)
par(mfrow =c(2,2))
plot(fit.OLS)
par(mfrow =c(1,1))
MSE.OLS <- mean(fit.OLS$residuals^2)
RSS.OLS <- sum(fit.OLS$resid^2)/fit.OLS$df.residual
R.OLS <- summary(fit.OLS)$r.squared
AR.OLS <- summary(fit.OLS)$adj.r.squared
AIC.OLS <- AIC(fit.OLS)
BIC.OLS <- BIC(fit.OLS)
#MSE= 0.179
#As can be seen from the figure at right hand side, residuals do not show any trend and
are almost equally separated from the zero. The data are almost following a regression
and the QQ plot shows not a good fit. Some data have a little high cook’s distance and at
the same time high leverage.
```



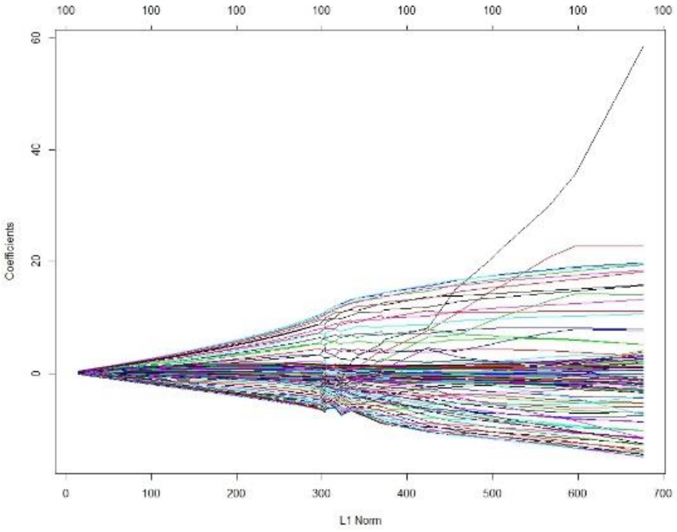
```
library(regclass)
VIF(fit.OLS)
#VIF shows very high values for all the variables, which confirms high multicollinearity.

#stepwise regression, backward
fit.backward <- step(fit.OLS, direction="backward")
MSE.backward <- mean(fit.backward$residuals^2)
RSS.backward <- sum(fit.backward$resid^2)/fit.backward$df.residual
R.backward <- summary(fit.backward)$r.squared
AR.backward <- summary(fit.backward)$adj.r.squared
AIC.backward <- AIC(fit.backward)
BIC.backward <- BIC(fit.backward)
par(mfrow =c(2,2))
plot(fit.backward)
par(mfrow =c(1,1))

#MSE= 0.198
#As can be seen from the figure at right hand side, it almost follows the same as linear
regression and only slightly has higher error. Residuals do not show any trend and are
almost equally separated from the zero. The data are following a regression and the QQ
plot doesn't show a good fit. Some data have a little high cook's distance and at the
same time high leverage. It should be noted that this method result simpler model and it
is preferred here comparing to linear regression.
```

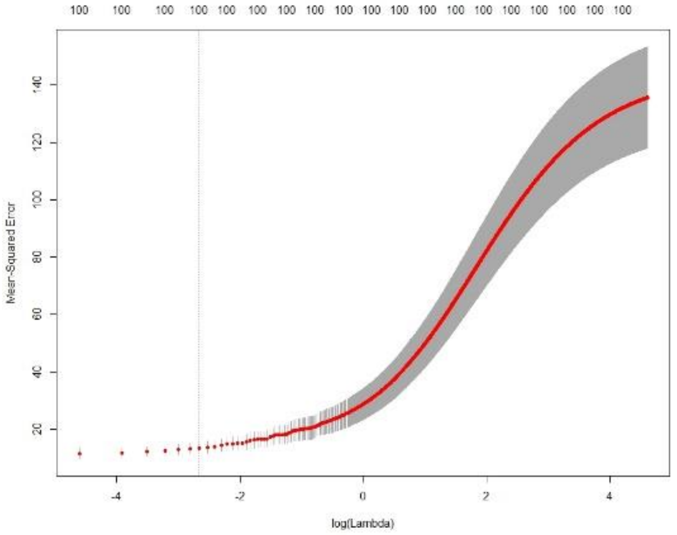


```
#stepwise regression, forward
scope <- list(upper=~endpoints.training[,2] + X1 + X2 + X3 + X4 + X5 + X6 +
                X7 + X8 + X9 + X10 + X11 + X12 + X13 + X14 + X15 + X16 +
                X17 + X18 + X19 + X20 + X21 + X22 + X23 + X24 + X25 + X26 +
                X27 + X28 + X29 + X30 + X31 + X32 + X33 + X34 + X35 + X36 +
                X37 + X38 + X39 + X40 + X41 + X42 + X43 + X44 + X45 + X46 +
                X47 + X48 + X49 + X50 + X51 + X52 + X53 + X54 + X55 + X56 +
                X57 + X58 + X59 + X60 + X61 + X62 + X63 + X64 + X65 + X66 +
                X67 + X68 + X69 + X70 + X71 + X72 + X73 + X74 + X75 + X76 +
                X77 + X78 + X79 + X80 + X81 + X82 + X83 + X84 + X85 + X86 +
                X87 + X88 + X89 + X90 + X91 + X92 + X93 + X94 + X95 + X96 +
                X97 + X98 + X99 + X100, lower=~.)
fit.forward <- step(lm(endpoints.training[,2] ~ 1, data=absorp.training), scope,
direction="forward")
MSE.forward <- mean(fit.forward$residuals^2)
RSS.forward <- sum(fit.forward$resid^2)/fit.forward$df.residual
R.forward <- summary(fit.forward)$r.squared
AR.forward <- summary(fit.forward)$adj.r.squared
AIC.forward <- AIC(fit.forward)
BIC.forward <- BIC(fit.forward)
par(mfrow =c(2,2))
plot(fit.forward)
par(mfrow =c(1,1))

#MSE= 0.239
#As can be seen from the figure at right hand side, comparing to the backward its QQ plot
shows better fit. Also, it has higher residuals. In the term of cook’s distance and
leverage it shows a little improvement compared to backward.
```



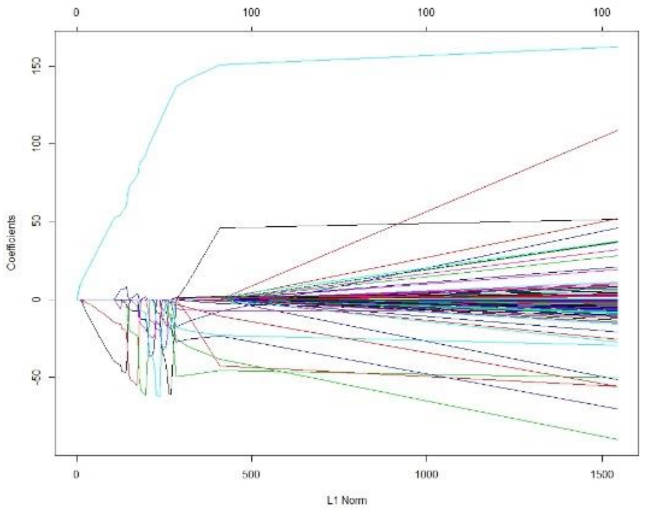
```
#Ridge Regression
#we use glmnet with alpha equal to 0 to perform ridge regression
library(glmnet)
fit.ridge <- glmnet(as.matrix(absorp.training), endpoints.training[,2], alpha=0,
lambda=seq(0,100,0.01))
plot(fit.ridge)
cv.out <- cv.glmnet(as.matrix(absorp.training), endpoints.training[,2], alpha=0,
lambda=seq(0,100,0.01))
plot(cv.out)
lambda.ridge <- cv.out$lambda.min
fit.ridge <- glmnet(as.matrix(absorp.training), endpoints.training[,2], alpha=0,
lambda=lambda.ridge)
resid.ridge <- endpoints.training[,2] - predict(fit.ridge, s=lambda.ridge
,newx=as.matrix(absorp.training))
MSE.ridge <- mean(resid.ridge^2)

#MSE=6.121
#The error is higher than the previous methods. In the figure on the left, the shrinkage
of coefficients is shown. In the tuning parameter which visualized on the right, the mean
squared error for various tuning parameter are shown using cross validation.
```

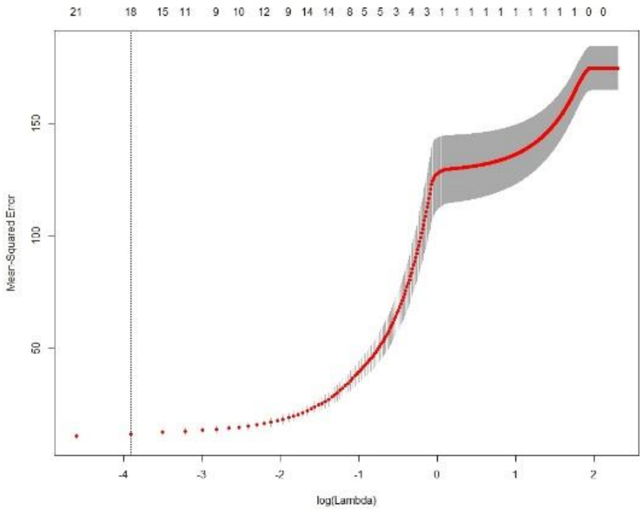




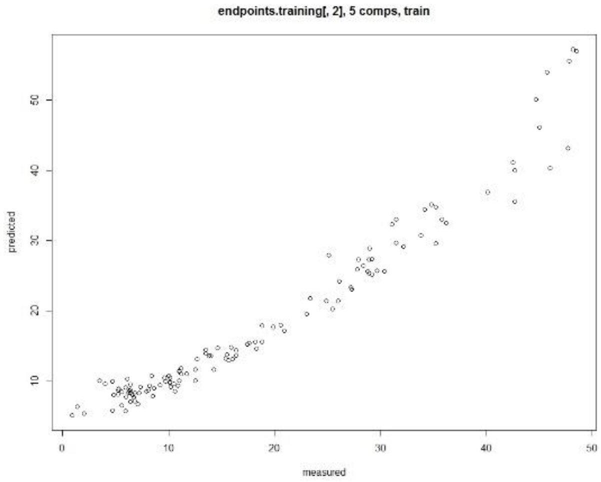
```
#LASSO Regression by help of glmnet with alpha equal to 1
fit.lasso <- glmnet(as.matrix(absorp.training), endpoints.training[,2], alpha=1,
lambda=seq(0,100,0.01))
plot(fit.lasso)
cv.out <- cv.glmnet(as.matrix(absorp.training), endpoints.training[,2], alpha=1,
lambda=seq(0,10,0.01))
plot(cv.out)
lambda.lasso <- cv.out$lambda.min
fit.lasso <- glmnet(as.matrix(absorp.training), endpoints.training[,2], alpha=1,
lambda=lambda.lasso)
resid.lasso <- endpoints.training[,2] - predict(fit.lasso, s=lambda.lasso
,newx=as.matrix(absorp.training))
MSE.lasso <- mean(resid.lasso^2)

#MSE=6.121
```

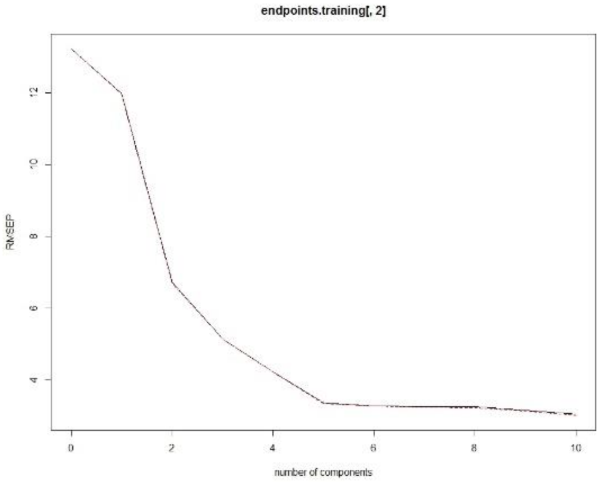
```
#The results are almost the same as ridge regression. We used alpha equals to 1 and tuned
the parameters using cross validation. The tuning parameter versus its error and the
shrinkage of the coefficient shown below.
```





```
#Partial Least Squares
library(pls)
fit.pls <- plsr(endpoints.training[,2] ~ ., 10, data=absorp.training, method =
"oscorespls", validation = "CV")
plot(fit.pls)
summary(fit.pls)
dev.off()
plot(RMSEP(fit.pls))
#It seems using only 5 components is enough
ncom <- 5
fit.pls <- plsr(endpoints.training[,2] ~ ., ncom, data=absorp.training, method =
"oscorespls")
summary(fit.pls)
resid.pls <- drop(fit.pls$resid)[,ncom]
MSE.pls <- mean(resid.pls^2)
#MSE=8.382
#It has the highest error among all the models. For tuning, cross validation method is
used.
```





```
#-------------------------------------------------------------------------------------
#(d) Here we can compare the MSE computed in previous part to help us rank and decide
which
method is better.

#Ordinary Least Squares (OLS)
MSE.OLS
#stepwise regression, backward
MSE.backward
#stepwise regression, forward
MSE.forward
#Ridge Regression
MSE.ridge
#LASSO Regression by help of CV
MSE.lasso
#Partial Least Squares
MSE.pls

#Here only the MSE is compared which is not a perfect method to use for ranking and
comparing the fitness of methods. In the previous part the MSE for training data is shown
for different method. Now we check the prediction of the fitted regression on the test
data.

#Testing prediction of each regression
test.MSE.OLS <- mean((endpoints.test[,2] - predict(fit.OLS, absorp.test))^2)
test.MSE.OLS
test.MSE.backward <- mean((endpoints.test[,2] - predict(fit.backward, absorp.test))^2)
test.MSE.backward
test.MSE.forward <- mean((endpoints.test[,2] - predict(fit.forward, absorp.test))^2)
test.MSE.forward
test.MSE.ridge <- mean((endpoints.test[,2] - predict(fit.ridge,s=lambda.ridge
,newx=as.matrix(absorp.test)))^2)
test.MSE.ridge
test.MSE.lasso <- mean((endpoints.test[,2] - predict(fit.lasso,s=lambda.lasso
,newx=as.matrix(absorp.test)))^2)
test.MSE.lasso
test.MSE.pls <- mean((endpoints.test[,2] - predict(fit.pls,absorp.test,ncom))^2)
test.MSE.pls
```

Method	MSE(Training)	MSE(Test)	Method	MSE(Training)	MSE(Test)
Linear	0.179	18.531	Ridge	6.121	9.810
Backward	0.198	15.337	LASSO	6.121	9.810
Forward	0.239	18.270	PLS	8.382	11.174

```
#As can be seen the first 3 method although they have less error in the training, but
they have a very high error in the test. So, they are not trustworthy. The last 3 method
has higher error in the training. However, they have consistent error in the test
comparing to the first 3 methods. So, these methods are preferred because we are not
looking to make zero error, instead we want to know how are model is consistent with new
data.
```