Assignment 2:

Step 1

In this step, the data is split into a training data set (60%) and a test data set (40%). The data is then scaled by using the preProcess function. This is necessary to construct good regression models.

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Loading required package: lattice

## # A tibble: 3,525 x 22
##    subject.     age    sex test_time motor_UP~1 total~2 Jitte~3 Jitte~4 Jitter~5
##       <dbl>   <dbl>  <dbl>     <dbl>      <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
##  1   -1.66   0.816  -0.682    -1.50       0.880  0.549  -0.573   -0.785 -0.636
##  2   -1.33   1.16   -0.682     1.15       1.47   1.42   -0.0967   0.122 -0.172
##  3   -1.33   1.16   -0.682    -0.642      1.19   1.11   -0.314   -0.118 -0.393
##  4    1.41   0.135  -0.682     0.393      1.32   1.28    0.163    0.199  0.436
##  5   -0.121  0.249  -0.682     1.23      -1.21  -1.15    0.134    0.277  0.161
##  6    1.17  -0.318   1.47     -1.47      -0.144 -0.192   4.36     2.78   3.48
##  7   -1.33   1.16   -0.682    -0.290      1.19   1.17   -0.459   -0.345 -0.412
##  8   -0.444  0.0221 -0.682     0.311     -1.58  -0.904   0.191    0.521 -0.0151
##  9   -0.687  0.929   1.47      0.502     -0.116 -0.0990 -0.146   -0.480  0.00410
## 10   -1.66   0.816  -0.682    -1.63       0.849  0.503  -0.708   -0.906 -0.681
## # ... with 3,515 more rows, 13 more variables: Jitter.PPQ5 <dbl>,
## #   Jitter.DDP <dbl>, Shimmer <dbl>, Shimmer.dB. <dbl>, Shimmer.APQ3 <dbl>,
## #   Shimmer.APQ5 <dbl>, Shimmer.APQ11 <dbl>, Shimmer.DDA <dbl>, NHR <dbl>,
## #   HNR <dbl>, RPDE <dbl>, DFA <dbl>, PPE <dbl>, and abbreviated variable names
## #   1: motor_UPDRS, 2: total_UPDRS, 3: Jitter..., 4: Jitter.Abs., 5: Jitter.RAP
```

Step 2

In this step, the linear model is constructed. The Parkinson disease symptom score (motor_UPDRS) is predicted based on variables representing voice characteristics. The mean square error for the training and test data is computed.

```
##
## Call:
## lm(formula = motor_UPDRS ~ Jitter... + Jitter.Abs. + Jitter.RAP +
##     Jitter.PPQ5 + Jitter.DDP + Shimmer + Shimmer.dB. + Shimmer.APQ3 +
##     Shimmer.APQ5 + Shimmer.APQ11 + Shimmer.DDA + NHR + HNR +
##     RPDE + DFA + PPE, data = trainS)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0162 -0.7340 -0.1084  0.7310  2.1892
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.006789   0.015790   0.430 0.667252
## Jitter...       0.180301   0.144277   1.250 0.211496
## Jitter.Abs.    -0.169795   0.040856  -4.156 3.32e-05 ***
## Jitter.RAP     -5.087728  18.186914  -0.280 0.779688
## Jitter.PPQ5    -0.071963   0.084712  -0.850 0.395659
## Jitter.DDP      5.068468  18.190294   0.279 0.780541
## Shimmer         0.590434   0.205314   2.876 0.004055 **
## Shimmer.dB.    -0.172730   0.139396  -1.239 0.215380
## Shimmer.APQ3   32.009848  77.023255   0.416 0.677738
## Shimmer.APQ5   -0.387249   0.113730  -3.405 0.000669 ***
## Shimmer.APQ11   0.310751   0.062288   4.989 6.37e-07 ***
## Shimmer.DDA   -32.325644  77.023042  -0.420 0.674739
## NHR            -0.186169   0.045766  -4.068 4.85e-05 ***
## HNR            -0.239653   0.036570  -6.553 6.45e-11 ***
## RPDE            0.004059   0.022615   0.179 0.857576
## DFA            -0.276898   0.019893 -13.919  < 2e-16 ***
## PPE             0.229100   0.033268   6.886 6.75e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9367 on 3508 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.24 on 16 and 3508 DF,  p-value: < 2.2e-16
```

```
## [1] 0.873147
```

```
## [1] 0.9297021
```

By looking at the p-value (forth column)for the estimated values of the coefficients, we can determine that the independent variables *Jitter.Abs, Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA,* and *PPE* contribute significantly to the model.

Task 3

The formulas for completing this were found in the lecture slides unless something else is stated.

Task 4

In this task, ridge regression is used with different lambda on the training data. The accuracy of the model is then evaluated by calculating the MSE of the test data and degrees of freedom for each lambda is computed.

```
## [1] "lambda: 1"
## [1] "train MSE: 0.873276950507478"
## [1] "test_MSE: 0.929035685734743"
## [1] "df_test: 13.7804331534807"
## [1] "df_train 13.8628107030632"
```

```
## [1] "lambda: 100"
## [1] "train MSE: 0.879059936030921"
## [1] "test_MSE: 0.926316386508651"
## [1] "df_test: 9.18289840204118"
## [1] "df_train 9.93908507636864"
```

```
## [1] "lambda: 1000"
## [1] "train MSE: 0.915626779289396"
## [1] "test_MSE: 0.947916644425209"
## [1] "df_test: 4.82167557512052"
## [1] "df_train 5.64335089965714"
```

We get the lowest test MSE for lambda = 100. This tells us that lambda = 100 is closer to the optimum than lambda = 1 or lambda = 1000 (after a certain threshold, adding a higher shrinkage penalty will result in higher variance, which in turn will result in a higher MSE).

We can see that as lambda increases, the degrees of freedom decrease. This makes sense since if there is no penalization, we have as many parameters as there are in the model. On the contrary, when lambda approaches infinity, the degrees of freedom will approach zero.

```r
library(knitr)
library(dplyr)
library(tidyr)
library(caret) #for preprocessing of data
set.seed(12345)
data = read.csv("parkinsons.csv")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train= tibble(data[id,])
test=tibble(data[-id,])

scaler=preProcess(data)
trainS=predict(scaler,train)
testS=predict(scaler,test)
trainS
# Computing the linear model and the MSE for the training data
m1 = lm(motor_UPDRS~ Jitter... + Jitter.Abs. + Jitter.RAP +
          Jitter.PPQ5 + Jitter.DDP + Shimmer + Shimmer.dB. +
          Shimmer.APQ3 + Shimmer.APQ5 + Shimmer.APQ11 +
          Shimmer.DDA + NHR + HNR + RPDE + DFA + PPE, data = trainS)
summary(m1)
preds = predict(m1, trainS)
MSE = sum((trainS$motor_UPDRS - preds)^2) / length(preds)
MSE
# MSE for the test data
preds = predict(m1, testS)
MSE = sum((testS$motor_UPDRS - preds)^2) / length(preds)
MSE
# The purpose of this function is to find values for the thetas and
# sigmas which maximizes the probability for our model to generate the data.
LogLikelihood <- function(data, theta, sigma){
  x = as.matrix(data[ ,7:22])
  y = as.matrix(data[, 5])
  n = length(y) # observations
  logLik = -(n/2)*(log(2*pi) + log(sigma^2)) - (2*sigma^2)^-1 * sum((x%*%theta - y)^2) #squared loss is
  return(logLik)
}
# The purpose of ridge regression is to reduce the variance when the independent
# variables have high correlarity with one another.This is accomplished by
# adding a shrinkage penalty which increases the bias a little bit but
# in turn reduce the variance significantly, which leads to a lower MSE.
Ridge <- function(sigmaThetas, lambda, data){
  sigma = sigmaThetas[1]
  theta = sigmaThetas[-1] #removes first element (sigma)
  logLik = LogLikelihood(data, theta, sigma)
  ridge = lambda * sum(theta^2) - logLik
  return(ridge)
}
RidgeOpt <- function(lambda, data){
  initSigma = 1
  initThetas = rep(1, 16)
  optRes = optim(par=c(initSigma, initThetas), fn=Ridge, lambda=lambda,
                 data=data, method="BFGS")
  return(optRes)
}
# formula found at https://online.stat.psu.edu/stat508/lesson/5/5.1
DF <- function(lambda, X){
  I = diag(dim(X)[2])
  #Sum of diagonal is the same as the trace function (tr)
  temp = X %*% solve(t(X)%*%X + lambda*I)%*%t(X)
  dt = sum(diag(temp))
  return(dt)
}

lambda = c(1, 100, 1000)
#lambda[1] = 1
#lambda[2] = 100
#lambda[3] = 1000
testOpt <- function(lambda) {
  opt = RidgeOpt(lambda, trainS)
  sigma = opt$par[1]
  thetas = as.matrix(opt$par[-1])
  X_train = as.matrix(trainS[, 7:22])
  X_test = as.matrix(testS[, 7:22])
  Y_train = as.matrix(trainS[, 5])
  Y_test = as.matrix(testS[, 5])

  train_predict = X_train %*% thetas
  test_predict = X_test %*% thetas

  #calculate MSE for train and test data
  train_MSE = (length(Y_train))^-1 * sum((Y_train - train_predict)^2)
  test_MSE = (length(Y_test))^-1 * sum((Y_test - test_predict)^2)

  #calculate degrees of freedom
  df_train = DF(lambda, X_train)
  df_test = DF(lambda, X_test)

  print(paste("lambda:", lambda))
  print(paste("train MSE:", train_MSE))
  print(paste("test_MSE:", test_MSE))
  print(paste("df_test:", df_test))
  print(paste("df_train", df_train))

}
opt1 = testOpt(lambda[1])
opt2 = testOpt(lambda[2])
opt3 = testOpt(lambda[3])
```