STAT 5330: HW2

Greyson Hardison

3/26/2021

On my honor as a student I have neither given nor received aid on this assignment.

Greyson Hardison

Any code not shown in the pdf can be found in the R Markdown file submitted with the assignment.

The following packages were used:

```
library(ROCR)
library(ggplot2)
library(pROC)
library(glmnet)
library(MASS)
library(splines2)
library(ISLR)
library(tree)
library(randomForest)
library(caret)
library(tidyverse)
library(corrplot)
```

Question 1

The code below shows the process for reading in the data and making training and testing sets.

```
vowel <- read.delim("vowel.txt", sep = ",")[, -1]
vowel$y <- as.factor(vowel$y)
set.seed(1)
training.samples <- vowel$y %>%
  createDataPartition(p = 0.8, list = FALSE)
vowel.train <- vowel[training.samples, ]
vowel.test <- vowel[-training.samples, ]
```

Part a

The code below is for transforming the data based on estimated parameters and finding the centroids and between class covariance.

```
preproc.param <- vowel.train %>%
  preProcess(method = c("center", "scale"))
# Transform the data using the estimated parameters
train.transformed <- preproc.param %>% predict(vowel.train)
test.transformed <- preproc.param %>% predict(vowel.test)

M <- aggregate(train.transformed[, -1], list(train.transformed$y),mean)
M <- as.matrix(M[, -1])
B <- cov(M)
```

The centroids were calculated and the matrix of values are printed below.

```
M
```

```
##               x.1         x.2         x.3        x.4         x.5         x.6
##  [1,] -0.27372111 -1.37877220  0.24268488  0.9167292  1.02138852  0.92687567
##  [2,]  0.46263651 -1.06194408 -0.14123806  0.3789020  0.83739450  0.63344123
##  [3,]  0.74170072 -0.84501210 -0.49655498  0.3615506  0.62116538  0.01503910
##  [4,]  1.04963546 -0.19838407 -0.52355038 -0.2124939  0.10392246 -0.59563897
##  [5,]  0.41374201  0.43166147 -0.09488020 -0.3347767 -0.86954945 -0.24626486
##  [6,]  0.45545861  0.02746323 -0.10843042 -0.2434103 -0.36362761 -0.22185844
##  [7,] -0.06643595  0.65124576  0.47843072 -0.1206007 -0.90361850 -0.67303877
##  [8,] -0.93045961  1.30607589  0.30042213 -0.1914549 -0.88129375 -0.52675952
##  [9,] -0.72076416  0.51698107  0.18862405 -0.2346523 -0.04078282  0.29672820
## [10,] -1.38038972  0.82770220  0.20262944 -0.1353510  0.55503757  0.08021579
## [11,]  0.24859724 -0.27701717 -0.04813719 -0.1844419 -0.08003629  0.31126057
##                x.7          x.8         x.9          x.10
##  [1,]  0.0955950223 -0.629092443 -0.50994356 -0.1255429468
##  [2,] -0.5022948231 -0.106737436 -0.38548857  0.3938352341
##  [3,] -0.6040851373 -0.372835487 -0.38729470  0.5056312737
##  [4,]  0.0003387283 -0.463563485 -0.11717319  0.0722395419
##  [5,]  0.4748381929 -0.029864665  0.06348186 -0.2790349035
##  [6,]  0.3370297675 -0.433133780  0.05810547  0.2575951906
##  [7,]  0.0081687525  0.601907789  0.49594510 -0.1397905691
##  [8,]  0.1937673253  0.684909663  0.53719208  0.0831642363
```

```
##  [9,] -0.0054932897  0.557148068  0.11275142 -0.0008193521
## [10,]  0.2223874137 -0.005391414  0.15198225 -0.4536934556
## [11,] -0.2202519525  0.196653190 -0.01955814 -0.3135842496
```

The between class variance is also printed below.

```
B
```

```
##              x.1        x.2          x.3          x.4         x.5          x.6
## x.1   0.56763815 -0.3600323 -0.187718704  0.021638012  0.02756176 -0.049001828
## x.2  -0.36003230  0.7128280  0.127742160 -0.259173064 -0.45073954 -0.295029695
## x.3  -0.18771870  0.1277422  0.101837971 -0.001918582 -0.06997684  0.006426214
## x.4   0.02163801 -0.2591731 -0.001918582  0.148975561  0.20626546  0.140228444
## x.5   0.02756176 -0.4507395 -0.069976845  0.206265457  0.49387925  0.279930459
## x.6  -0.04900183 -0.2950297  0.006426214  0.140228444  0.27993046  0.264704793
## x.7  -0.08948889  0.1626861  0.042907829 -0.058843691 -0.12704913 -0.059047852
## x.8  -0.18724323  0.2878582  0.092830069 -0.090801495 -0.20171836 -0.081653451
## x.9  -0.12249831  0.2706076  0.062369061 -0.097014668 -0.20129832 -0.125859592
## x.10  0.11501084 -0.1071603 -0.049532980  0.035268482  0.04984611  0.002528516
##              x.7         x.8         x.9         x.10
## x.1  -0.08948889 -0.18724323 -0.12249831  0.115010838
## x.2   0.16268615  0.28785819  0.27060759 -0.107160346
## x.3   0.04290783  0.09283007  0.06236906 -0.049532980
## x.4  -0.05884369 -0.09080150 -0.09701467  0.035268482
## x.5  -0.12704913 -0.20171836 -0.20129832  0.049846106
## x.6  -0.05904785 -0.08165345 -0.12585959  0.002528516
## x.7   0.11010276  0.01484402  0.05741558 -0.057777280
## x.8   0.01484402  0.21300541  0.12581937 -0.037516247
## x.9   0.05741558  0.12581937  0.11505077 -0.038240387
## x.10 -0.05777728 -0.03751625 -0.03824039  0.090659572
```

Part b

The process for finding the within class covariance is shown below and also printed

```
Sigma <- cov(train.transformed[, -1] - M[as.factor(train.transformed$y),])
```

```
Sigma
```

```
##              x.1         x.2         x.3         x.4         x.5          x.6
## x.1   0.48275963 -0.19246298 -0.27219031 -0.02828032 -0.17749003  0.285571152
## x.2  -0.19246298  0.35046049 -0.02470299 -0.17224845  0.06398390 -0.171721341
## x.3  -0.27219031 -0.02470299  0.90720372  0.13685612  0.04362489 -0.534414303
## x.4  -0.02828032 -0.17224845  0.13685612  0.86425124 -0.27499769 -0.015169497
## x.5  -0.17749003  0.06398390  0.04362489 -0.27499769  0.54996984 -0.154212905
## x.6   0.28557115 -0.17172134 -0.53441430 -0.01516950 -0.15421291  0.758797035
## x.7   0.04914827  0.07859456 -0.34523581 -0.51026830  0.11490000  0.091867260
## x.8   0.35801299 -0.11369404 -0.20291758  0.03765705 -0.28902476  0.307737485
## x.9   0.13522212 -0.16240613  0.09935418  0.35005525 -0.36340395 -0.009609704
## x.10 -0.18070352 -0.04079194  0.35529156  0.16245543  0.01391413 -0.305317720
##              x.7         x.8          x.9         x.10
## x.1   0.04914827  0.35801299  0.135222120 -0.180703520
```

```
## x.2   0.07859456 -0.11369404 -0.162406130 -0.040791940
## x.3  -0.34523581 -0.20291758  0.099354181  0.355291558
## x.4  -0.51026830  0.03765705  0.350055249  0.162455431
## x.5   0.11490000 -0.28902476 -0.363403954  0.013914127
## x.6   0.09186726  0.30773748 -0.009609704 -0.305317720
## x.7   0.89967272 -0.10116706 -0.272871858 -0.242166782
## x.8  -0.10116706  0.80590629  0.044974975 -0.266194159
## x.9  -0.27287186  0.04497498  0.895164019 -0.001700471
## x.10 -0.24216678 -0.26619416 -0.001700471  0.917389643
```

Part c

Now we find sigma inverse in order to find the eigenvectors. We also use lda to find the linear discriminate coordinates. both of them are printed below the code. It does seem that the values are often trending negative or positive in both matrices but the magnitudes are typically different.

```
Sigma.inver <- solve(Sigma)
Sigma.inver.B <- Sigma.inver %*% B
eigen.B <- eigen(Sigma.inver.B)

vowel.lda <- lda(y ~ ., data = train.transformed)
v <- vowel.lda$scaling
```

```
eigen.B$vectors
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
##  [1,] -0.46763866 -0.47094645 -0.01010222  0.24504319 -0.11275794 -0.31107250
##  [2,]  0.59148342 -0.26027615  0.20040072  0.53019246 -0.20511371 -0.49075368
##  [3,]  0.20364457  0.15748239 -0.46201524 -0.05523217 -0.09465295 -0.15195338
##  [4,]  0.03501623  0.22221476 -0.12662191  0.41780792  0.58677574 -0.12739197
##  [5,]  0.07049264  0.58851557  0.56083012  0.44037640  0.03151866 -0.11397021
##  [6,]  0.26936794  0.43216757 -0.26297654 -0.08554344 -0.50541484 -0.49350869
##  [7,]  0.23321000  0.16173059 -0.33549186  0.35730292  0.36572230 -0.26323575
##  [8,]  0.40502541  0.23633324  0.33259316 -0.32835271  0.38035101  0.08964528
##  [9,]  0.29432148  0.14113012  0.24670527  0.17088931 -0.05737401 -0.05101439
## [10,]  0.07860210  0.03519064  0.25004547 -0.12370386  0.23194830 -0.53535441
##              [,7]        [,8]        [,9]        [,10]
##  [1,] 0.52162011 -0.38927428  0.28679729 -0.238013915
##  [2,] 0.42739607 -0.59923640  0.54370387 -0.208862044
##  [3,] 0.44665165 -0.06376760  0.37201057  0.263186455
##  [4,] 0.16354828 -0.47902598  0.21459499 -0.371858261
##  [5,] 0.42911902 -0.12570603 -0.03003142 -0.137805928
##  [6,] 0.19013488 -0.22907022  0.32337301 -0.317326235
##  [7,] 0.21968502 -0.06241903 -0.40616280 -0.097241271
##  [8,] 0.13433424 -0.13511050 -0.40209123  0.005643878
##  [9,] 0.17501975  0.39288051 -0.06216885 -0.748410521
## [10,] 0.04202159  0.10520373 -0.04132798  0.053739251
```

```
v
```

```
##             LD1         LD2         LD3         LD4         LD5         LD6
## x.1  -0.9810892  0.94916706  0.01836702  0.40488219  0.17435888  0.6277787
```

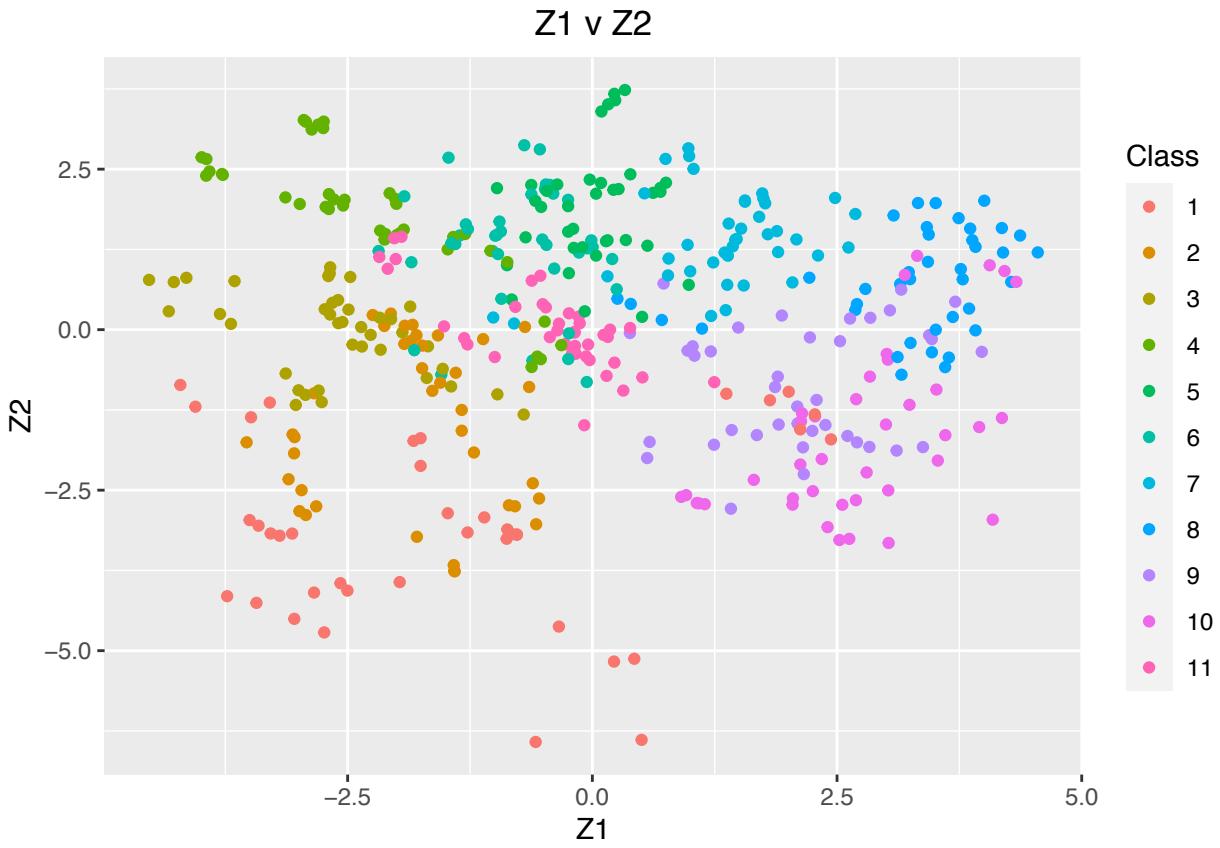
```
## x.2   1.2409110  0.52457248 -0.36435191  0.87603123  0.31716963  0.9903953
## x.3   0.4272390 -0.31739722  0.83999762 -0.09125951  0.14636292  0.3066587
## x.4   0.0734628 -0.44786181  0.23021341  0.69033947 -0.90733792  0.2570911
## x.5   0.1478910 -1.18612126 -1.01965461  0.72762913 -0.04873766  0.2300045
## x.6   0.5651243 -0.87101032  0.47812203 -0.14134249  0.78152866  0.9959552
## x.7   0.4892662 -0.32595923  0.60996336  0.59036772 -0.56552050  0.5312389
## x.8   0.8497288 -0.47631685 -0.60469318 -0.54253360 -0.58814104 -0.1809141
## x.9   0.6174759 -0.28444011 -0.44853898  0.28235855  0.08871808  0.1029527
## x.10  0.1649044 -0.07092484 -0.45461185 -0.20439454 -0.35866427  1.0804045
##            LD7        LD8         LD9         LD10
## x.1  1.5125237 -0.6784176 -0.45912875 -0.250019896
## x.2  1.2393055 -1.0443344 -0.87040601 -0.219397537
## x.3  1.2951403 -0.1111326 -0.59554521  0.276462199
## x.4  0.4742353 -0.8348347 -0.34354136 -0.390615665
## x.5  1.2443015 -0.2190774  0.04807678 -0.144757183
## x.6  0.5513275 -0.3992179 -0.51768219 -0.333332916
## x.7  0.6370130 -0.1087823  0.65021892 -0.102146350
## x.8  0.3895243 -0.2354673  0.64370081  0.005928569
## x.9  0.5074987  0.6847025  0.09952502 -0.786162108
## x.10 0.1218485  0.1833465  0.06616123  0.056449985
```

Part d

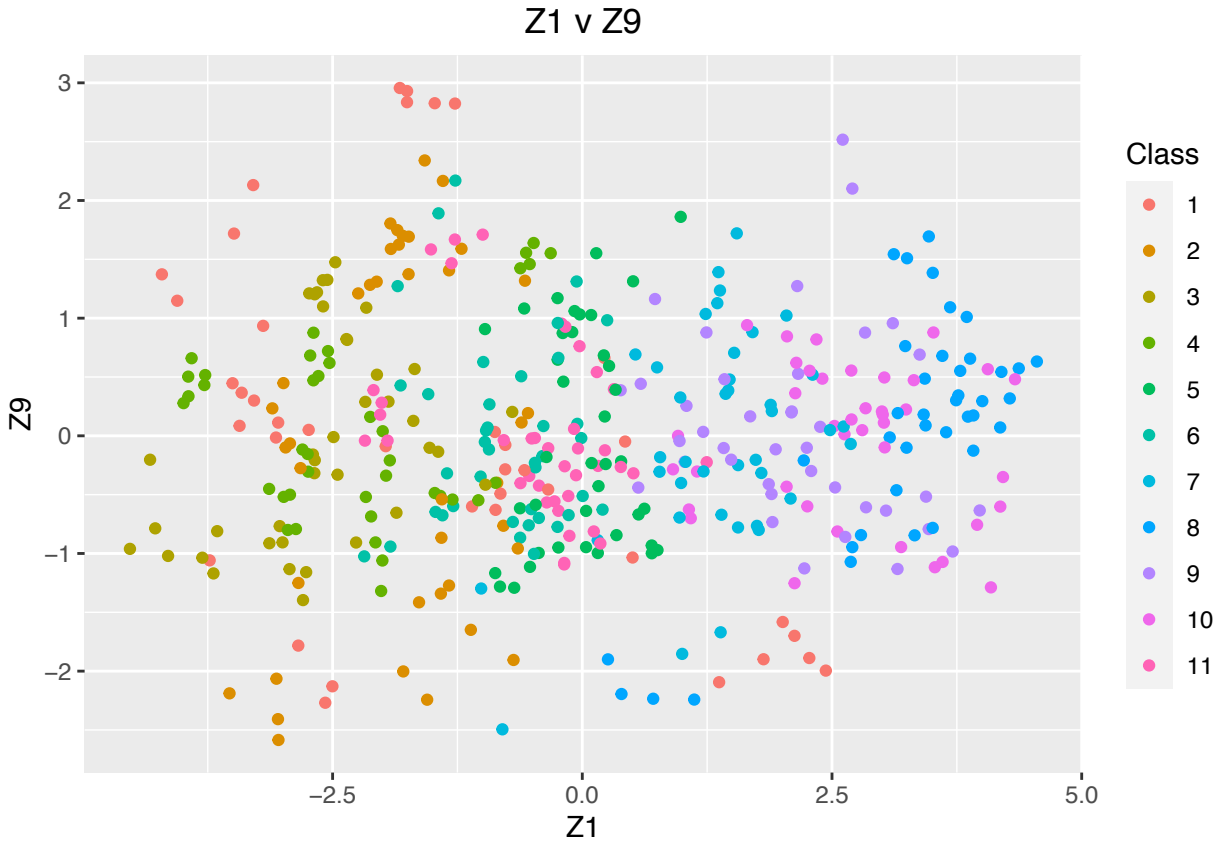
Below is the process for finding the values of the new variables z. The plots for z1 v z2 and z1 v z9 are below the code.

```
v <- vowel.lda$scaling
X <- train.transformed[-1]
z <- as.matrix(X) %*% as.matrix(v)
zplots <- as.data.frame(cbind(train.transformed$y,z))
zplots$V1 <- as.factor(zplots$V1)
```

In the z1 v z2 plot the samples from each class do seem to be contained and most observations are close to those of the same class. There is obviously some overlap since we have a large amount of classes.



The sepearation between the points in the z1vz9 plot are not as obvious as in the previous. There is more mixing between class observations.



Part e

The code below shows how lda was performed on the reduced space using z1 or z1 and z2 as inputs. Only the score calculation for class one is shown on this document in order to save space, but these were calcualted for all 11 classes.

```
# calculate pi_k
prior = table(train.transformed[, 5])/length(train.transformed[, 5])
# calcuate class centroids M
M <- aggregate(train.transformed[, -1], list(train.transformed$y),
               mean)
M <- as.matrix(M[, -1])
# variance-covariance estimation
Sigma <- cov(train.transformed[, -1] - M[as.factor(train.transformed$y),])
Sigma.inver <- solve(Sigma)

# linear discriminant functions
x.test <- as.matrix(test.transformed[, -1])
dis.func <- function(x, Sigma.inver, centroid, prior) {
  -0.5 * t(x - centroid) %*% Sigma.inver %*% (x - centroid) + log(prior)
}

scaling <- as.matrix(v[,1])
X.s <- as.matrix(train.transformed[,-1]) %*% scaling
M.s <- M %*% scaling
```

```r
# conduct lda based on the reduce space for z1
X.s.test <- as.matrix(test.transformed[,-1]) %*% scaling
Sigma.s <- cov(X.s - M.s[as.factor(train.transformed$y),])
Sigma.s.inver <- solve(Sigma.s)

score1 <- apply(X.s.test, 1, dis.func, Sigma.inver = Sigma.s.inver,
                centroid = M.s[1, ], prior = prior[1])

pred.indexz1 <- apply(cbind(score1, score2, score3,score4,score5,
                            score6,score7,score8,score9,score10,score11), 1, which.max)
sum(pred.indexz1==test.transformed$y)/nrow(test.transformed)


# conduct lda based on the reduce space for z1 and z2
scaling <- as.matrix(v[,c(1,2)])
X.s <- as.matrix(train.transformed[,-1]) %*% scaling
M.s <- M %*% scaling

X.s.test <- as.matrix(test.transformed[,-1]) %*% scaling
Sigma.s <- cov(X.s - M.s[as.factor(train.transformed$y),])
Sigma.s.inver <- solve(Sigma.s)

score1 <- apply(X.s.test, 1, dis.func, Sigma.inver = Sigma.s.inver,
                centroid = M.s[1, ], prior = prior[1])
pred.indexz1 <- apply(cbind(score1, score2, score3,score4,score5,
                            score6,score7,score8,score9,score10,score11), 1, which.max)
sum(pred.indexz1==test.transformed$y)/nrow(test.transformed)
```

The error rate for the reduced model using only z1 as an input is shown below.

```
## [1] 0.5959596
```

The error rate for the reduced model using z1 and z2 as inputs is shown below.

```
## [1] 0.2929293
```

Part f

Below is the code for doing lda using the complete model.

```r
# calculate pi_k
prior = table(train.transformed[, 5])/length(train.transformed[, 5])
# calcuate class centroids M
M <- aggregate(train.transformed[, -1], list(train.transformed$y),
               mean)
M <- as.matrix(M[, -1])
# variance-covariance estimation
Sigma <- cov(train.transformed[, -1] - M[as.factor(train.transformed$y),])
Sigma.inver <- solve(Sigma)

# linear discriminant functions
```

```
x.test <- as.matrix(test.transformed[, -1])
dis.func <- function(x, Sigma.inver, centroid, prior) {
  -0.5 * t(x - centroid) %*% Sigma.inver %*% (x - centroid) + log(prior)
}

score1 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[1, ], prior = prior[1])
score2 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[2, ], prior = prior[2])
score3 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[3, ], prior = prior[3])
score4 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[4, ], prior = prior[4])
score5 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[5, ], prior = prior[5])
score6 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[6, ], prior = prior[6])
score7 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[7, ], prior = prior[7])
score8 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[8, ], prior = prior[8])
score9 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                centroid = M[9, ], prior = prior[9])
score10 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                 centroid = M[10, ], prior = prior[10])
score11 <- apply(x.test, 1, dis.func, Sigma.inver = Sigma.inver,
                 centroid = M[11, ], prior = prior[11])

pred.index <- apply(cbind(score1, score2, score3,score4,score5,score6,
                          score7,score8,score9,score10,score11), 1, which.max)
error.full <- 1-sum(pred.index==test.transformed$y)/nrow(test.transformed)
```

The error rate when using X as the model input is shown below. The results are fairly similar to only using z1 as the input and performs similarly to using z1 and z2 as the input.

```
## [1] 0.3636364
```

Question 2

Part a

Here we load in the data and split into training and testing sets.

```
credit <- read.csv("creditcard.csv")
credit.train.index <- sort(sample(1:nrow(credit),nrow(credit)*.8))
set.seed(1)
credit.train <- credit[credit.train.index,]
credit.test <- credit[-credit.train.index,]
```

Part b

Below we fit a logistical regression model and print the results to view the p-values. Very few of the variables were found to be significant given the p-values.

```
credit.glm <- glm(Class~.,data=credit.train,family="binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

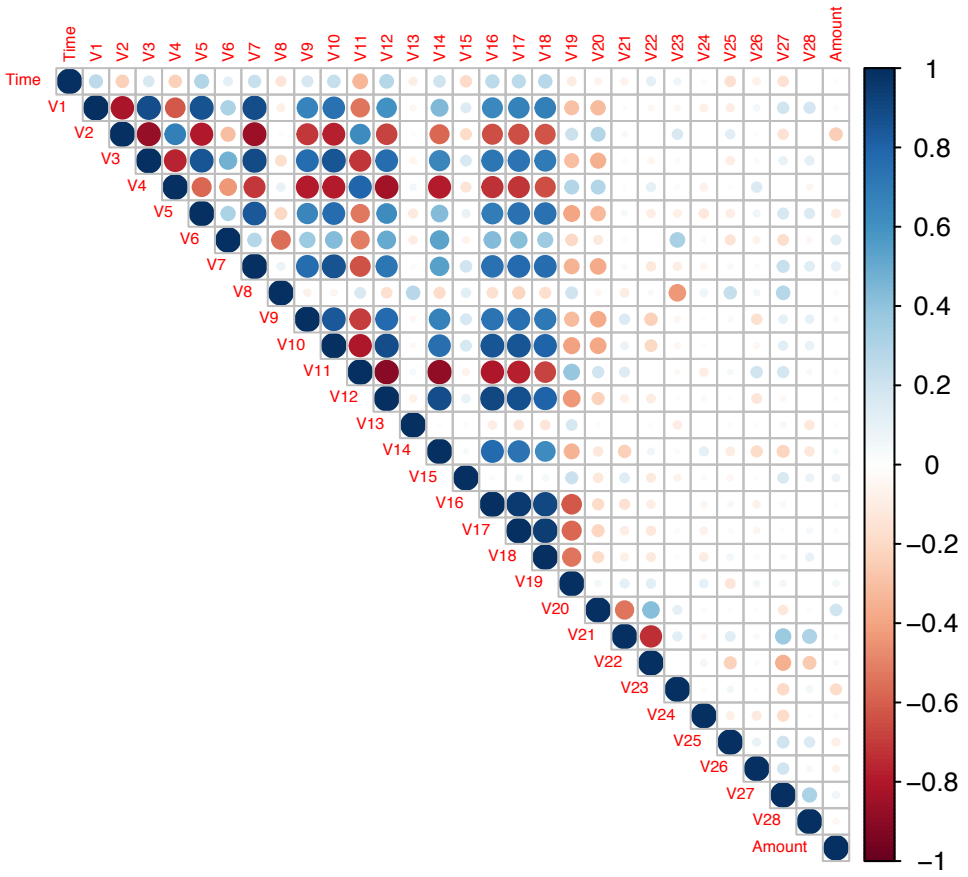
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = credit.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9291  -0.1211   0.0000   0.0000   3.2263
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.469e+01  1.177e+02  -0.125   0.9007
## Time        -2.144e-05  9.639e-06  -2.225   0.0261 *
## V1          -1.157e+01  2.665e+01  -0.434   0.6642
## V2           4.000e+01  1.733e+02   0.231   0.8175
## V3          -4.744e+01  6.873e+01  -0.690   0.4901
## V4           3.243e+01  5.538e+01   0.586   0.5581
## V5          -2.447e+01  1.348e+01  -1.816   0.0694 .
## V6          -2.064e+01  7.881e+01  -0.262   0.7934
## V7          -8.967e+01  2.717e+02  -0.330   0.7414
## V8           1.595e+01  4.622e+01   0.345   0.7300
## V9          -3.825e+01  8.324e+01  -0.460   0.6459
## V10         -8.814e+01  1.914e+02  -0.461   0.6452
## V11          6.571e+01  1.622e+02   0.405   0.6853
## V12         -1.176e+02  2.912e+02  -0.404   0.6864
## V13         -5.906e-01  7.674e+00  -0.077   0.9387
## V14         -1.245e+02  3.177e+02  -0.392   0.6952
## V15         -4.043e+00  1.133e+01  -0.357   0.7212
## V16         -1.100e+02  2.803e+02  -0.392   0.6948
## V17         -1.983e+02  4.921e+02  -0.403   0.6870
```

```
## V18         -7.453e+01  1.880e+02  -0.396   0.6919
## V19          2.717e+01  7.789e+01   0.349   0.7272
## V20         -2.423e+00  5.300e+01  -0.046   0.9635
## V21          1.194e+01  1.334e+01   0.894   0.3711
## V22          7.122e+00  3.447e+01   0.207   0.8363
## V23          1.441e+01  1.042e+02   0.138   0.8901
## V24         -2.184e+00  9.936e+00  -0.220   0.8261
## V25          9.561e+00  4.746e+01   0.201   0.8403
## V26          1.518e+00  1.196e+01   0.127   0.8990
## V27          1.751e+01  3.757e+01   0.466   0.6412
## V28          3.020e+01  1.262e+02   0.239   0.8109
## Amount       1.815e-01  1.204e+00   0.151   0.8802
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1099.33  on 792  degrees of freedom
## Residual deviance:  130.99  on 762  degrees of freedom
## AIC: 192.99
##
## Number of Fisher Scoring iterations: 25
```

Part c

A correlation plot of all predictor variables is shown below. There are a significant amount of variables that have a correlation coefficient between 0.7 and 0.9 which is worrying.



Part d

Here we subset the data with the predictor variables given in the question and again fit the losgistic regression. The output from this regression is printed below the code.

```
credit.sub <- credit.train[,c(1,3,4,6,8,13,15,19,20,21,24,25,26,27,28,31)]

credit.sub.glm <- glm(Class~.,data=credit.sub, family="binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(credit.sub.glm)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = credit.sub)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0097  -0.3035   0.0000   0.0099   3.3895
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.741e+00  4.247e-01  -4.099 4.15e-05 ***
```

```
## Time        -1.210e-05  4.180e-06  -2.896  0.00378 **
## V2          -2.587e-01  9.635e-02  -2.685  0.00725 **
## V3          -3.185e-01  1.163e-01  -2.740  0.00615 **
## V5           3.104e-01  1.089e-01   2.849  0.00438 **
## V7          -1.194e-02  9.733e-02  -0.123  0.90232
## V12         -1.005e+00  1.668e-01  -6.024 1.71e-09 ***
## V14         -1.022e+00  1.444e-01  -7.082 1.42e-12 ***
## V18          1.734e-01  1.865e-01   0.930  0.35240
## V19         -2.354e-01  1.772e-01  -1.328  0.18417
## V20         -9.310e-03  1.909e-01  -0.049  0.96110
## V23         -5.142e-02  1.515e-01  -0.339  0.73426
## V24          1.129e-01  2.732e-01   0.413  0.67939
## V25         -2.027e-01  3.385e-01  -0.599  0.54936
## V26         -6.684e-01  4.085e-01  -1.636  0.10179
## V27         -1.131e-01  3.522e-01  -0.321  0.74808
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1099.33  on 792  degrees of freedom
## Residual deviance:  257.16  on 777  degrees of freedom
## AIC: 289.16
##
## Number of Fisher Scoring iterations: 9
```

Part e

i The first part of e requires that we calcualte the error rate, true positive rate, and false positive rate at the I>0.5 cutoff. The values of these variables are printed below the code.

```
credit.test.sub <- credit.test[,c(1,3,4,6,8,13,15,19,20,21,24,25,26,27,28,31)]
rownames(credit.test.sub) <- NULL
credit.pred <- predict(credit.sub.glm,credit.test.sub,type="response")
credit.pred <- 1 *(credit.pred > 0.5)
error.credit <- mean(credit.pred != credit.test.sub$Class)

credit.pred2 <- prediction(credit.pred,credit.test.sub$Class)
credit.perf <- performance(credit.pred2,"tpr","fpr")
str(credit.perf)
fpr <- unlist(credit.perf@x.values)[2]
tpr <- unlist(credit.perf@y.values)[2]
```

Misclassification rate

```
error.credit
```

```
## [1] 0.120603
```

True positive rate

```
tpr
```

```
## [1] 0.8105263
```

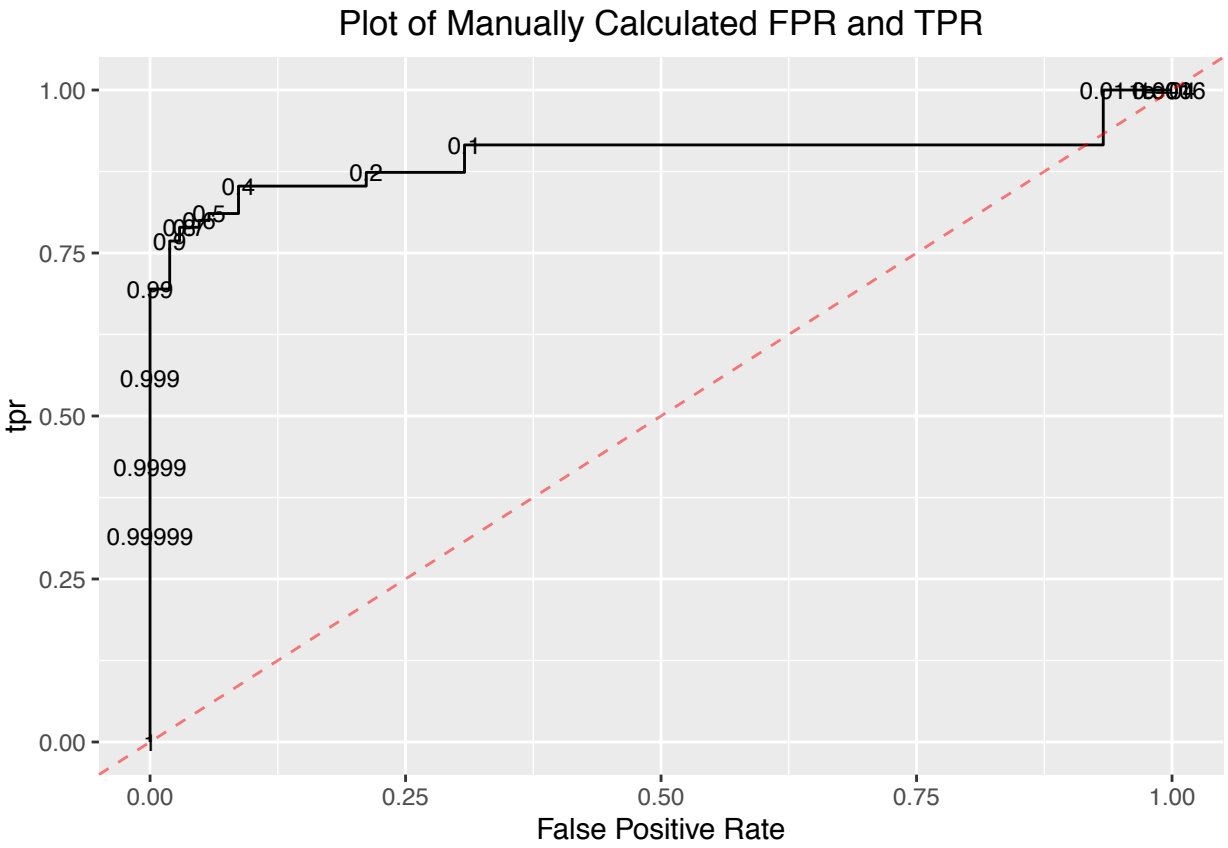
False postive rate

```
fpr
```

```
## [1] 0.05769231
```

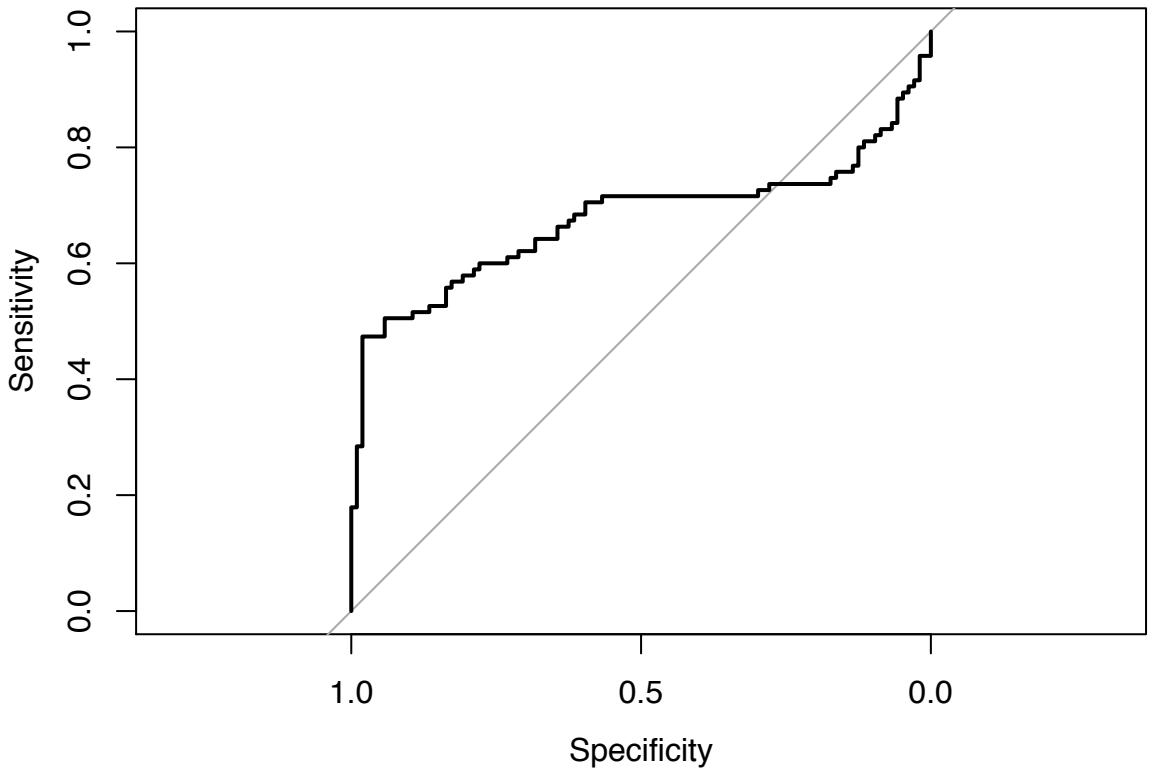
ii Part two asks us to calculate the true positive rate and false positive rate at a variety of differnt cutoffs and then plot the data.

```
C = c(0.000001, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 0.999, 0.9999, 0.999
tprs <- rep(0,length(C))
fprs <- rep(0,length(C))
for (i in 1:length(C)){
  loop.pred <- predict(credit.sub.glm,credit.test.sub,type="response")
  loop.pred <- 1 *(loop.pred > C[i])
  loop.pred2 <- prediction(loop.pred,credit.test.sub$Class)
  loop.perf <- performance(loop.pred2,"tpr","fpr")
  fprs[i] <- unlist(loop.perf@x.values)[2]
  tprs[i] <- unlist(loop.perf@y.values)[2]
}

C2 <- c(0,C,1)
fprs <- c(1,fprs,0)
tprs <- c(1,tprs,0)
rates <- data.frame(C=C2,fpr=fprs,tpr=tprs)
rates <- rates[order(rates$C,decreasing = TRUE),]
```



iii The plot below shows the roc curve and area under the curve using the roc() function from pROC. the area under the curve seems much lower using this method than the one used in the previous part.

```
## Area under the curve: 0.6754
```



Part f

Below is the process for using glmnet to fit a logistic regression with lasso penalty. The misclassification error rate is printed below as well.

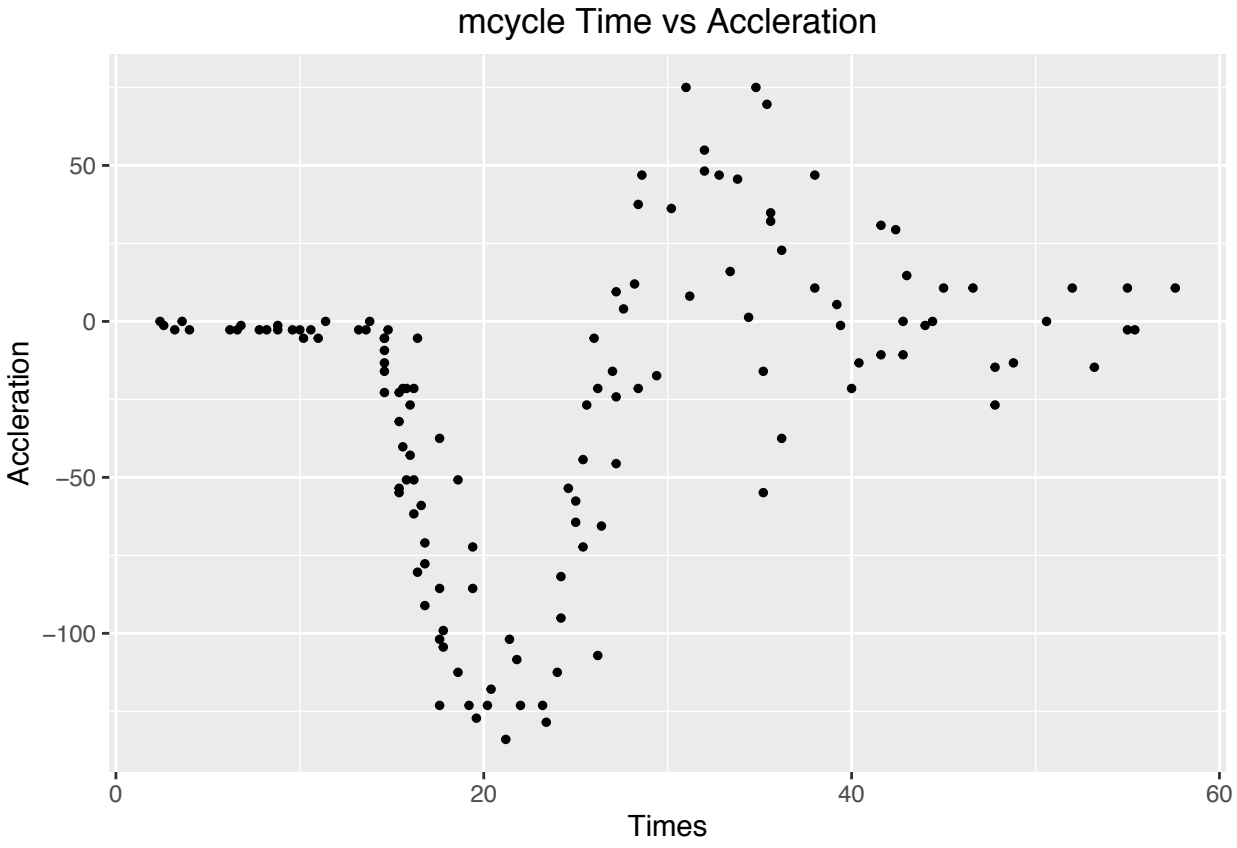
```
X <- model.matrix(Class~.,credit.sub)[,-1]
Y <- credit.sub$Class

credit.cv.lasso <- cv.glmnet(X,Y, family = "binomial")
best.lam <- credit.cv.lasso$lambda.min

credit.lasso.fit <- glmnet(X,Y,family="binomial",lambda=best.lam)
newX <- model.matrix(Class~.,credit.test.sub)[,-1]

pred.glm.lasso <- predict(credit.lasso.fit, newx =  newX, type = "response")

pred.glm.lasso <- 1 * (pred.glm.lasso > 0.5)

error.glm.lasso <- 1- mean(pred.glm.lasso ==  credit.test.sub$Class)
```

```
## [1] 0.1105528
```

Question 3

Part a

Below we have a scatterplot with time on the x axis and accel on the yaxis from the mcycle data set.

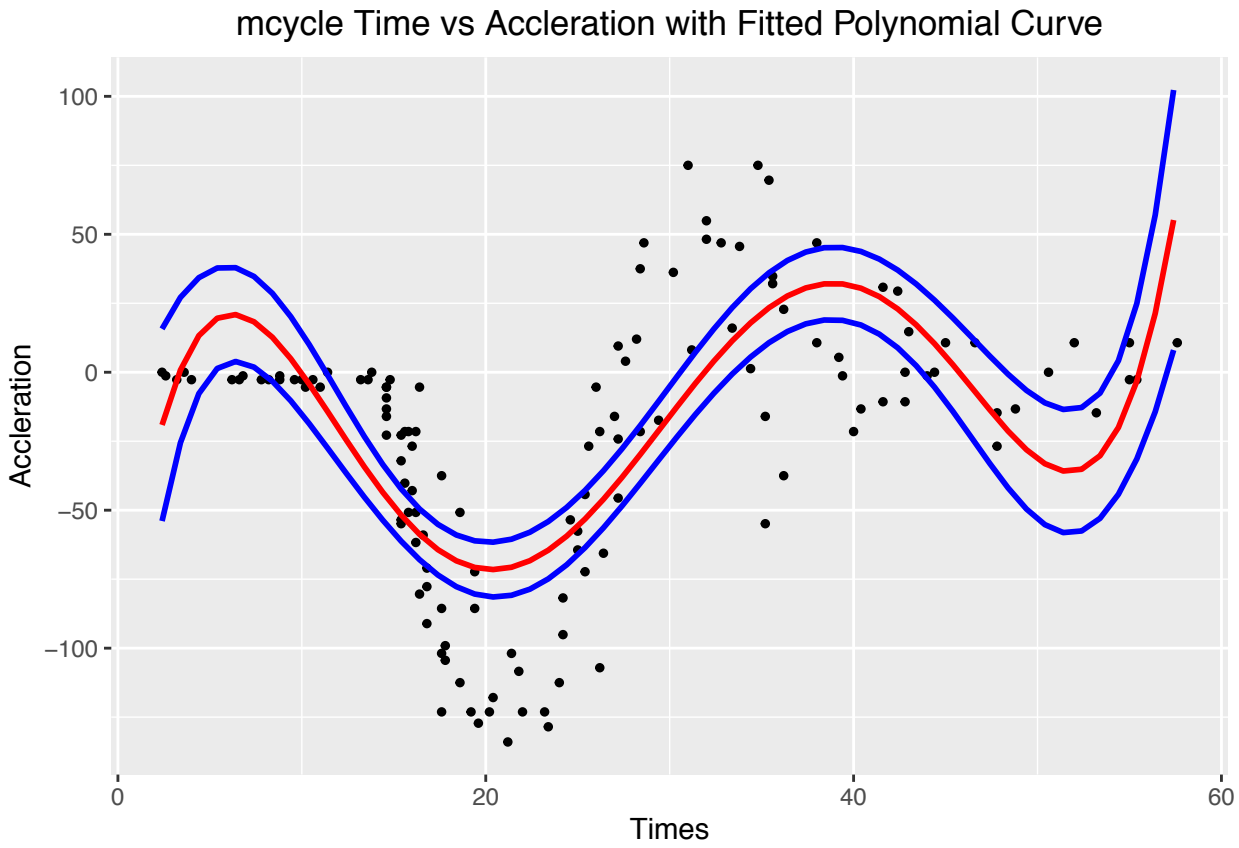


Part b

The code below shows the process of fitting a polynomial regression with d=5 to the mcycle data.

```
polyfit <- lm(accel ~ poly(times,5),data=mcycle)
summary(polyfit)

timelims <- range(mcycle$times)
times.grid <- seq(from =  timelims[1], to = timelims[2])
preds <- predict(polyfit, newdata = list(times = times.grid),se = TRUE)
se.bands <- cbind(preds$fit + (2 * preds$se),preds$fit - (2 * preds$se))
dat.fitted <- data.frame(time= times.grid, fitted = preds$fit,
                         se.bands.upper = se.bands[,1],se.bands.lower = se.bands[,2])
```

The plot below shows the fitted curve from polynomial regression as the red line and its confidence bands as the blue.



Part c

i The code below shows how we find the time values for the given quantiles to be used as knots. Their values are also printed.

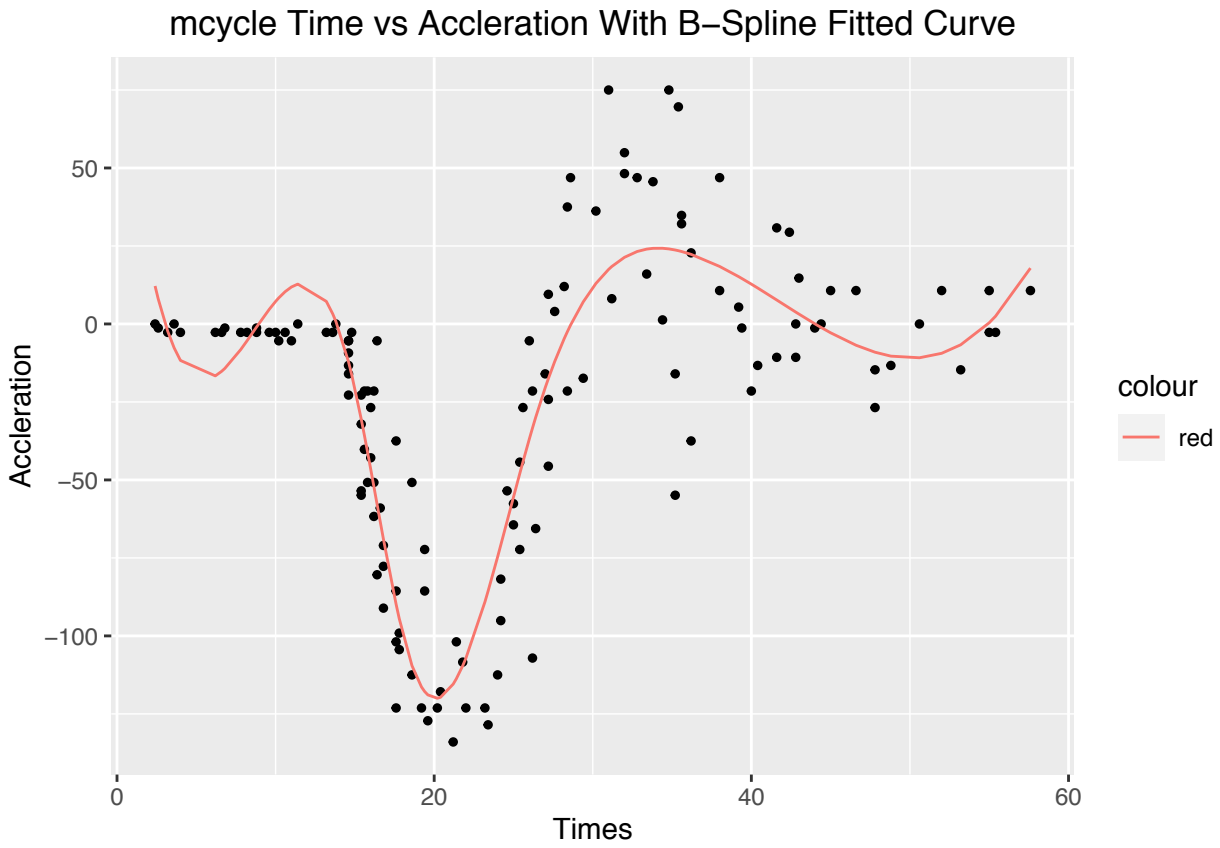
```
knots <- quantile(mcycle$time, c(.2,.4,.6,.8))
```

```
##   20%   40%   60%   80%
## 14.68 18.44 26.52 36.20
```

ii This codes shows how we conduct a cubic spline with the interior knots we just calculated.

```
basis <- bSpline(mcycle$time, knots=knots, degree=3, intercept=TRUE)
basis2 <- data.frame(mcycle$times,basis)
colnames(basis2) <- c("times", paste0("basis", 1:8))
basis2 <- basis2 %>% gather(key="basis",value="value", -times)

bfit <- lm(mcycle$accel ~ 0 + bSpline(mcycle$times,knots=knots,degree=3,intercept=TRUE))

mcycle2 <- cbind(mcycle,t(bfit$coefficients %*% t(basis)))
colnames(mcycle2) <- c("times","accel","fitted")
```

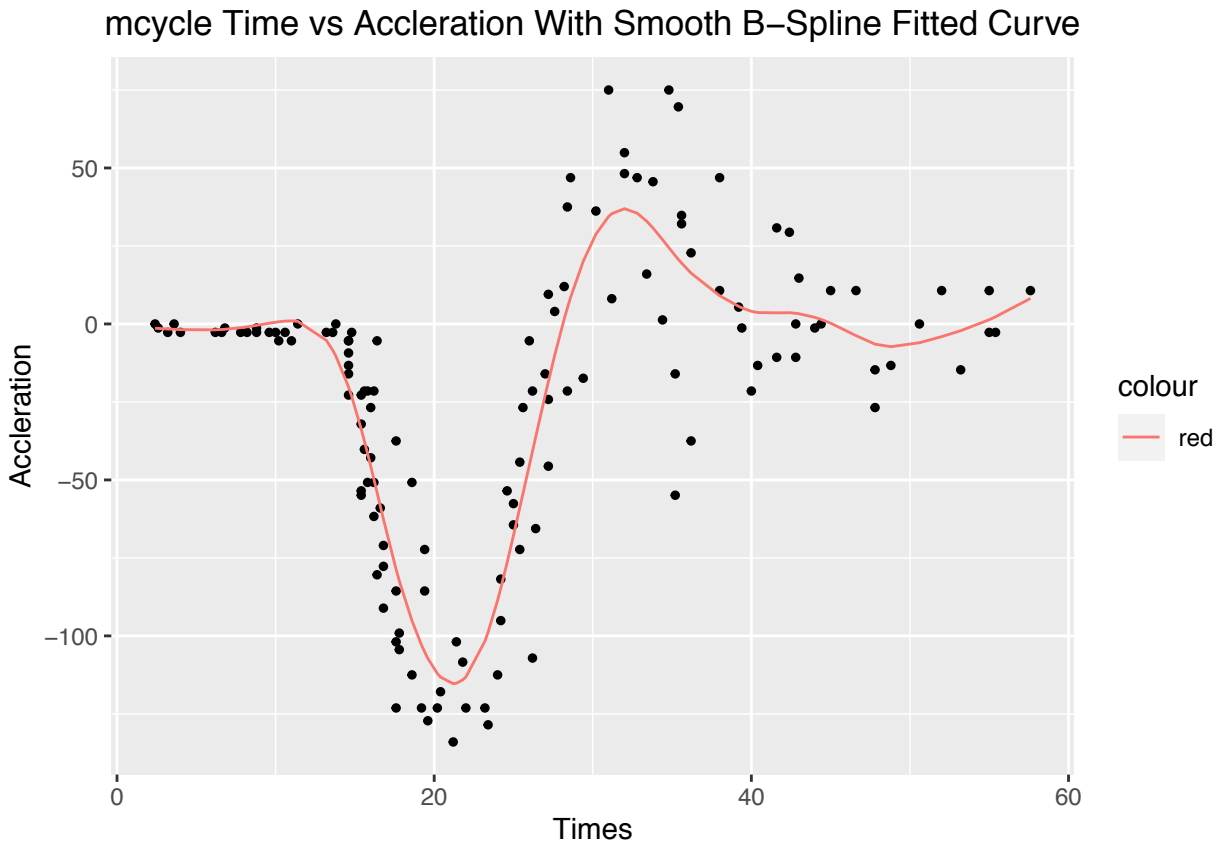
iii Below is the plot of the cubic spline plotted over the scatterplot from part a.



Part d

This code shows how to find a smooth spline and it is overlayed on the scatterplot from part a below.

```
smooth.bfit <- smooth.spline(mcycle$times,mcycle$accel)
mcycle3 <- as.data.frame(cbind(smooth.bfit$x,smooth.bfit$y))
colnames(mcycle3) <- c("times","fitted")
```



The degrees of freedom used in the smoothed spline is shown below.

```
smooth.bfit$df
```

```
## [1] 12.20876
```

Question 4

Part a

The code below shows how I implemented the Nadaraya-Watson estimator to the mcycle dataset with $\lambda = 5$.

```
data(mcycle)
cycle <- mcycle

#a
nw <- function(x,X,Y,h){
  K<-rep(0,length(X))
  for (i in 1:length(X)){
    t <- abs(x-X[i])/h
    ifelse(abs(t)<1,Dt <- 3/4*(1-t^2),Dt <- 0)
    K[i] <- Dt/h
  }
  sum(K*Y)/sum(K)
}
```

Below is the estimated value when $x_0 = 30$.

```
nw(30,cycle$times,cycle$accel,5)
```

```
## [1] 9.476122
```

b.) $(\hat{\alpha}, \hat{\beta}) = \underset{\alpha,\beta}{\operatorname{argmin}} \sum_{i=1}^{N} k_\lambda(x, x_i)\{y_i - \alpha - \beta(x - x_i)\}^2$

$= \underset{\beta}{\arg\min} \sum_{i=1}^{N} \left(y_i - \sum_{j=0}^{p} \beta_j (x_i - x)^j\right)^2 K_\lambda(x, x_i)$

$$X = \begin{pmatrix} 1 & x_1 - x & \cdots & (x_1 - x)^p \\ \vdots & \vdots & \ddots & \\ 1 & x_n - x & \cdots & (x_n - x)^p \end{pmatrix}$$

$\beta = \arg\min (y - X\beta)' W (y - X\beta)$

$= (X^T W X)^{-1} X^T W y$

$= \sum_{i=1}^{n} w_i^p(x) y_i$ where $w_i^p = e_1^T (X^T W X)^{-1} X^T W e_i$

When $p = 1$ (local linear)

$w_i = \frac{1}{n} \frac{S_2(x;h) - S_1(x;h)(x_i - x)}{S_2(x;h) S_0(x;h) - S_1(x;h)^2} K(x - x_i)$

Part c

Below is the code I used to create my own local linear estimator function with the information found in part b. The estimate for $x_0 = 30$ is printed below the code.

```
Kest <- function(x,X,Y,h){
  K<-rep(0,length(X))
  for (i in 1:length(X)){
    t <- abs(x-X[i])/h
    ifelse(abs(t)<1,Dt <- 3/4*(1-t^2),Dt <- 0)
    K[i] <- Dt/h
  }
  return(K)
}

K <- Kest(30,mcycle$times,mcycle$accel,5)

n <- nrow(mcycle)
s0 <- (sum((mcycle$times-30)^0*K))/n
s1 <- (sum((mcycle$times-30)^1*K))/n
s2 <- (sum((mcycle$times-30)^2*K))/n

W <- ((s2-s1*(mcycle$times-30))/(s2*s0-s1^2))*K
W <- W/n

x30.est <- sum(W*Y)
```

```
## Warning in W * Y: longer object length is not a multiple of shorter object
## length
```

```
x30.est
```

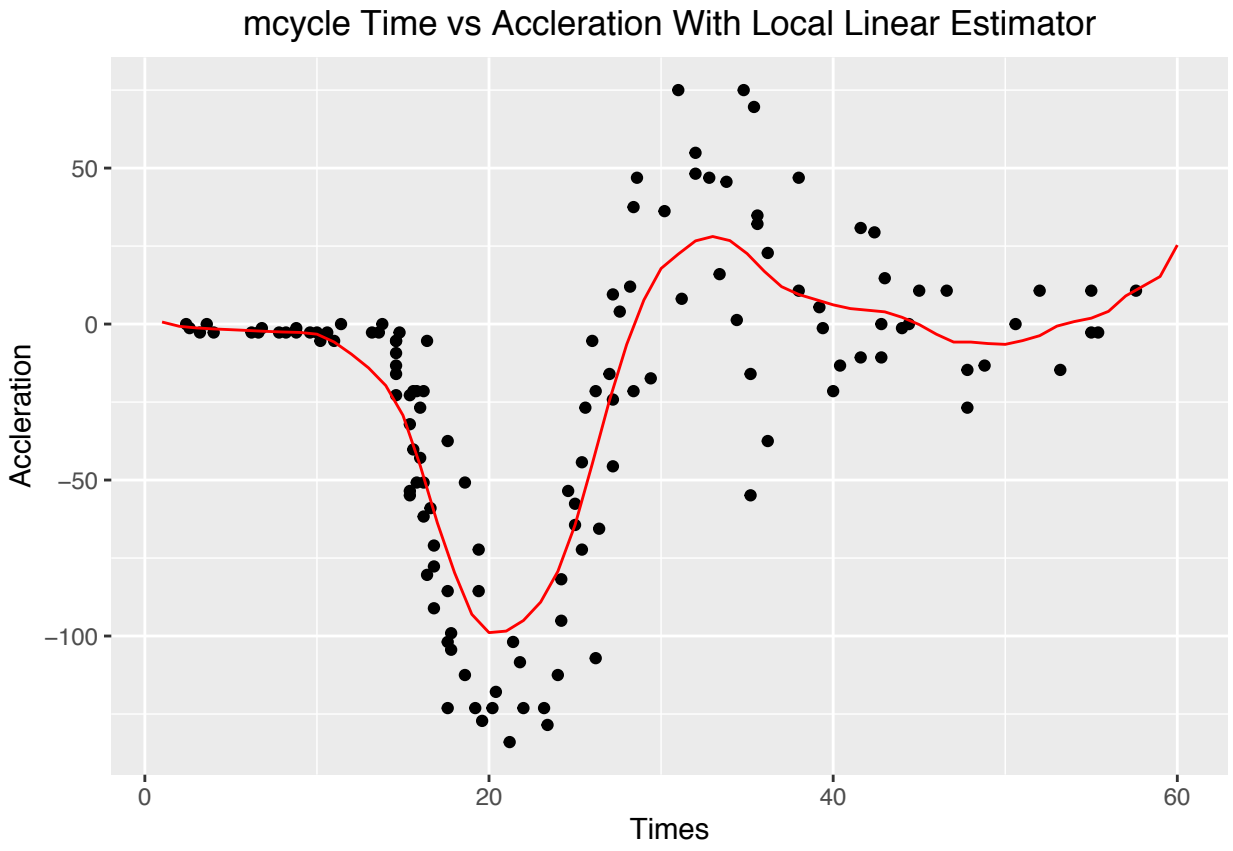
```
## [1] 3
```

Part d

This code shows the process in part c being repeated for a sequence of numbers that cover the range of the mcycle data. The final line is overlayed the scatterplot in part a.

```
Kest <- function(x,X,Y,h){
  K<-rep(0,length(X))
  for (i in 1:length(X)){
    t <- abs(x-X[i])/h
    ifelse(abs(t)<1,Dt <- 3/4*(1-t^2),Dt <- 0)
    K[i] <- Dt/h
  }
  return(K)
}

grid <- seq(1,60,by=1)
y.est <- rep(0,60)
for (i in 1:60){
```

```
  currentx <- grid[i]
  K2 <- Kest(currentx,mcycle$times,mcycle$accel,5)

  n <- nrow(mcycle)
  s0 <- (sum((mcycle$times-currentx)^0*K2))/n
  s1 <- (sum((mcycle$times-currentx)^1*K2))/n
  s2 <- (sum((mcycle$times-currentx)^2*K2))/n

  W <- ((s2-s1*(mcycle$times-currentx))/(s2*s0-s1^2))*K2
  W <- W/n

  y.est[i] <- sum(W*mcycle$accel)
}

mcycle.est <- data.frame(x1=grid,y1=y.est)
```



Question 5

Here we read in the data and turn Sales into a factor with 2 levels.

```
set.seed(430)
data(Carseats)
car <- Carseats
car$Sales = as.factor(ifelse(car$Sales <= 8, "Low", "High"))
```

Part a

This code shows the process of splitting the data into training and testing sets.

```
car.index <- sample(1:nrow(car),round(nrow(car)*.8,0))
car.train <- car[car.index,]
car.test <- car[-car.index,]
```

Part b

i The codes below shows how we can use cross validation to find the best size for the decision tree model. The best size is printed below the code.

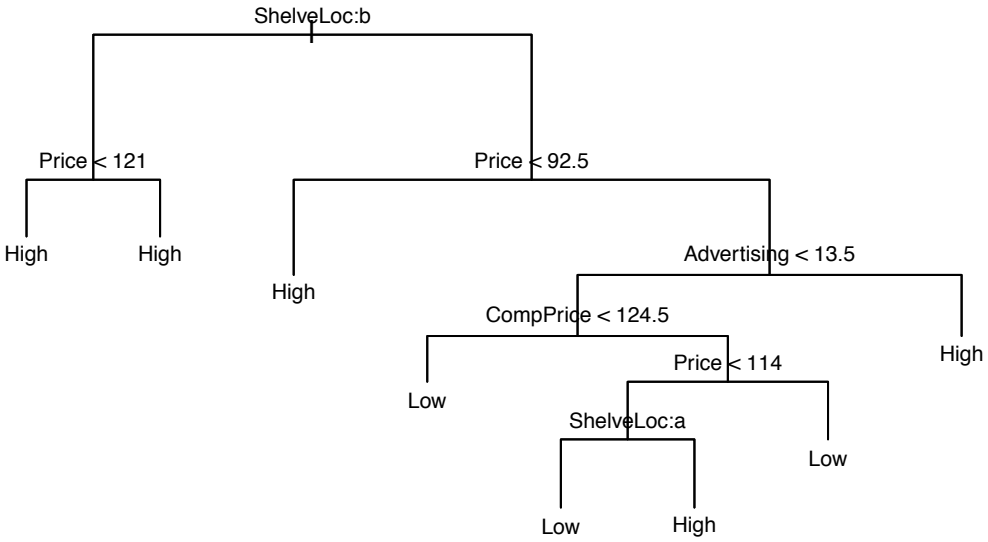
```
set.seed(1)
cartree <- tree(Sales~., data=car.train)
cv.cartree <-cv.tree(cartree,K=10)

best.size <- cv.cartree$size[which.min(cv.cartree$dev)]
```

```
## [1] 8
```

ii This code shows how the tree was pruned to fit the best size parameter.

```
cartree.best <- prune.tree(cartree,best=best.size)
```

iii Below is a plot of the fitted tree model.



iiii With this code we use the tree model to predict the sales and then find the misclassification error rate. The error rate and confusion matrix are printed below.

```
tree.predictions <- predict(cartree.best,car.test,type="class")
tree.error <- 1 - sum(tree.predictions==car.test$Sales)/nrow(car.test)
confusion.tree <-confusionMatrix(tree.predictions,car.test$Sales)
confusion.tree <- confusion.tree$table
```

```
##           Reference
## Prediction High Low
##       High   25  14
##       Low     3  38
```

```
tree.error
```

```
## [1] 0.2125
```

Part c

i This code shows the process for fitting the initial random forest model and the inputs used at each node.

```
car.rf <- randomForest(Sales~.,data=car.train)
splits <- car.rf$mtry
```

The inputs used at each node are printed below.

```
## [1] 3
```

ii This code shows the process of predicting sales using the random forest model and finding its corresponding error rate and confusion matrix.

```
forest.predictions <- predict(car.rf,car.test)
forest.error <- 1-sum(forest.predictions==car.test$Sales)/nrow(car.test)

confusion <- confusionMatrix(forest.predictions,car.test$Sales)
confusion.table <- confusion$table
```

The confusion table from estimating the data is shown below.

```
##            Reference
## Prediction High Low
##       High   24   5
##       Low     4  47
```

The misclassification error rate from the random forest model is printed below.

```
## [1] 0.1125
```

iii Below is the variable importance plot.