503 HW 3

Naomi Giertych

2/19/2018

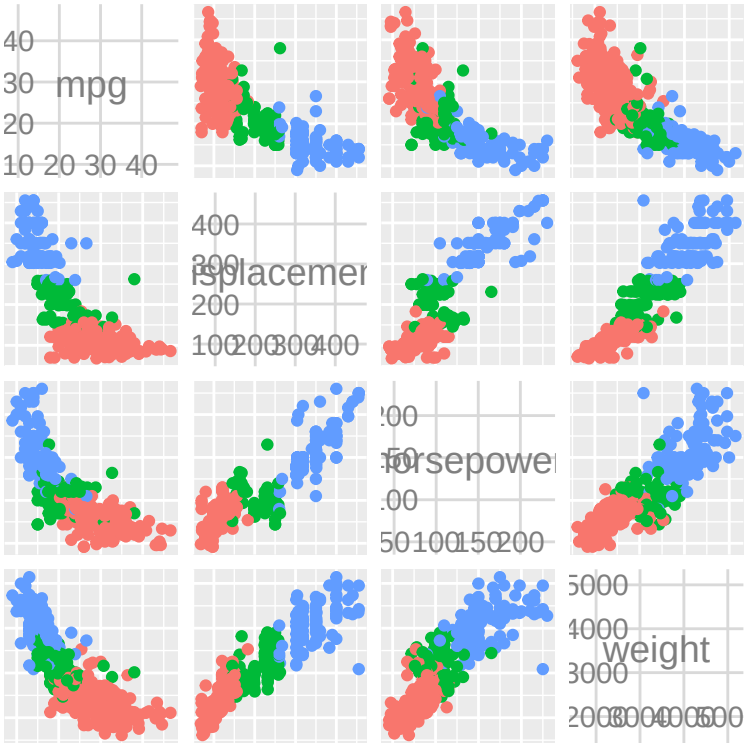
Question 2

I read in the vehical MPG dataset and cleaned it. To clean the dataset, I removed "?" marks in the "horsepower", and I converted "cylinders", "model year", and "origin" to be factors.

For this analysis, I prepared two additional versions of the original data: 1) standardized all continuous variables, 2) and principal components. Recall from HW 1, variables I included for PCA analysis were MPG, displacement, horsepower, weight and acceleration. Before performing PCA, I re-centered each of the variables to 0. In HW 1, my final PCA analysis was performed using correlation and I chose the first two principal components since they explain 92.77% of the variance in the data. I will use the dataset projected on the first two principal components from HW 1 as my PCA dataset (note I have fixed the projection so that it is projected onto the scaled matrix).

Next, I add the class label to the dataset and randomly select from each class 75% of the data to be used for training/validation purposes and the remaining 25% is left as test data. Note, I've interpreted this as meaning we want 75% of class 1, 75% of class 2, and 75% of class 3 in our training datasets (for each dataset).

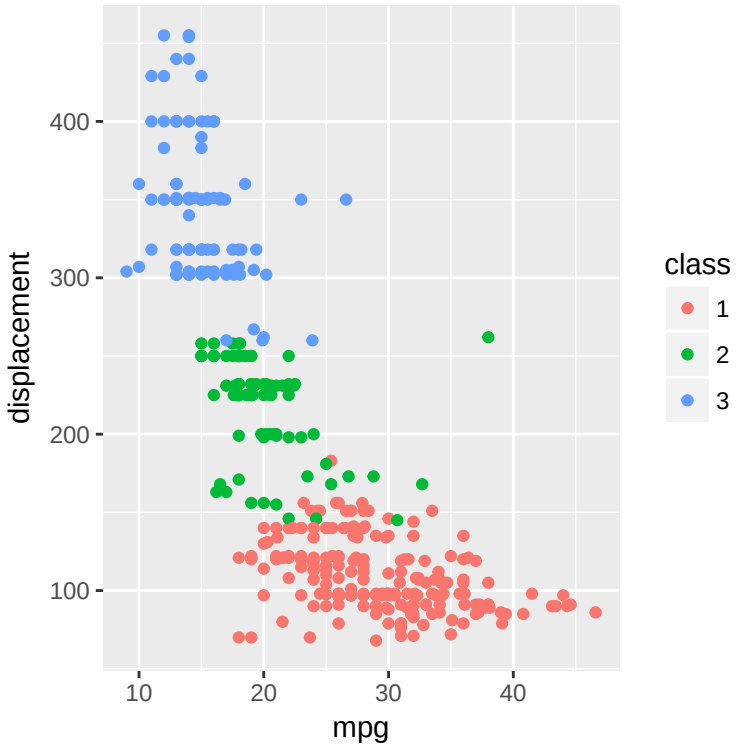
Below is a graph for all the pairwise combinations of the numerical variables that we will be using in our analysis. It is immediately noticable that displacement separates the data fairly well and that mpg might be the second best separator of the data.

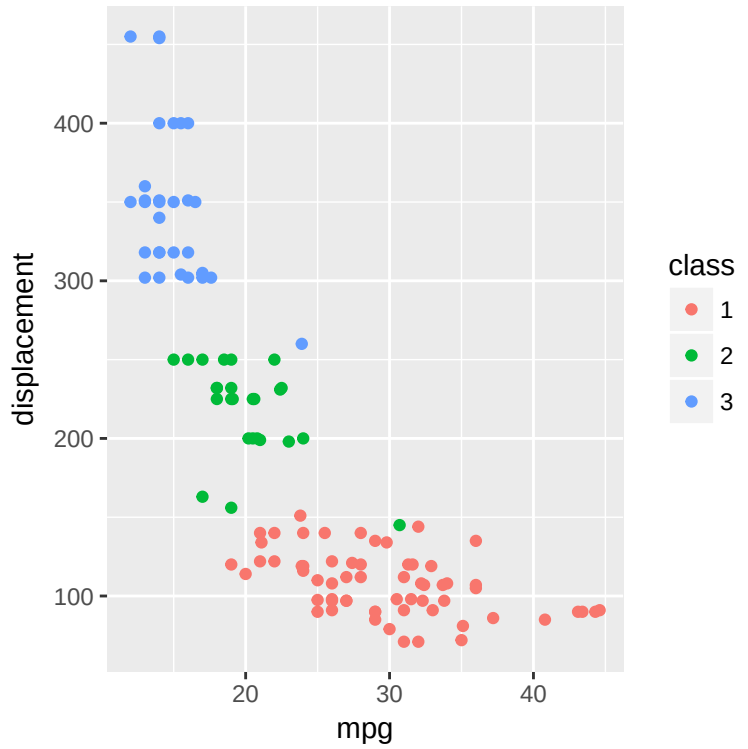


Below are summary statistics and graphs for the original dataset first by the training set and then by the test dataset.

```
##       mpg         displacement     horsepower        weight
##  Min.   : 9.00   Min.   : 68.0   Min.   : 46.0   Min.   :1649
##  1st Qu.:17.50   1st Qu.:101.8   1st Qu.: 78.0   1st Qu.:2222
##  Median :22.65   Median :151.0   Median : 95.0   Median :2822
##  Mean   :23.30   Mean   :194.6   Mean   :105.8   Mean   :2991
##  3rd Qu.:28.70   3rd Qu.:265.8   3rd Qu.:128.8   3rd Qu.:3618
##  Max.   :46.60   Max.   :455.0   Max.   :230.0   Max.   :5140
##   acceleration   class
##  Min.   : 8.50   1:144
##  1st Qu.:13.50   2: 58
##  Median :15.25   3: 72
##  Mean   :15.39
##  3rd Qu.:16.90
##  Max.   :24.80
```

```
##       mpg         displacement     horsepower        weight
##  Min.   :12.00   Min.   : 71.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:107.0   1st Qu.: 74.0   1st Qu.:2231
##  Median :22.75   Median :140.0   Median : 90.0   Median :2733
##  Mean   :23.79   Mean   :193.9   Mean   :101.4   Mean   :2947
##  3rd Qu.:29.95   3rd Qu.:291.5   3rd Qu.:123.8   3rd Qu.:3588
##  Max.   :44.60   Max.   :455.0   Max.   :225.0   Max.   :4951
##   acceleration   class
##  Min.   : 8.00   1:62
##  1st Qu.:14.00   2:25
##  Median :15.90   3:31
##  Mean   :15.90
##  3rd Qu.:17.68
##  Max.   :23.70
```



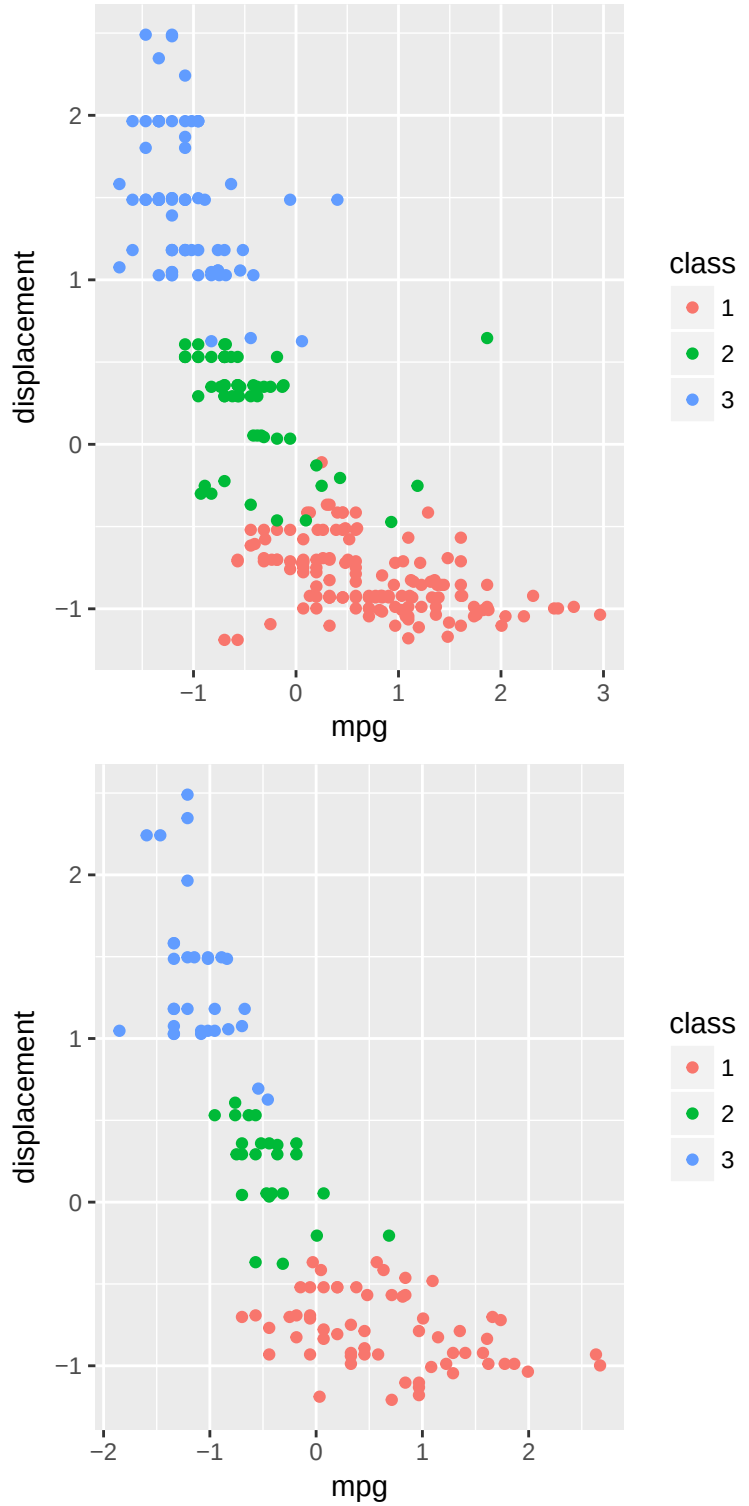


Below are summary statistics and graphs for the scaled dataset first by the training set and then by the test dataset.

```
##       mpg              displacement        horsepower
##  Min.   :-1.72273   Min.   :-1.188907   Min.   :-1.519034
##  1st Qu.:-0.82587   1st Qu.:-0.921333   1st Qu.:-0.765614
##  Median :-0.05713   Median :-0.438745   Median :-0.271995
##  Mean   : 0.01493   Mean   : 0.005148   Mean   : 0.006769
##  3rd Qu.: 0.71160   3rd Qu.: 0.932571   3rd Qu.: 0.637305
##  Max.   : 2.96657   Max.   : 2.490234   Max.   : 3.261284
##      weight           acceleration      class
##  Min.   :-1.606522   Min.   :-2.73349   1:144
##  1st Qu.:-0.924278   1st Qu.:-0.73992   2: 58
##  Median :-0.196119   Median :-0.05123   3: 72
##  Mean   :-0.001358   Mean   :-0.02027
##  3rd Qu.: 0.766911   3rd Qu.: 0.58309
##  Max.   : 2.545808   Max.   : 3.35597
```

```
##       mpg              displacement        horsepower
##  Min.   :-1.85085   Min.   :-1.20802   Min.   :-1.46707
##  1st Qu.:-0.75861   1st Qu.:-0.82099   1st Qu.:-0.55777
##  Median :-0.18526   Median :-0.41485   Median :-0.29797
##  Mean   :-0.03466   Mean   :-0.01195   Mean   :-0.01572
##  3rd Qu.: 0.67317   3rd Qu.: 0.67694   3rd Qu.: 0.37101
##  Max.   : 2.67188   Max.   : 2.49023   Max.   : 3.13138
##      weight           acceleration      class
##  Min.   :-1.564140   Min.   :-2.55226   1:62
##  1st Qu.:-0.732379   1st Qu.:-0.53150   2:25
##  Median :-0.224374   Median : 0.02127   3:31
##  Mean   : 0.003153   Mean   : 0.04707
##  3rd Qu.: 0.713638   3rd Qu.: 0.52872
```

```
##  Max.   : 2.324476   Max.   : 3.28348
```

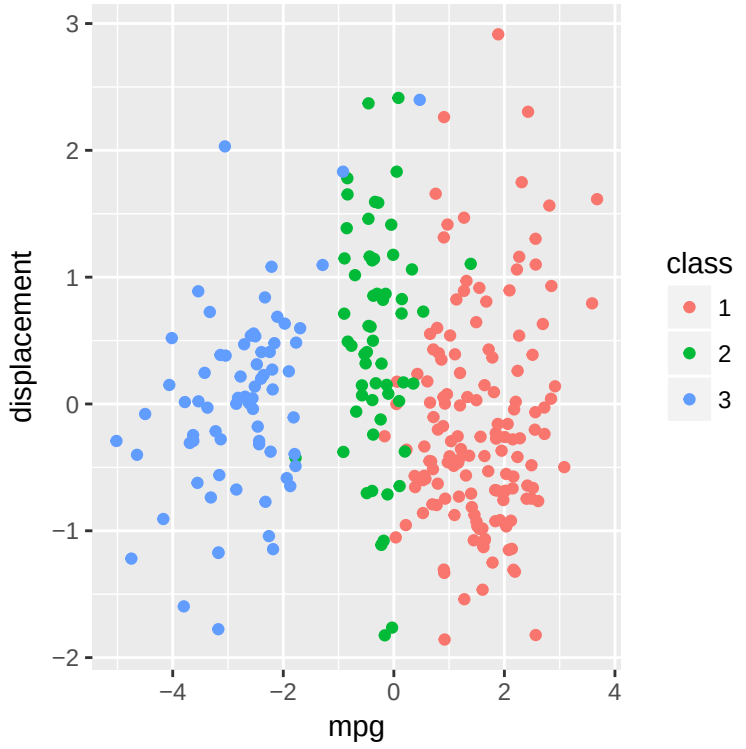


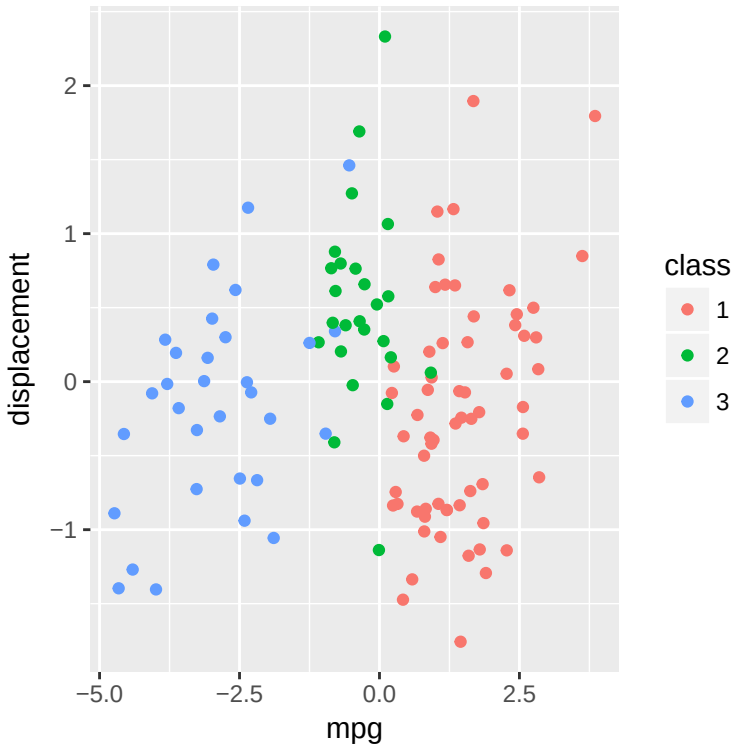
Below are summary statistics and graphs for the PCA dataset first by the training set and then by the test dataset.

```
##       mpg              displacement       class
##  Min.   :-5.01845   Min.   :-1.85732   1:144
```

```
##  1st Qu.:-1.74958   1st Qu.:-0.59806   2: 58
##  Median : 0.49771   Median :-0.01986   3: 72
##  Mean   : 0.02362   Mean   : 0.01978
##  3rd Qu.: 1.60492   3rd Qu.: 0.53725
##  Max.   : 3.67653   Max.   : 2.91490
```

```
##       mpg             displacement       class
##  Min.   :-4.73432   Min.   :-1.75699   1:62
##  1st Qu.:-0.93604   1st Qu.:-0.68571   2:25
##  Median : 0.30589   Median :-0.03966   3:31
##  Mean   :-0.05485   Mean   :-0.04594
##  3rd Qu.: 1.40909   3rd Qu.: 0.42140
##  Max.   : 3.85110   Max.   : 2.33168
```





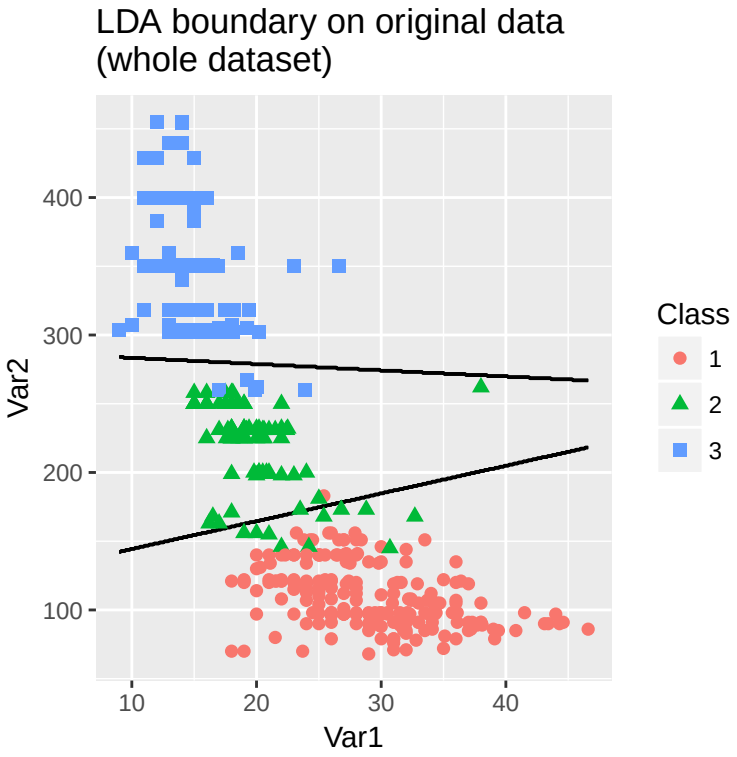
None of test sets contain significantly different information than the training datasets. There are no test datasets without a class (by construction) and it does not appear that the test datasets only contain "extreme values" as compared to the training datasets. Since, the training and test pairs appear to be compatible, we will move on to the data analysis.

First, I apply LDA and QDA to my three training datasets. LDA and QDA are very similar; they both assume that the variables given their class are multivariate normal and computationally do essentially the same thing. The main difference is that LDA assumes that the covariance matrices of the variables given their class are equal and QDA does not make that assumption.

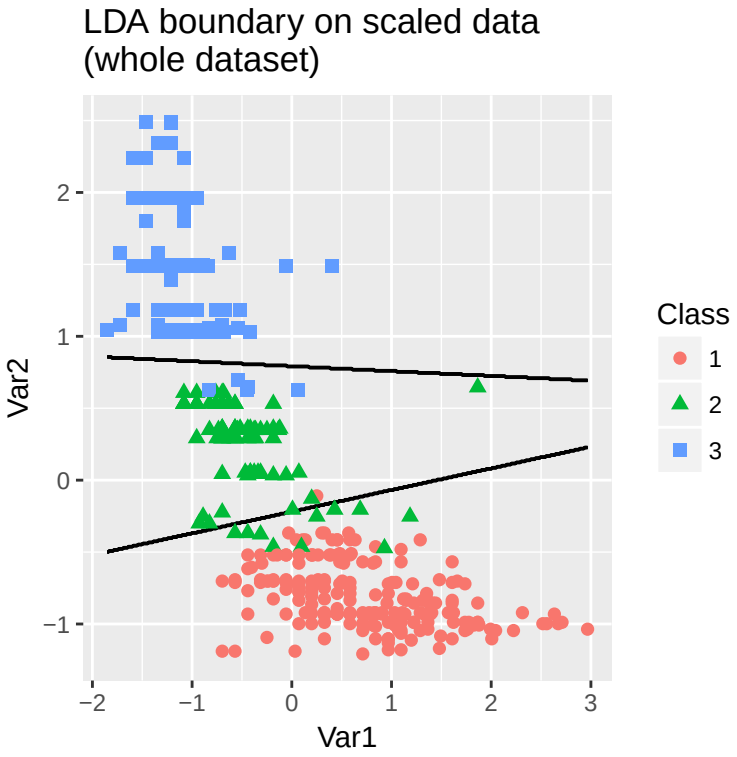
Below is a table comparing my LDA error with my QDA error for all of my test datasets. The functions were trained on the training datasets. It is interesting that the QDA error rates are consistently lower than the LDA error rates.

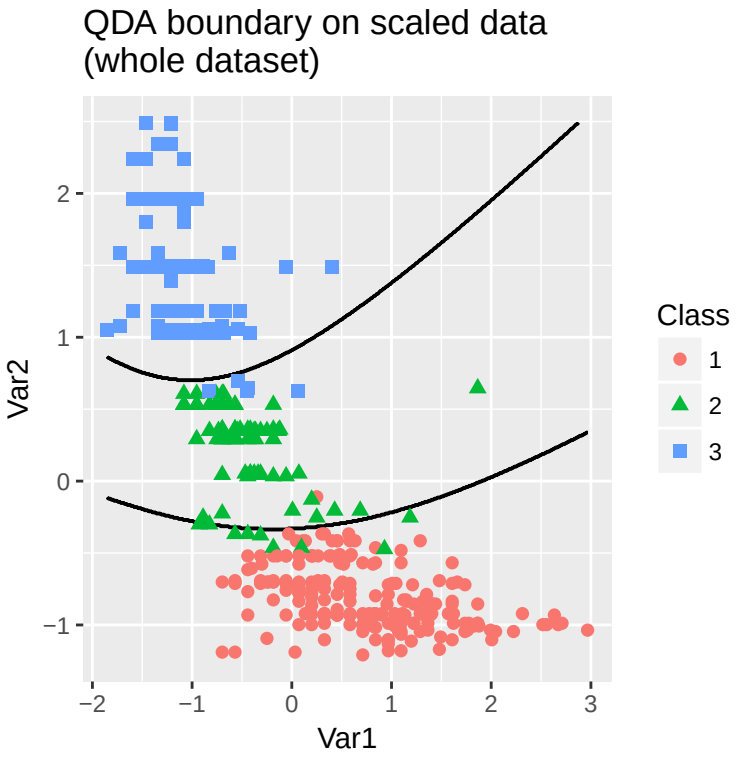
	lda error	qda error
original cars	0.0338983	0.0254237
scaled cars	0.0508475	0.0169492
pc cars	0.0677966	0.0508475

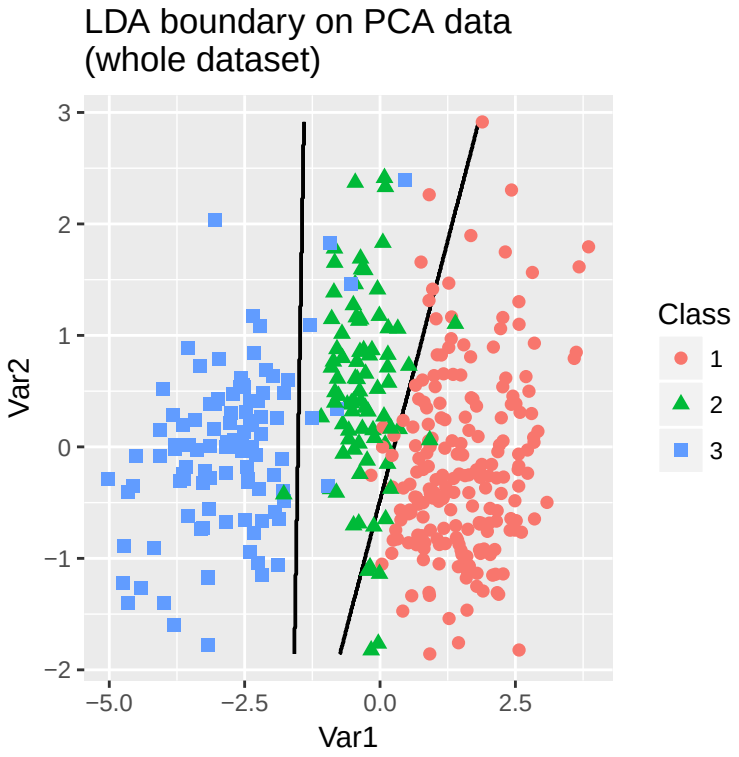
Below are the plots of my LDA and QDA analyses on the original data, standardized data, and PCA-prepossed data respectively. They all perform about the same (which we would expect from the table above). Since the LDA analysis is close to the QDA analysis in terms of error rate and captured points. I would probably stick with the LDA analysis if I had to pick one.

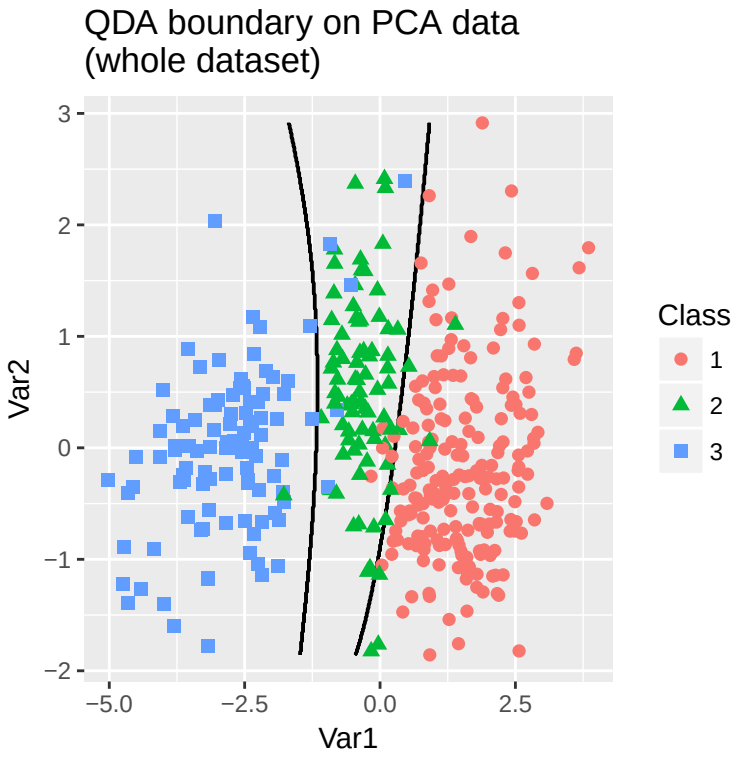

LDA boundary on original data
(whole dataset)
Var2
Var1
400
300
200
100
10
20
30
40
Class
1
2
3


QDA boundary on original data
(whole dataset)
Var2
Var1
400
300
200
100
10
20
30
40
Class
1
2
3


LDA boundary on scaled data
(whole dataset)
Var2
Var1
Class
1
2
3


QDA boundary on scaled data
(whole dataset)
Var2
Var1
Class
1
2
3





Next, I try fitting a logistic regression model to the data and see how that does in terms of separating the data. Below is a table of the training and test mean error rates for each of the pre-processed datasets. Oddly, the test dataset had lower errors than the training dataset for the original and scaled dataset.

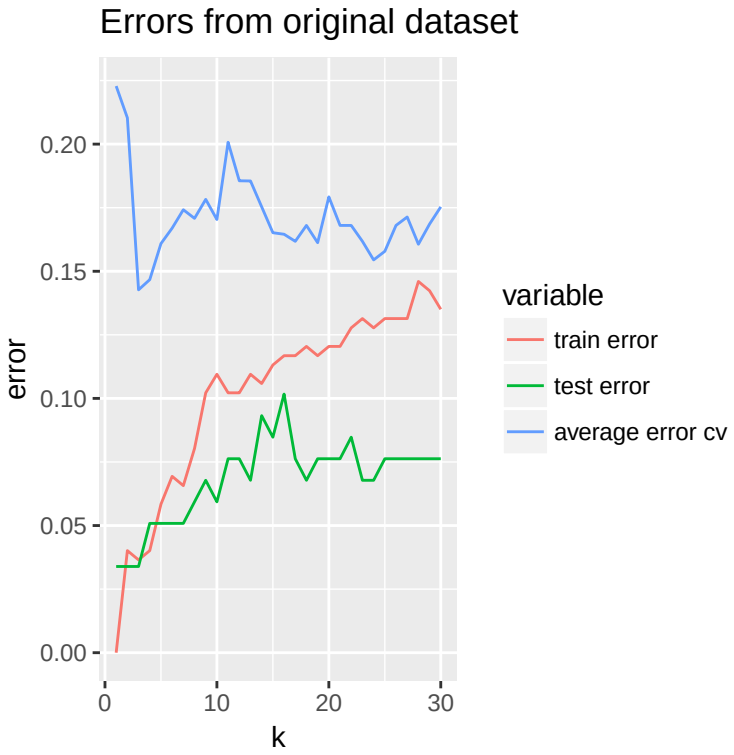
	train error	test error
original cars	0.0218978	0.0169492

	train error	test error
scaled cars	0.0218978	0.0084746
pc cars	0.0401460	0.0593220

Finally, I perform a nearest neighbor classifier to each of the three versions of the data. The distance metric is Euclidean; this makes sense since all of our variables are numeric. First I compute at the training and test errors for each dataset using anywhere from 1 to 30 nearest neighbors. These are used to get a feel for the best number of neighbors. Next, I compute the training and test errors for each dataset using cross-validation. To create the cross-validation, I randomly split the data into 5 parts, allocate one of the parts to be the "test" dataset, perform k-nearest neighbor on the other 4 "train" parts, and repeat through each subset. This is another method of determining the optimum number of neighbors and reduces any effects from extreme values being in the dataset.

Below is a table of the average cross-validation error on the test original dataset. The graph shows the error rates for the test set using the whole training data has the "test" set, using the whole test set, and the average of the cross-validated test sets.

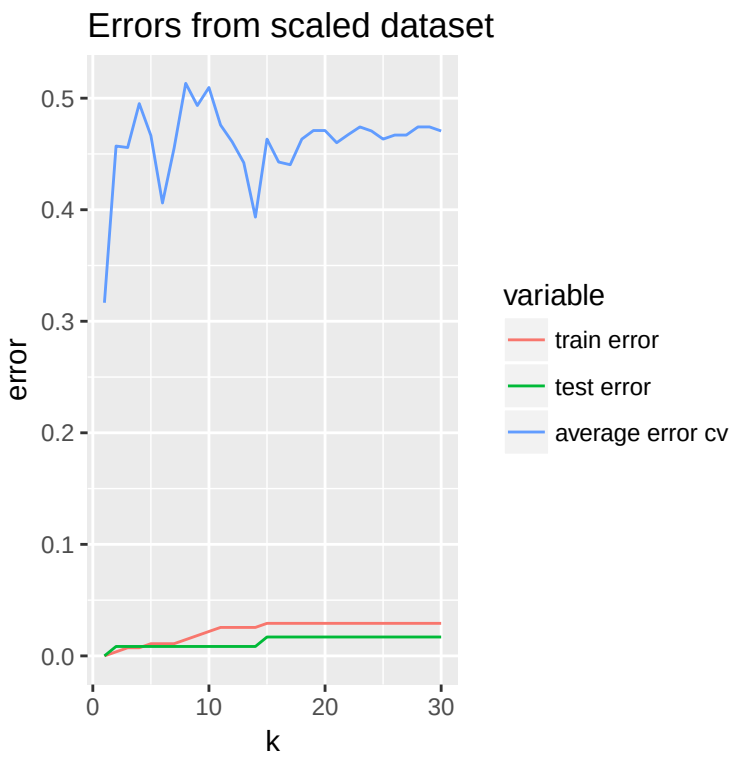
	average error cv
1	0.2228275
2	0.2103317
3	0.1426986
4	0.1466986
5	0.1608681
6	0.1669591
7	0.1741921
8	0.1708023
9	0.1782705
10	0.1703490
11	0.2007365
12	0.1855819
13	0.1855035
14	0.1753340
15	0.1651645
16	0.1645543
17	0.1617747
18	0.1680226
19	0.1612429
20	0.1792556
21	0.1680226
22	0.1680226
23	0.1617747
24	0.1544633
25	0.1578531
26	0.1680226
27	0.1713340
28	0.1606328
29	0.1685543
30	0.1753340



Below is a table of the average cross-validation error on the test scaled dataset. The graph is the same as previously but with the scaled dataset.

	average error cv
1	0.3165958
2	0.4570714
3	0.4557477
4	0.4951491
5	0.4664537
6	0.4060176
7	0.4552664
8	0.5132948
9	0.4934004
10	0.5095764
11	0.4759549
12	0.4608780
13	0.4421737
14	0.3933643
15	0.4633016
16	0.4427512
17	0.4403465
18	0.4633016
19	0.4709849
20	0.4709849
21	0.4600758
22	0.4673485
23	0.4742107
24	0.4705743
25	0.4633016
26	0.4669380

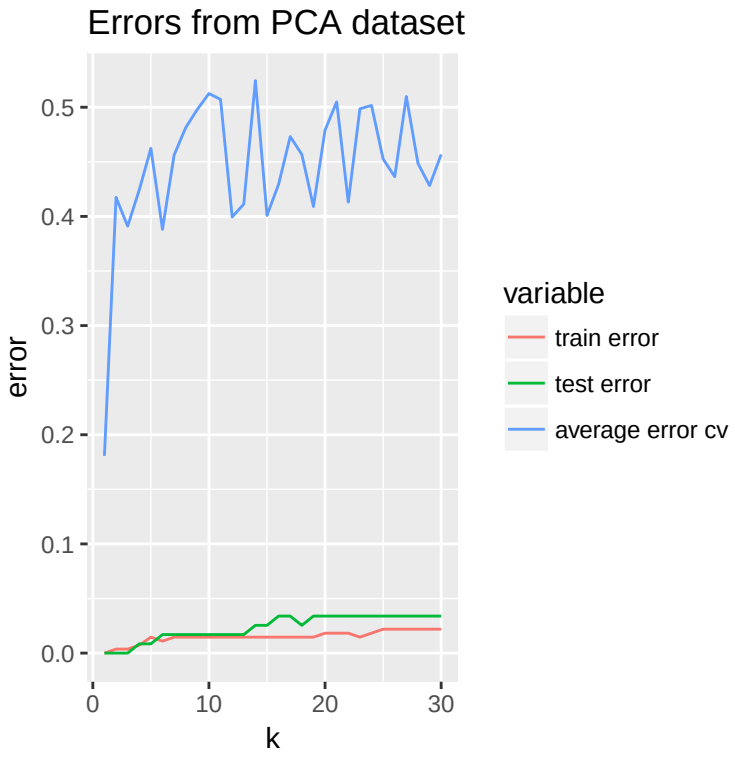
	average error cv
27	0.4669380
28	0.4742107
29	0.4742107
30	0.4705743



Finally, below is a table of the average cross-validation error on the test PCA dataset. The graph is the same as previously but with the PCA dataset.

	average error cv
1	0.1806366
2	0.4175043
3	0.3910540
4	0.4243302
5	0.4622863
6	0.3881474
7	0.4561777
8	0.4813478
9	0.4980454
10	0.5125908
11	0.5071734
12	0.3994925
13	0.4112921
14	0.5243905
15	0.4009025
16	0.4294739
17	0.4731103
18	0.4567838
19	0.4090658

	average error cv
20	0.4790472
21	0.5049470
22	0.4131474
23	0.4985649
24	0.5016817
25	0.4527021
26	0.4363756
27	0.5098450
28	0.4486205
29	0.4282124
30	0.4567838



Overall, it's odd that the cross validation error is so much higher than the test and train errors in all of the datasets. Additionally, all of the graphs suggest that we should be using a very small number of neighbors (around 1-3) which is really small and suggests a very complicated model which we might not want.

Comparing across all of the classification methods, the logistic regression method did the best in terms of the test errors. The logistic regression method only makes an assumption about the conditional distributions, specifically it assumes that the conditional distributions of the variables given their class are indepedent. QDA and LDA assume that the variables follow a Gaussian distribution. For LDA, the covariance matrices are assumed to be equal across the classes; whereas, the QDA does not make this assumption.

Comparing across all of the tests, the original dataset had the lowest test errors when we performed LDA but the scaled data had the lowest test errors when we performed QDA. This might be because we know that the variables are not all on the same scale so it doesn't necessarily make sense to assume that the covariance matrices are equal across the classes. Similarly, the original and the scaled dataset performed equally well on the training dataset in the logistic regression but the scaled dataset performed better on the test dataset. Again, this might be caused by a difference in the scales of the variables. These results suggests that scaling the data was a better approach, again this makes sense given what we know about the scales of the variables.

It also somewhat makes sense that the PCA data would perform worse on separating the data because PCA projects the data onto a linear combination of the variables which would be intuitively difficult to re-separate.

```
library(knitr)
knitr::opts_chunk$set(echo = FALSE, warning = FALSE, fig.align = "center", fig.height = 4, fig.width = 4

##########################################
# Set working directory and load libraries
##########################################

library(mixtools)
library(ggplot2)
library(plotly)
library(matrixcalc)
library(scatterplot3d)
library(reshape2)
library(knitr)
library(MASS)
library(GGally)
library(nnet)
library(class)


#####################################
# Read in data and clean
#####################################

set.seed(23987258)

cars <- read.table("~/Documents/UMich/Classes/2018 Spring/STATS 503/Homework/HW1/auto-mpg.data", col.nam

# contains "?"
cars$horsepower <- as.numeric(as.character(cars$horsepower))

# Converting to factors
cars$cylinders <- as.factor(cars$cylinders)
cars$model.year <- as.factor(cars$model.year)
cars$origin <- as.factor(cars$origin)

cars$class = 1
for (i in 1:length(cars$mpg)){
  if(cars$cylinders[i] == 6) {cars$class[i] = 2}
  if(cars$cylinders[i] == 8) {cars$class[i] = 3}
}

cars$class <- as.factor(cars$class)

cars_original <- na.omit(cars[,c(1, 3:6, 10)])

#####################################
# Scaled and centered dataset
#####################################

scale_center_cars <- scale((cars_original[,-6]))
```

```r
scale_center_cars <- as.data.frame(cbind(scale_center_cars, cars_original[,c("class")]))
colnames(scale_center_cars) <- colnames(cars_original)
scale_center_cars$class <- as.factor(scale_center_cars$class)

#####################################
# Corrected PCA analysis
#####################################

# The variables of interest are: MPG, displacement, horsepower, weight, and acceleration
pca_el_cars <- cars_original[,-6]

pca_el_cars_cent <- as.matrix(scale(na.omit(pca_el_cars), scale = F)) # matrix so we can do PCA #standa

# Perform PCA using correlation
# Pick correlation because the variables are not on similar scales
pca_cor_results <- princomp(pca_el_cars_cent, cor = T)
cars_eigenvec <- loadings(pca_cor_results)[, 1:2]
cars_pc <- as.matrix(scale((pca_el_cars))) %*% cars_eigenvec
cars_pc <- as.data.frame(cbind(cars_pc, cars_original$class))
colnames(cars_pc) <- c("mpg", "displacement", "class")
cars_pc$class <- as.factor(cars_pc$class)

#####################################
# Sample from each of the datasets
#####################################

classes = lapply(levels(cars_original$class), function(x) which(cars_original$class==x))

# Sample from original dataset
  train = lapply(classes, function(class) sample(class, 0.7*length(class), replace = F))
  train = unlist(train)
  test = (1:nrow(cars_original))[-train]

  original_train = cars_original[train,]
  original_test = cars_original[test,]

# Sample from scaled dataset
  train = lapply(classes, function(class) sample(class, 0.7*length(class), replace = F))
  train = unlist(train)
  test = (1:nrow(cars_original))[-train]

  scale_center_cars_train = scale_center_cars[train,]
  scale_center_cars_test = scale_center_cars[test,]

# Sample from PCA dataset
  train = lapply(classes, function(class) sample(class, 0.7*length(class), replace = F))
  train = unlist(train)
  test = (1:nrow(cars_original))[-train]

  cars_pc_train = cars_pc[train,]
  cars_pc_test = cars_pc[test,]
```

```
#####################################
# Summary statistics
#####################################

ggpairs(cars_original, columns=1:4,
        mapping=aes(color=class),
        diag="blank",
        axisLabels = "internal",
        upper=list(continuous='points'))

summary(original_train)
summary(original_test)

ggplot(cars_original, aes(x = mpg, y = displacement)) + geom_point(aes(color = class))
ggplot(original_test, aes(x = mpg, y = displacement)) + geom_point(aes(color = class))
summary(scale_center_cars_train)
summary(scale_center_cars_test)

ggplot(scale_center_cars_train, aes(x = mpg, y = displacement)) + geom_point(aes(color = class))
ggplot(scale_center_cars_test, aes(x = mpg, y = displacement)) + geom_point(aes(color = class))
summary(cars_pc_train)
summary(cars_pc_test)

ggplot(cars_pc_train, aes(x = mpg, y = displacement)) + geom_point(aes(color = class))
ggplot(cars_pc_test, aes(x = mpg, y = displacement)) + geom_point(aes(color = class))

#####################################
# creating a function for literally everything
#####################################

## creates the plots of the data points "projected" onto the first two discriminant directions (the lin
boundary_plot <- function(df, classifier, predict_function, resolution = 500, ...) {
  colnames(df) = c("Var1", "Var2", "Class")
  classifier_obj <- classifier(df)
  v1 = seq(min(df[,1]), max(df[,1]), length=resolution)
  v2 = seq(min(df[,2]), max(df[,2]), length=resolution)
  Grid = expand.grid(Var1 = v1, Var2 = v2)
  Grid$class = predict_function(classifier_obj, Grid, ...) ## dot, dot, dot passes in other arguments a
  ggplot(data=df, aes(x=Var1, y=Var2, color=Class)) +
    geom_contour(data=Grid, aes(z=as.numeric(class)),
                 color="black",size=.5)+
    geom_point(size=2,aes(color=Class, shape=Class))
}

# Add LDA; this is to plot
lda_wrapper <- function(df) lda(Class ~ ., df)
lda_predict_wrapper <- function(classifier, data) predict(classifier, data)$class

# Add QDA; this is to plot
qda_wrapper <- function(df) qda(x = df[,-3], grouping = df[,3])
qda_predict_wrapper <- function(classifier, data) predict(classifier, data)$class

# Wrap our logit prediction in a wrapper; this is to plot
```

```r
logit_wrapper <- function(df) glm(Class ~ ., family = "binomial", data = df)
logit_predict_wrapper <- function(classifier, data) (predict(classifier, data, type = "response")  > 0.5

# add k-nearest-neighbor function to plot
knn_wrapper <- function(df) {}
knn_predictor <- function(classifier_obj, Grid, train_data, class, k) {
  knn(train = train_data, cl = class, test = Grid, k = k) # knn needs training and test data at the sam
}

#-----------------------------------------------------------------

# creates error rates for the LDA and QDA
error_rate <- function(test_set, predicted_vals){
    1-sum(diag(table(test_set$class,predicted_vals$class)))/
    sum(table(test_set$class, predicted_vals$class))
}


# performs lda and then qda returns respective errors
lda_qda_analysis <- function(train_set, test_set){
  lda = lda(data=train_set, train_set$class~.)
  predicted_vals_lda = predict(lda, test_set)
  lda_error <- error_rate(test_set, predicted_vals_lda)

  qda = qda(data=train_set, train_set$class~.)
  predicted_vals_qda = predict(qda, test_set)
  qda_error <- error_rate(test_set, predicted_vals_qda)

  return(c(lda_error, qda_error))

}

#-----------------------------------------------------------------

logit_analysis <- function(train_data, test_data){

  # Run logistic regression using glm (can also use glmnet)
   cars_logit <- multinom(class ~ mpg + displacement, data = train_data, trace = F)

  # How did we do numerically? (How many did we predict incorrectly or what's our regret?)
    # training data
    logit_prediction <- (predict(cars_logit, train_data, type = "class"))
    logit_train_error <- mean(predict(cars_logit, train_data, type = "class") != train_data$class)

  # test
    logit_prediction <- (predict(cars_logit, test_data, type = "class"))
    logit_test_error <- mean(predict(cars_logit, test_data, type = "class") != test_data$class)

  return(c(logit_train_error, logit_test_error))
}

#-----------------------------------------------------------------
```

```r
# function for nearest neighbor classifier
# m is the number of nearest neighbors

knn_function <- function(train_data, test_data, m){

  knn_error <- as.data.frame(matrix(rep(0,2*m), nrow = m, ncol = 2))
  colnames(knn_error) <- c("train error", "test error")

  for(j in 1:m){
    cars_train_knn <- knn(train = train_data[,-7], cl = train_data$class,
                test = train_data[,-7], k = j)
    cars_test_knn <- knn(train = train_data[,-7], cl = train_data$class,
                test = test_data[,-7], k = j)
    knn_error[j, 1] <- mean(cars_train_knn != train_data$class)
    knn_error[j, 2] <- mean(cars_test_knn != test_data$class)
  }

    knn_error
}

# cross-validation function: subset training dataset into "train" and "test sets" (i = # of subsets)
# k-nearest neighbor (limited to n/i)
# i is the number of folds, m is the number of nearest neighbors

cv_function <- function(train_data, test_data, i, m){

  cv_knn_test_error <- as.data.frame(matrix(rep(0,m), nrow = m, ncol = 1))
  colnames(cv_knn_test_error) <- c("cv test error")

  train_data2 <- train_data
  train_data2$train_index <- floor(runif(nrow(train_data2))*100)
  train_data2$fold <- ceiling(train_data2$train_index / (100/i))

  cv_knn_error <- as.data.frame(matrix(rep(0,i*2*m), nrow = m, ncol = 1))

  for(i in 1:i){
    train_cv <- subset(train_data2, train_data2$fold != i)[, c(-2, -1)]
    train_test_cv <- subset(train_data2, train_data2$fold == i)[, c(-2, -1)]

    cv_knn_errori <- knn_function(train_cv, train_test_cv, m)
    colnames(cv_knn_errori) <- c(paste("fold", i, "train"), paste("fold", i, "test"))
    cv_knn_error <- cbind(cv_knn_error, cv_knn_errori)
  }
  cv_knn_error[,-1]
}

#-----------------------------------------------------------------
#######################################
# part b: LDA and QDA analysis
#######################################

# table thing (kable)
  lda_qda_error_o <- lda_qda_analysis(original_train, original_test)
```

```
  lda_qda_error_s <- lda_qda_analysis(scale_center_cars_train, scale_center_cars_test)
  lda_qda_error_p <- lda_qda_analysis(cars_pc_train, cars_pc_test)
  lda_qda_error <- rbind(lda_qda_error_o, lda_qda_error_s, lda_qda_error_p)
  rownames(lda_qda_error) <- c("original cars", "scaled cars", "pc cars")
  colnames(lda_qda_error) <- c("lda error", "qda error")

  kable(lda_qda_error)
# boundary plots for original data
  boundary_plot(cars_original[, c("mpg", "displacement", "class")], lda_wrapper,
    lda_predict_wrapper) + ggtitle("LDA boundary on original data \n(whole dataset)")

  boundary_plot(cars_original[, c("mpg", "displacement", "class")], qda_wrapper,
    qda_predict_wrapper) + ggtitle("QDA boundary on original data \n(whole dataset)")

# boundary plots for scaled data
  boundary_plot(scale_center_cars[, c("mpg", "displacement", "class")], lda_wrapper,
    lda_predict_wrapper) + ggtitle("LDA boundary on scaled data \n(whole dataset)")

  boundary_plot(scale_center_cars[, c("mpg", "displacement", "class")], qda_wrapper,
    qda_predict_wrapper) + ggtitle("QDA boundary on scaled data \n(whole dataset)")

# boundary plots for pca data
  boundary_plot(cars_pc[, c("mpg", "displacement", "class")], lda_wrapper,
    lda_predict_wrapper) + ggtitle("LDA boundary on PCA data \n(whole dataset)")

  boundary_plot(cars_pc[, c("mpg", "displacement", "class")], qda_wrapper,
    qda_predict_wrapper) + ggtitle("QDA boundary on PCA data \n(whole dataset)")
#######################################
# part c: logit analysis
#######################################

logit_o_error <- logit_analysis(original_train, original_test)
logit_s_error <- logit_analysis(scale_center_cars_train, scale_center_cars_test)
logit_p_error <- logit_analysis(cars_pc_train, cars_pc_test)
logit_error <- rbind(logit_o_error, logit_s_error, logit_p_error)
rownames(logit_error) <- c("original cars", "scaled cars", "pc cars")
colnames(logit_error) <- c("train error", "test error")
kable(logit_error)


#######################################
# part d: cross validation analysis
#######################################

# original dataset
knn_train_test_errors <- knn_function(original_train, original_test, 30)
cv_train_test_errors <- cv_function(original_train, original_test, 5, 30)
cv_test_errors <- cv_train_test_errors[, c(2,4,6,8,10)]
cv_test_errors_average <- as.data.frame(apply(cv_test_errors, 1, mean))
colnames(cv_test_errors_average) <- c("average error cv")
kable(cv_test_errors_average, row.names = T)

knn_train_test_cv_errors <- cbind(knn_train_test_errors, cv_test_errors_average)
```

```r
knn_train_test_cv_errors$k <- seq(1:30)
errors_melted <- melt(knn_train_test_cv_errors, id.vars = 'k')

ggplot(errors_melted, aes(x = k, y = value)) + geom_line(aes(color = variable, group = variable)) + labs
# scaled dataset
knn_train_test_errors <- knn_function(scale_center_cars_train, scale_center_cars_test, 30)
cv_train_test_errors <- cv_function(scale_center_cars_train, scale_center_cars_test, 5, 30)
cv_test_errors <- cv_train_test_errors[, c(2,4,6,8,10)]
cv_test_errors_average <- as.data.frame(apply(cv_test_errors, 1, mean))
colnames(cv_test_errors_average) <- c("average error cv")
kable(cv_test_errors_average, row.names = T)

knn_train_test_cv_errors <- cbind(knn_train_test_errors, cv_test_errors_average)
knn_train_test_cv_errors$k <- seq(1:30)
errors_melted <- melt(knn_train_test_cv_errors, id.vars = 'k')

ggplot(errors_melted, aes(x = k, y = value)) + geom_line(aes(color = variable, group = variable)) + labs

# pc dataset
knn_train_test_errors <- knn_function(cars_pc_train, cars_pc_test, 30)
cv_train_test_errors <- cv_function(cars_pc_train, cars_pc_test, 5, 30)
cv_test_errors <- cv_train_test_errors[, c(2,4,6,8,10)]
cv_test_errors_average <- as.data.frame(apply(cv_test_errors, 1, mean))
colnames(cv_test_errors_average) <- c("average error cv")
kable(cv_test_errors_average, row.names = T)

knn_train_test_cv_errors <- cbind(knn_train_test_errors, cv_test_errors_average)
knn_train_test_cv_errors$k <- seq(1:30)
errors_melted <- melt(knn_train_test_cv_errors, id.vars = 'k')

ggplot(errors_melted, aes(x = k, y = value)) + geom_line(aes(color = variable, group = variable)) + labs
```