Introduction:

The Auto MPG dataset contains car data that includes elements for mpg, number of cylinders, displacement, horsepower, weight, acceleration, model year, origin, and car name (exempt). This data set will be used to increase our understanding in data cleansing, which includes statistically handling missing data, data exploration, and splitting the data into training and test sets. Additionally, we will run classification models to predict the mpg using LDA, QDA, Naive Bayes and Logistic Regression. We'll be using the latest classification methods to increase our understanding and practicallity of their uses.

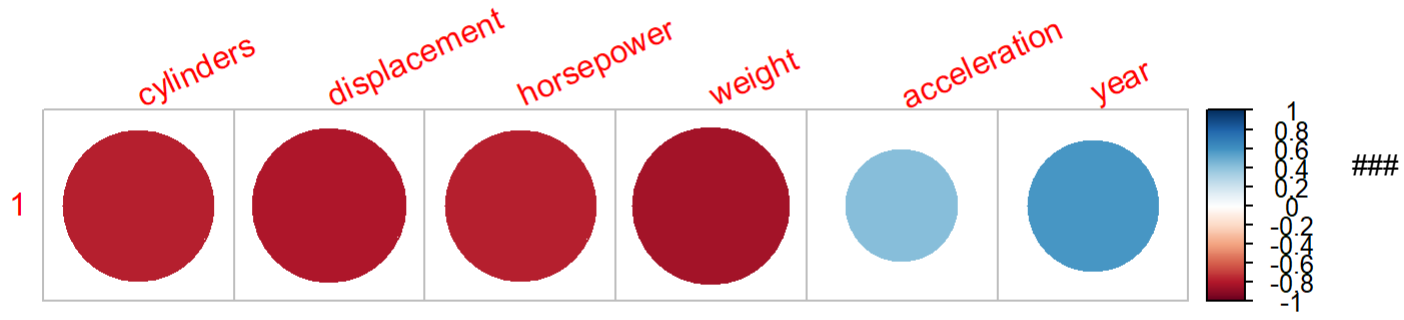
Exploratory Data Analysis:

Since we will be running classifications models with the mpg as our y variable, we will look into how the other variables correlate. Once we have the correlations between the mpg variable and the predictors, we will have more insight as to which x variables are more likely to affect the prediction score.

Correlations:

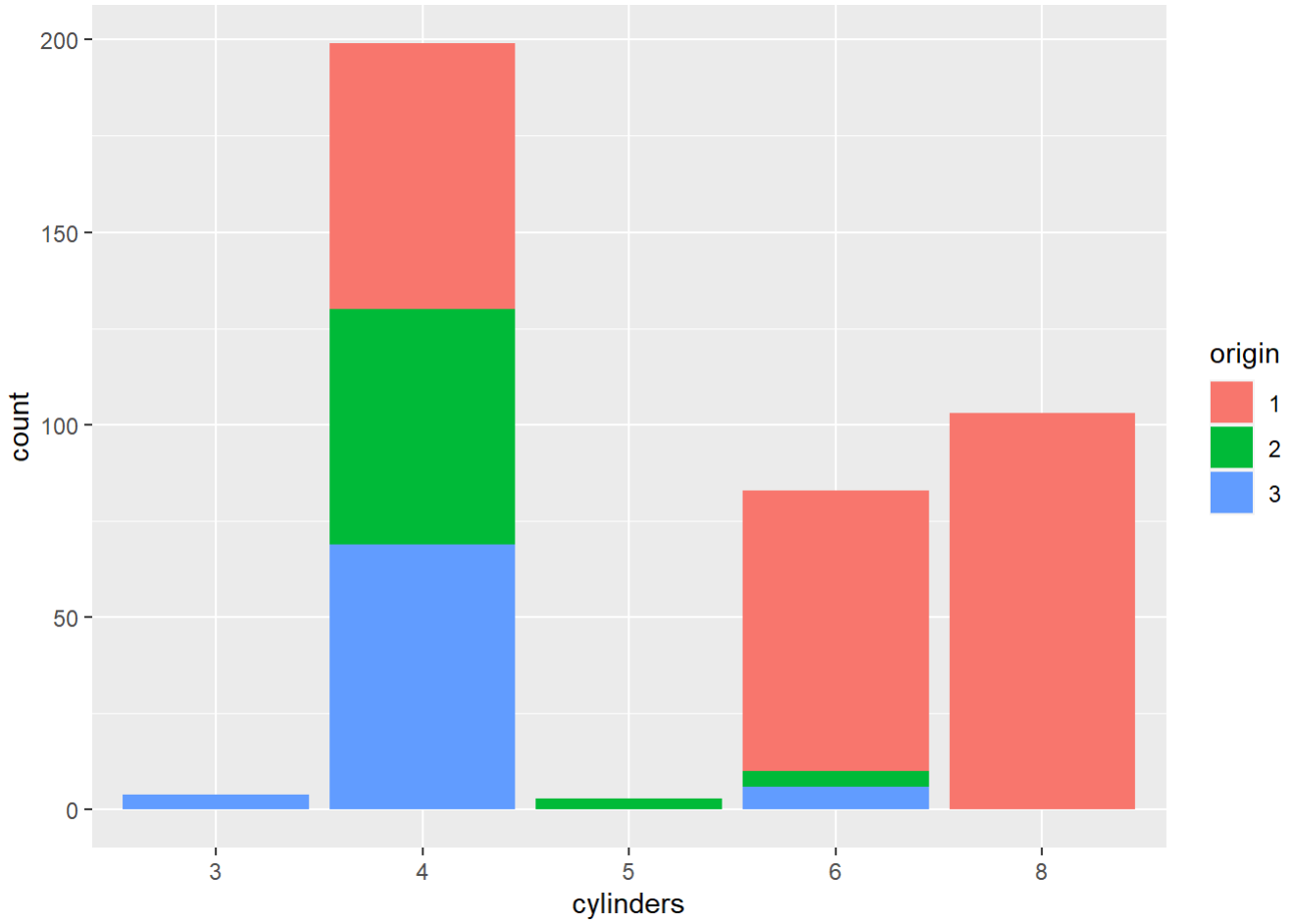
Below contains a heat map of all the predictor variables against the mpg, the y variable. The colors (red/blue) represent the correlation each variable has to mpg. The darker the shade the higher the correlation. Cylinders, displacement, horsepower, and weight are all have a high negative correlation to mpg and acceleration and year have a positive correlation. Year has the highest positive correlation.

```
mcor <- cor(x = Auto1$mpg, y = Auto1[,c(-1, -8)], use="complete.obs")
corrplot(mcor, tl.srt = 25)
```



Comparing Categorical Data: This chart was created to show the data representation between the different categorical groups. The grouped bar chart below has the counts of the cylinders grouped by their orgin (country). Based on the data it looks like there isn't an equal representation between the two groups, origin and cylinders. Most cars have four cylinders, where the origins are equally represented. The other cylinder types have a majority representation of one country.

```
Auto2 <- Auto1
Auto2[, 8] <- as.factor(Auto2[, 8]) ## converting origin to factor
Auto2[, 2] <- as.factor(Auto2[, 2]) ## converting cylinders to factor
ggplot(Auto2, aes(cylinders, fill=origin)) +
  geom_bar(position = "stack")
```



Methods:

Train/Test Sets:

Prior to classifiying the data, it needs to be split into training and test sets. For splitting the data we will randomly use 75% of the samples for the training set and the remaining 25% as test data. For the data paritioning we will utilize functions from the caTools package.

```
set.seed(42)
sample <- sample.split(Auto[,1], SplitRatio = .75)
train <- subset(Auto, sample == TRUE)
test  <- subset(Auto, sample == FALSE)
Auto$mpg01 <- as.factor(Auto$mpg01)
cl <- train[,1]
```

Next, in this we will cover the five different classication methods we will use to predict if a car has a higher mpg than the median mpg of our sample.

LDA:

```
lda_model <- lda(mpg01~., data = train)
lda_predict <- predict(lda_model, test[,-1])
mean(cl == lda_predict$class)
```

```
## [1] 0.5238095
```

QDA:

```
qda_model <- qda(mpg01~., data = train)
qda_predict <- predict(qda_model, test[,-1])
mean(cl == qda_predict$class)
```

```
## [1] 0.5136054
```

Naive Bayes:

```
nb_model <- naiveBayes(mpg01~., data = train)
nb_predict <- predict(nb_model, test[,-1])
mean(cl == nb_predict)
```

```
## [1] 0.5204082
```

Logistic Regression:

```
X1 <- cbind(Auto[Auto$mpg01 == TRUE, ]$mpg01)
X0 <- cbind(Auto[Auto$mpg01 == FALSE, ]$mpg01)
lr_model <- glm(mpg01~., data = train, family = binomial(link=logit))

lr_predict <- predict(lr_model, test[,-1])
lr_predict.classes <- ifelse(lr_predict > 0.5, TRUE, FALSE)
mean(lr_predict.classes == test[,1])
```

```
## [1] 0.9081633
```

KNN:

The chosen values for the KNN method will be 1, 3, 5, 7, and 10. These were arbitrarily chosen to determine where the optimized k value could be. If 10 (the highest value) is the optimal value we will add to the list. From the KNN scores, 1 performed the highest with 3 falling in second place. When 1 has the optimal k value its usually a sign of over fitting, however this is the test data set, so this is probably due to the similaries in the test and training dataset. Considering most of the cars were four cylinders in the analysis above, it is possible that the sample of cars are similar. In a real life situation I would go with 3 (2nd highest) instead of 1.

```
knn_hash <- hash()
cl <- train[, 1]
for (i in c(1,3,5,7,10)) {
  # Running KNN on each i
  key = as.character(i)
  knn_model = knn(train=train, test=test, cl=cl, k=i)
  knn_predict = mean(cl == knn_model)
  knn_hash[[key]] = knn_predict * 100
}

knn_hash
```

```
## <hash> containing 5 key-value pair(s).
##   1 : 53.40136
##   10 : 51.02041
##   3 : 52.04082
##   5 : 51.02041
##   7 : 51.36054
```

Results:

Based the results above the logist regression model gave the highest score based on the sample and the test/train data split. I'm skeptical about the logist regression score because the prediction data didn't return values between 0 and 1. Considering most of the logistic regression prediction values were above 1, I don't feel confident the outcome. The scores for the other classifiers were 52% Niave Bayes, 51.3% QDA, 52.3 LDA and 52.04 KNN(3). Removing KNN(1) and the Logistic Regression prediction scores

Appendix:

Further KNN Analysis:

```
knn_hash <- hash()
cl <- train[, 1]
for (i in 1:40) {
  # Running KNN on each i
  key = as.character(i)
  knn_model = knn(train=train, test=test, cl=cl, k=i)
  knn_predict = mean(cl == knn_model)
  knn_hash[[key]] = knn_predict * 100
}

knn_hash
```

```
## <hash> containing 40 key-value pair(s).
##   1 : 53.40136
##   10 : 51.02041
##   11 : 50.68027
##   12 : 51.70068
##   13 : 50.68027
##   14 : 50.68027
##   15 : 51.02041
##   16 : 50.68027
##   17 : 50.68027
##   18 : 49.65986
##   19 : 50.68027
##   2 : 51.36054
##   20 : 51.02041
##   21 : 51.02041
##   22 : 50.68027
##   23 : 49.31973
##   24 : 49.31973
##   25 : 49.31973
##   26 : 49.31973
##   27 : 49.31973
##   28 : 49.31973
##   29 : 48.97959
##   3 : 52.04082
##   30 : 49.31973
##   31 : 48.97959
##   32 : 48.97959
##   33 : 48.97959
##   34 : 48.97959
##   35 : 48.97959
##   36 : 48.97959
##   37 : 48.97959
##   38 : 49.31973
##   39 : 48.97959
##   4 : 51.70068
##   40 : 48.97959
##   5 : 51.02041
##   6 : 50
##   7 : 51.36054
##   8 : 52.04082
##   9 : 51.36054
```