Introduction

This homework assignment's goal is to get more familiar with classification. There are many kinds of classification techniques, and this assignment focuses on a couple of them – Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Naïve Bayes, Logistic Regression, and K Nearest Neighbors (KNN). Each classification technique's training error, testing error, average cross validation error, and cross validation variance will be looked at and compared.

The data set being used for this assignment is the "Auto MPG" dataset. This data set shows the miles per gallon (mpg) of multiple cars based on 9 variables. For this assignment, the mpg column will be turned into a classifier – 1 if the mpg is greater than the median mpg, or 0 if the mpg is less than the median mpg.

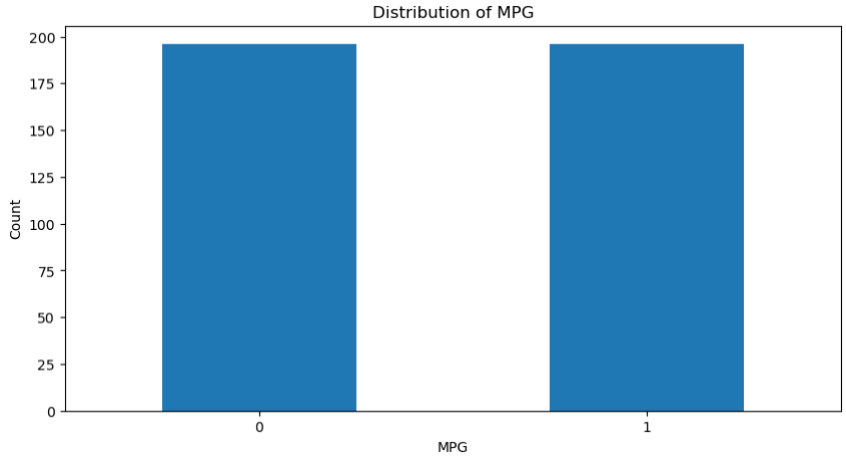
Explanatory Data Analysis

The original data set had 398 rows, which meant that there were 398 different kinds of cars. Each row consisted of 9 columns – mpg, cylinders, displacement, horsepower, weight, acceleration, model year, origin, and car_name. Horsepower had some missing values in it, so for this assignment if there was a row missing a horsepower value the entire row was removed. Also, the car_name column was removed. This led to the data set being used to be 392 rows, each with 8 columns.

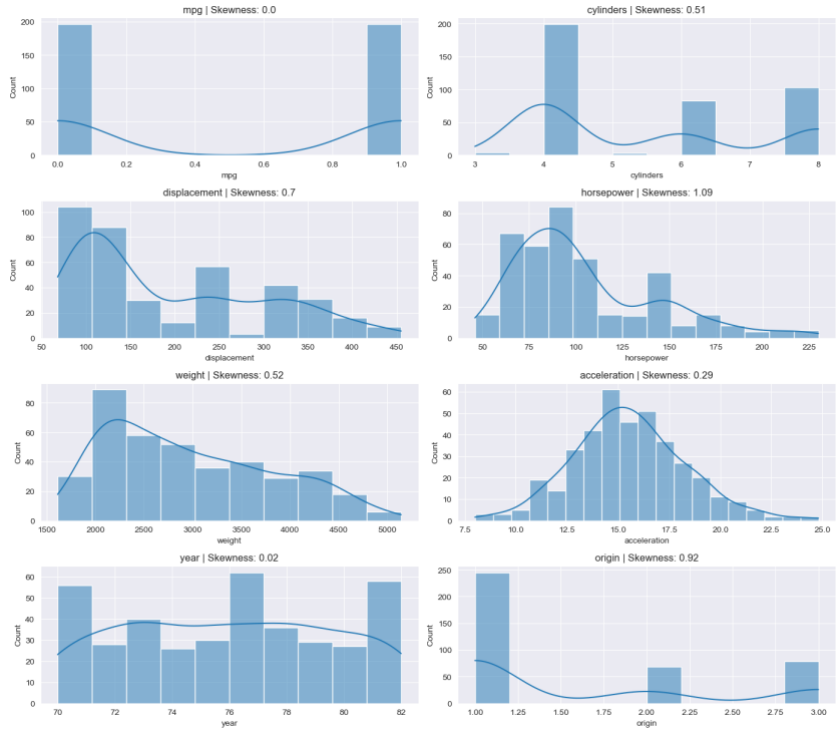
To convert the mpg column to a classifier, the median of the column had to be found. Below is some information on each column:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
Mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.541327	75.979592	1.576531
Median	22.75	4.00	151.00	93.50	2803.50	15.50	76.00	1.00
Mode	13.00	4.00	97.00	150.00	1985.00	14.50	73.00	1.00

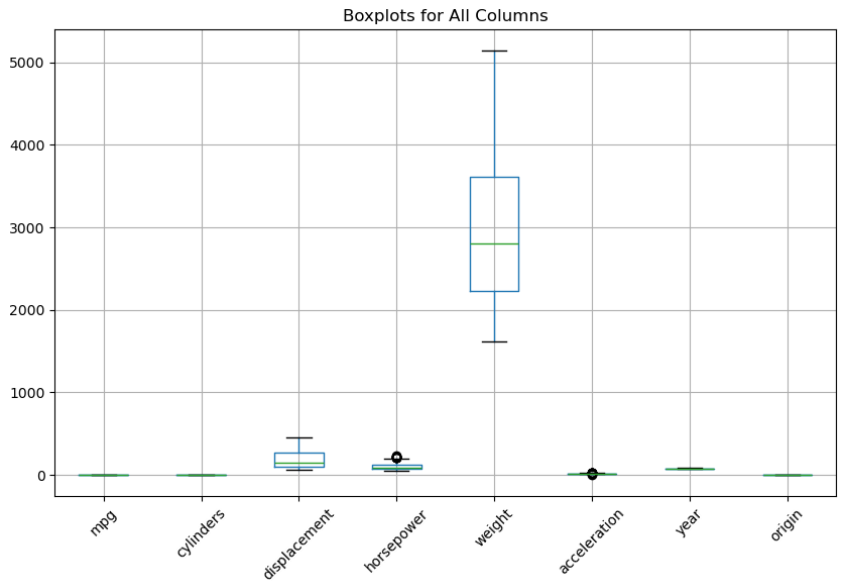
So, in this case, for each row in the data set, if the mpg value is greater than 22.75 then mpg got converted to a 1, and a 0 if it was less. After this was done, the amount of 1's and 0's was found.



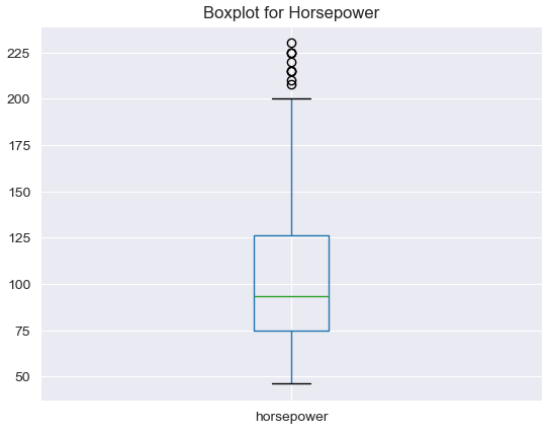
It turned out that there was a perfect split between the 1's and 0's – each had a count of 196 instances. The distribution of each column was then looked at to understand the spread and patterns of each one.

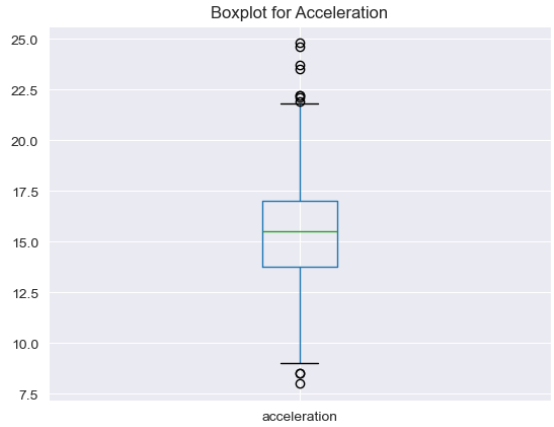


After looking at these histograms, it is shown that a couple of the columns have only some values they can be. The mpg column can only be 1 or 0 due to being the classifier, the cylinders column contains only the values (3,4,5,6,8), and the origin column contains only the values (1,2,3). None of the columns are too skewed, except for horsepower which is slightly positively skewed. This means that there are more lower values than higher in horsepower. Although there are no apparent outliers when looking at the histograms, boxplots were run for each column to see if that held true.

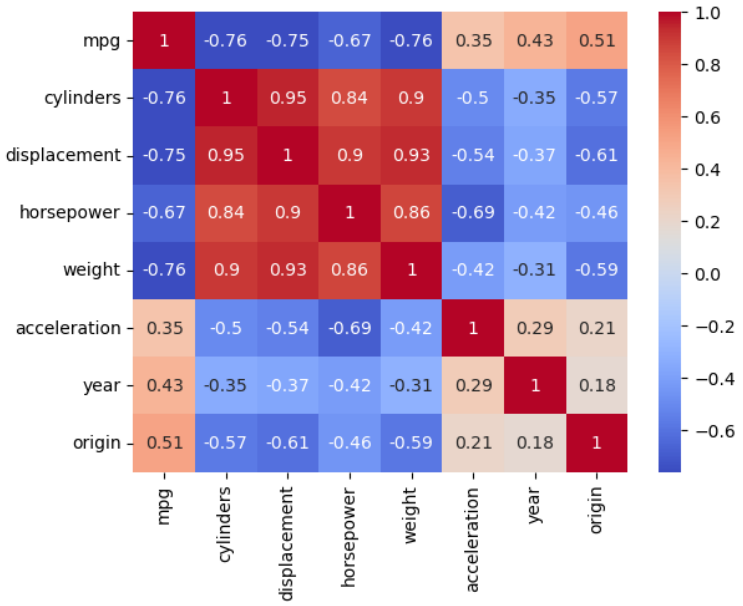


Upon further inspection, it was found that there are outliers in the horsepower and acceleration columns. These were looked at more closely.





It was found that horsepower has multiple values that fall above the 75th percentile and acceleration has 2 values that fall below the 25th percentile and multiple that fall above the 75th. Due to these values not being too extreme and not making the histograms too skewed, it was decided to keep them in the data set. The last thing looked at was the correlation matrix of the data set.



If two variables are closely related to each other then they will be red (1 = perfect positive correlation), if they are not related to each other they will be white (0 = no correlation), and if they are negatively related to each other they will be blue (-1 = perfect negative correlation). The darker the color, the stronger the relation. This matrix also contains the correlation values. Looking at this matrix, it shows some variables are very correlated to

each other. Displacement, horsepower, and weight all have a strong positive correlation with each other. Mpg has a strong negative correlation with cylinders, displacement, horsepower, and weight. Intuitively, this makes sense. For example, a semi-truck would eat up more gas on the road than a small car would, lowering its mpg.

Methods/Methodology

There were multiple methods used in this assignment, 5 in total. Each one was a different form of classification. The first one used was Linear Discriminant Analysis (LDA). LDA is used for both classification and dimensionality reduction. It looks for variable combinations that maximize separation of the different classes. It also reduces the number of dimensions while keeping the most class-discriminatory information as it can. LDA assumes each class is normally distributed and they have the same covariance. The second method used was Quadratic Discriminant Analysis (QDA). Unlike the LDA, the QDA allows each class to have its own covariance. QDA tends to be better than LDA when the class distributions are more complex and not linearly separable. Naïve Bayes was the third method used, specifically Gaussian Naïve Bayes (GNB). GNB is best used when each variable is numeric and about normally distributed – which this data set has. GNB uses the Bayes' Theorem to get the probability of a class given a set of variables, assumes all the variables are independent, and assigns a new data point to the class with the highest probability found with the Bayes' Theorem. The fourth method was logistic regression, which in this case it was binary logistic regression. Binary logistic regression is used when there are two classes, which in this case there was either 1 or 0. It converts a linear output into a probability between 0 and 1. If the probability of the event occurring, for example in this case, if the probability of mpg being a 1 given the independent variable values is greater than 0.5, then that event is classified as a 1. If it is lower, then it is classified as a 0 – meaning the event does not occur. The last classification method used was K Nearest Neighbors (KNN). KNN in this case works by finding some k number of data points around a new data point, seeing if there are more 1's or 0's around the new data point, and classifying the new data point based on which had the majority. For example, if there were 3 1's and 2 0's around a new data point, then the new data point would be classified as a 1.

The mean standard error of the training model, testing model, and cross validated models were all found and compared. The variance of the cross validated models was also found. The training and testing data were randomly split at 80% training 20% testing. I chose an 80/20 split because it provides a good balance between having enough data to train the model and enough to test it. 80% training allows the model to generalize well to unseen data, while 20% testing allows for a good amount to evaluate performance. There ended up being 313 rows in the training data and 79 in the testing data.

The cross validation used for this assignment was k folds cross validation. K folds cross validation splits the data up into k equal parts (folds), in this case 10, and trains the model k times. Each time it trains the model it uses a different fold as the testing data and uses the rest as the training data.

Results

LDA:

Training Error	0.079872
Testing Error	0.126582
Mean Cross Validation Error	0.104808
Cross Validation Variance	0.002307

LDA has a relatively low training error compared to all the other models, but higher testing and cross validation errors than the others. The variance shows that the model is moderately stable, meaning that its performance was slightly different across the different folds. In comparison to the other models, this model may not be as effective due to its higher testing and cross validation errors.

QDA:

Training Error	0.067093
Testing Error	0.113924
Mean Cross Validation Error	0.102115
Cross Validation Variance	0.001063

QDA produced results that were similar to LDA, but it performed better across all tests as it had a lower value in each test. This likely happened due to how both tests function. QDA makes fewer assumptions about the data's covariance, which allows it to build more complex relationships between the variables. As shown from the histograms, the spread of each variable was not identical, so it makes sense that QDA would produce better results than the test that assumes all variables are normally distributed.

GNB:

Training Error	0.086262
Testing Error	0.126582
Mean Cross Validation Error	0.099744
Cross Validation Variance	0.002162

The GNB model produced higher training and testing errors than QDA, but a lower cross validated value. The variance of the GNB model also shows that it is relatively stable. It produced the same testing error as LDA and a lower cross validation error, which suggests that while GNB is less complex than LDA, it seems to capture the relationships in the data just as well if not better.

Logistic Regression:

Training Error	0.086262
Testing Error	0.113924
Mean Cross Validation Error	0.104615
Cross Validation Variance	0.001766

Logistic regression produced the same training error as the GNB model and a slightly lower testing error. The cross validation error is similar to the LDA and QDA models, and even has a lower variance value than the LDA model. This model is relatively stable and performs similarity to all the other models. Although similar and having the same testing error, it does not outperform the QDA model due to the higher cross validated error and variance.

KNN:

k value	Training Error	Testing Error	Mean CV Error	CV Variance
1	0.000000	0.164557	0.137692	0.001951
3	0.063898	0.075949	0.109808	0.001602
5	0.086262	0.113924	0.117500	0.001760
7	0.099042	0.113924	0.122564	0.001830
9	0.111821	0.139241	0.117436	0.001483
11	0.108626	0.151899	0.117372	0.000686
13	0.111821	0.126582	0.120000	0.001998
15	0.118211	0.139241	0.112372	0.001624

When looking at the training error, it seems to steadily increase with each value of k except for when k goes from 9 to 11. A value of k = 1 on the training data indicates that the model perfectly fits the data. This means that if a new data point was introduced, it would be classified as a 1 or 0 perfectly. This changes as the model got introduced to the testing data, where k = 3 produced the lowest testing error value and every other k value also produced lower testing error values than k = 1. This indicates that the model was being overfit with just the training data at k = 1 and likely every other k value as well. k = 3 also produced the lowest cross validated error while k = 1 produced the largest. As k increases, the model becomes more generalized and considers more neighbors when making

predictions. More neighbors mean less reliance on individual data points, which reduces overfitting. This balance between overfitting and generalization seems to be best at k = 3, which produced the lowest testing error and one of the lowest cross validation errors across all models while having a low variance.

Findings

Looking at all the results, if I were to pick a model, I would choose the KNN model with k = 3. Out of all the models, this one had the lowest testing error and a competitive cross validation error and variance. In terms of KNN, k = 3 allows for the model to use enough data points to balance between overfitting and generalizing. The QDA model also provides results with low error and with good generalization.

I found a few things interesting in this assignment. QDA working better than LDA was surprising to me. LDA is generally considered a robust method and works well in many cases. After further inspection, I believe that QDA worked better because it allowed for different covariances for each class. This means that this data set may now follow the assumptions that LDA makes. It was also surprising that GNB performed similarly to QDA. GNB has simple independence assumptions, so this may mean that these assumptions might not be too far from the truth in this data set.

In conclusion, the performance of the various models highlights the importance of understanding the underlying assumptions and characteristics of each algorithm. LDA, despite being a commonly used method, underperformed relative to QDA and GNB, likely due to its assumption of equal covariance across classes. QDA, with its more flexible approach, had more of an advantage in this case. Logistic Regression did not deliver better results, suggesting that its linear assumptions may not be ideal for this dataset. The KNN model demonstrated sensitivity to the choice of k, where smaller values led to overfitting and larger values caused underfitting.

Appendix

```
#!/usr/bin/env python
# coding: utf-8


# Load Data


# In[504]:



import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

auto1 = pd.read_csv("Auto.csv")
auto1.head()


# In[470]:



auto = auto1.copy()
auto['mpg'] = (auto['mpg'] > auto['mpg'].median()).astype('int')
display(auto)
```

EDA

In[472]:

```
mean_values = auto1.mean()      # Mean of each column
median_values = auto1.median()   # Median of each column
mode_values = auto1.mode().iloc[0]

print("Mean:\n", mean_values)
print("Median:\n", median_values)
print("Mode:\n", mode_values)
```

In[473]:

```
auto1.describe()
```

In[474]:

```
counts = auto.groupby('mpg').size()
print(counts)
```

```
auto['mpg'].value_counts().sort_index().plot(kind='bar', figsize=(10, 5))
plt.xlabel('MPG')
plt.ylabel('Count')
plt.title('Distribution of MPG')
plt.xticks(rotation=0)
plt.show()



# In[475]:



sns.set_style("darkgrid")

numerical_columns = auto.select_dtypes(include=["int64", "float64"]).columns

plt.figure(figsize=(14, len(numerical_columns) * 3))
for idx, feature in enumerate(numerical_columns, 1):
    plt.subplot(len(numerical_columns), 2, idx)
    sns.histplot(auto[feature], kde=True)
    plt.title(f"{feature} | Skewness: {round(auto[feature].skew(), 2)}")

plt.tight_layout()
plt.show()



# In[476]:
```

```
auto.boxplot(figsize=(10, 6))  # Creates boxplots for all numerical columns
plt.title("Boxplots for All Columns")
plt.xticks(rotation=45)  # Rotate labels for better readability
plt.show()



# In[477]:



auto.boxplot(column = 'horsepower')
plt.title("Boxplot for Horsepower")
plt.show()
auto.boxplot(column = 'acceleration')
plt.title("Boxplot for Acceleration")
plt.show()



# In[478]:



auto.corr()
sns.heatmap(auto.corr(), annot=True, cmap="coolwarm")  # Heatmap of correlations
plt.show()
```

Models

In[480]:

```
X = auto.drop(columns=['mpg'])  # Features
y = auto['mpg']  # Target variable

# Split into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set size:", X_train.shape)
print("Testing set size:", X_test.shape)
```

In[481]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score

#LDA

# LDA Model
```

```
lda = LinearDiscriminantAnalysis()

# Fit on the training data
lda.fit(X_train, y_train)

# Predict on the training data
y_train_pred = lda.predict(X_train)

# Calculate training MSE
training_mse = mean_squared_error(y_train, y_train_pred)
print(f"Training Mean Squared Error: {training_mse:.6f}")

# Predict on the test data
y_test_pred = lda.predict(X_test)

# Calculate testing MSE
testing_mse = mean_squared_error(y_test, y_test_pred)
print(f"Testing Mean Squared Error: {testing_mse:.6f}")

# Perform 10-fold cross-validation on the training data
cv_scores = cross_val_score(lda, X, y, cv=10, scoring='neg_mean_squared_error')

# Negate the scores to get the positive MSE values
cv_mse_scores = -cv_scores

# Compute the mean of MSE across all folds
```

```
average_mse = np.mean(cv_mse_scores)
# Compute the variance of MSE across all folds
variance_mse = np.var(cv_mse_scores)

# Print the results
print(f"Average Mean Squared Error from Cross-Validation: {average_mse:.6f}")
print(f"Variance of Mean Squared Error from Cross-Validation: {variance_mse:.6f}")



# In[482]:



from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

#QDA
qda = QuadraticDiscriminantAnalysis()
# Fit on the training data
qda.fit(X_train, y_train)

# Predict on the training data
y_train_pred = qda.predict(X_train)

# Calculate training MSE
training_mse = mean_squared_error(y_train, y_train_pred)
print(f"Training Mean Squared Error: {training_mse:.6f}")
```

```
# Predict on the test data
y_test_pred = qda.predict(X_test)

# Calculate testing MSE
testing_mse = mean_squared_error(y_test, y_test_pred)
print(f"Testing Mean Squared Error: {testing_mse:.6f}")

# Perform 10-fold cross-validation on the training data
cv_scores = cross_val_score(qda, X, y, cv=10, scoring='neg_mean_squared_error')

# Negate the scores to get the positive MSE values
cv_mse_scores = -cv_scores

# Compute the mean of MSE across all folds
average_mse = np.mean(cv_mse_scores)
# Compute the variance of MSE across all folds
variance_mse = np.var(cv_mse_scores)

# Print the results
print(f"Average Mean Squared Error from Cross-Validation: {average_mse:.6f}")
print(f"Variance of Mean Squared Error from Cross-Validation: {variance_mse:.6f}")


# In[483]:
```

```
from sklearn.naive_bayes import GaussianNB

#Naive Bayes
gnb = GaussianNB()
# Fit on the training data
gnb.fit(X_train, y_train)

# Predict on the training data
y_train_pred = gnb.predict(X_train)

# Calculate training MSE
training_mse = mean_squared_error(y_train, y_train_pred)
print(f"Training Mean Squared Error: {training_mse:.6f}")

# Predict on the test data
y_test_pred = gnb.predict(X_test)

# Calculate testing MSE
testing_mse = mean_squared_error(y_test, y_test_pred)
print(f"Testing Mean Squared Error: {testing_mse:.6f}")

# Perform 10-fold cross-validation on the training data
cv_scores = cross_val_score(gnb, X, y, cv=10, scoring='neg_mean_squared_error')

# Negate the scores to get the positive MSE values
cv_mse_scores = -cv_scores
```

```
# Compute the mean of MSE across all folds
average_mse = np.mean(cv_mse_scores)
# Compute the variance of MSE across all folds
variance_mse = np.var(cv_mse_scores)

# Print the results
print(f"Average Mean Squared Error from Cross-Validation: {average_mse:.6f}")
print(f"Variance of Mean Squared Error from Cross-Validation: {variance_mse:.6f}")



# In[484]:



from sklearn.linear_model import LogisticRegression

#Logistic Regression

log_reg = LogisticRegression()
# Fit on the training data
log_reg.fit(X_train, y_train)

# Predict on the training data
y_train_pred = log_reg.predict(X_train)

# Calculate training MSE
```

```
training_mse = mean_squared_error(y_train, y_train_pred)
print(f"Training Mean Squared Error: {training_mse:.6f}")

# Predict on the test data
y_test_pred = log_reg.predict(X_test)

# Calculate testing MSE
testing_mse = mean_squared_error(y_test, y_test_pred)
print(f"Testing Mean Squared Error: {testing_mse:.6f}")

# Perform 10-fold cross-validation on the training data
cv_scores = cross_val_score(log_reg, X, y, cv=10, scoring='neg_mean_squared_error')

# Negate the scores to get the positive MSE values
cv_mse_scores = -cv_scores

# Compute the mean of MSE across all folds
average_mse = np.mean(cv_mse_scores)
# Compute the variance of MSE across all folds
variance_mse = np.var(cv_mse_scores)

# Print the results
print(f"Average Mean Squared Error from Cross-Validation: {average_mse:.6f}")
print(f"Variance of Mean Squared Error from Cross-Validation: {variance_mse:.6f}")
```

In[485]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

#KNN
#varibales most associated are cylinders, displacement, horsepower, weight, origin (all >.50 in correlation)
knn_dataset = auto[['mpg','cylinders','displacement','horsepower','weight','origin']]
X_selected = knn_dataset.drop(columns=['mpg'])  # Features
y_selected = knn_dataset['mpg']  # Target variable

# Split into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X_selected, y_selected, test_size=0.2, random_state=42)

k_vals = [1,3,5,7,9,11,13,15]

training_errors = []

testing_errors = []

cv_scores = []

for k in k_vals:
    knn = KNeighborsClassifier(n_neighbors = k)
```

```
    # Fit on the training data
    knn.fit(X_train, y_train)

    # Predict on the training data
    y_train_pred = knn.predict(X_train)

    # Calculate training MSE
    training_mse = mean_squared_error(y_train, y_train_pred)
    training_errors.append((k,round(training_mse, 6)))

    # Predict on the test data
    y_test_pred = knn.predict(X_test)

    # Calculate testing MSE
    testing_mse = mean_squared_error(y_test, y_test_pred)
    testing_errors.append((k,round(testing_mse, 6)))

    cv_score = cross_val_score(knn, X_selected, y_selected, cv=10,
scoring='neg_mean_squared_error')

    cv_mse = -cv_score
    var_mse = np.var(cv_mse)
    cv_scores.append((k, cv_mse, var_mse))

cv_scores_means = []
```

```
for tup in cv_scores:
    cv_scores_means.append((tup[0], np.mean(tup[1]), tup[2]))

print(training_errors)
print(testing_errors)
print(cv_scores_means)
lowest_mean = 1
best_k = 0

for tup in cv_scores_means:
    if tup[1] < lowest_mean:
        lowest_mean = tup[1]
        best_k = tup[0]

print()
print(lowest_mean)
print(best_k)



# In[ ]:
```