ISyE 7406: Data Mining and Statistical Learning **(Due: 02/13/2020)**

Homework Assignment #3

Instructor: Yajun, Mei *Name:* Chen-Yang(Jim), Liu *GTID:* 90345****

Problem 1: Classification Methods (50 points)

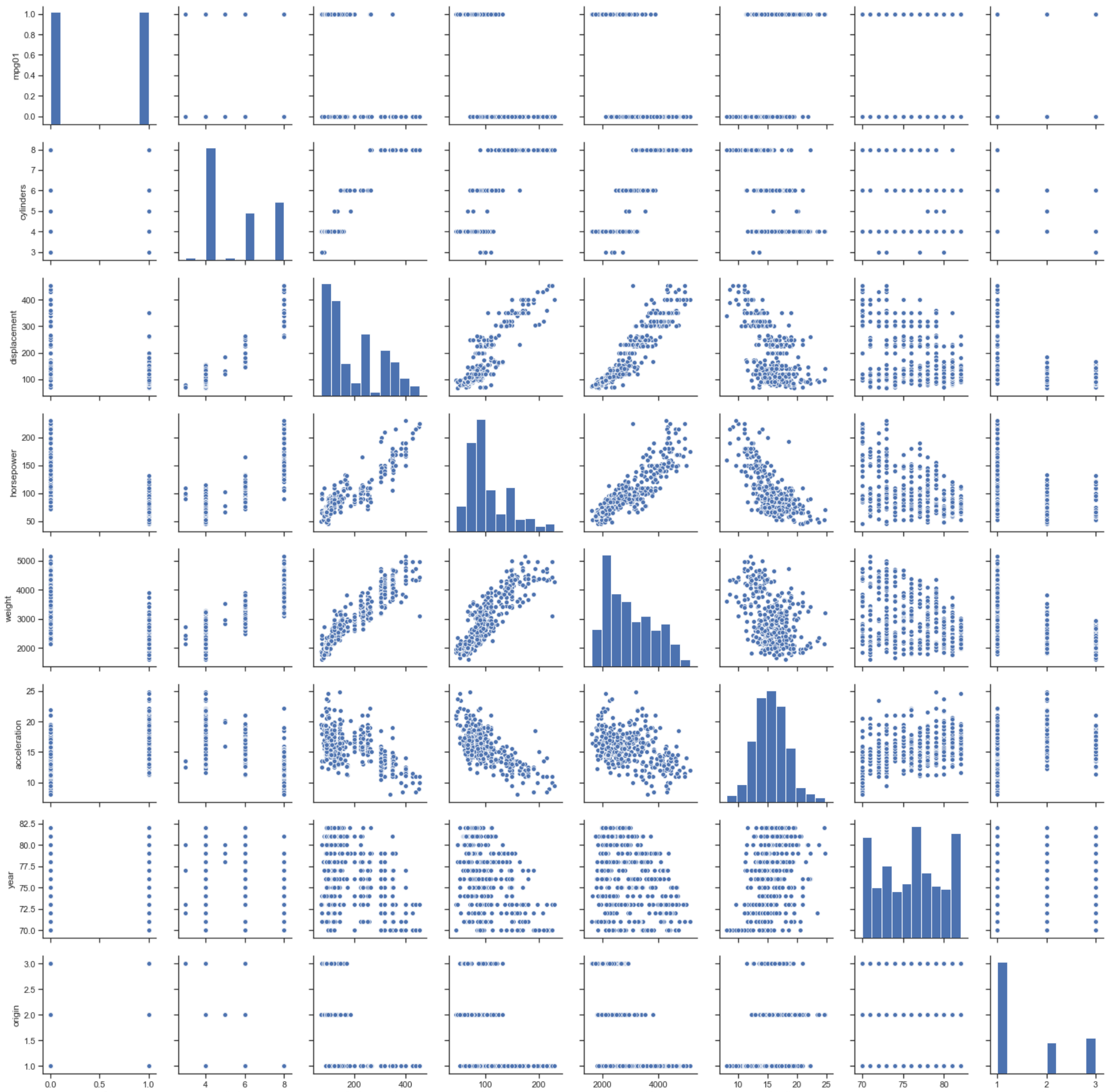
(a) Introduction

The purpose of this assignment is to use classification methods to analyze data. Before data analysis, data source and feature explanation are needed. Data source is critical because we could not analyze artificial data. So we will briefly introduce data sources. This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. Originally, there are nine features and one of them is names of manufacturer. However, for further analysis, this feature is deleted and updated dataset is provided with eight features. Among those features, the origin feature is hard to interpret. As a result, looking back to the original dataset, it seems that the numbers in origin column represent where cars are made from.

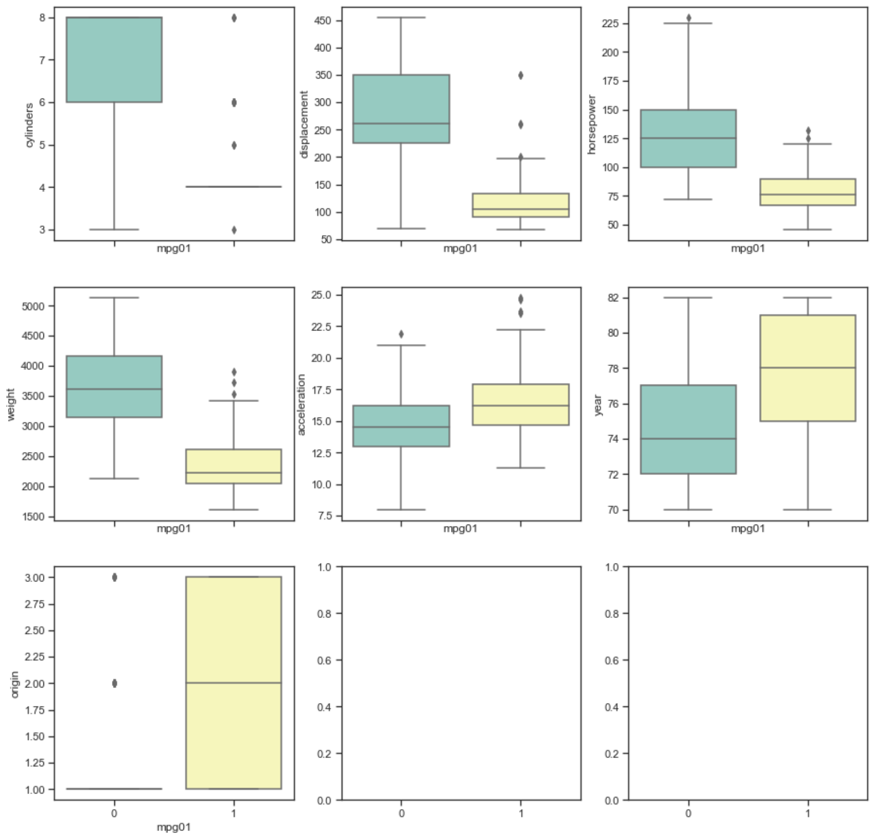
- mpg: Miles per gallon (Continuous variable)
- cylinders: Power unit of an engine (Categorical variable)
- displacement: Combined swept volume of the pistons inside the cylinders of an engine
(Continuous variable)
- horsepower: Power an engine produces(Continuous variable)
- weight: mass of a vehicle (Continuous variable)
- acceleration: the rate of change of the velocity of a car (Continuous variable)
- year: the time taken after a car is made (Discrete variable)
- origin: 1 is a car made in America, 2 in Europe and 3 in Asia or other part of the world (Categorical variable)

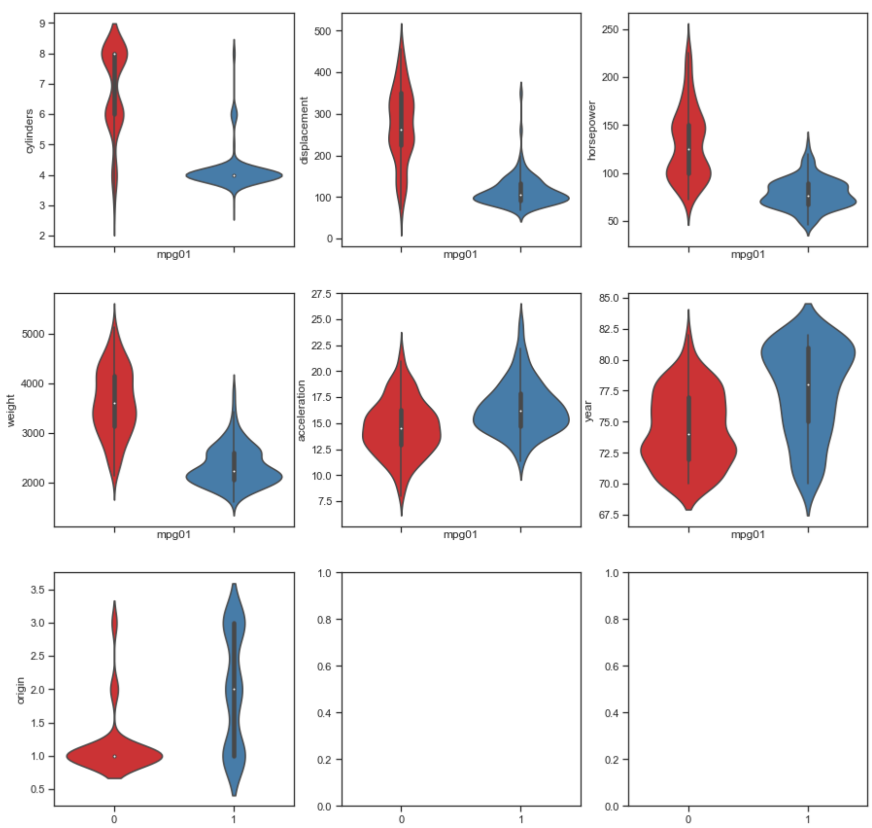
After feature explanation is presented, transformation of data sometimes is necessary for future data analysis. In this dataset, as an ouput, mpg feature was transformed into binary variable(0-1) based on the median of mpg. Classification methods such as Linear Discriminant analysis(LDA), Quadratic Disciminant analysis(QDA), Naive Bayes, Logistic Regression and K-nearest neighbors(KNN) will be utilized.

(b) Exploratory Data Analysis
Not every feature has influence on the output(mpg). As a result, feature engineering is crucial in this step. First, scatter plot will be provided.



Clearly, displacement, horsepower, weight and acceleration features have possible influence on mpg. From the figure, there are invisible lines that could separate two groups. As for origin, years and cylinders, they are uniform on mpg feature.
In order to further know the possible relationship, box and violin plots are performed. Box plots provide an insight about quantiles and violin plots offer different distribution among two groups. From below figures, weight, displacement, horsepower and acceleration have distinction between two groups of mpg. But noticed that categorical variables are not useful when there are shown as violin or box plots. Since from those two figures, there are no clear difference. Therefore, in this assignment, weight, displacement, horsepower and acceleration are selected as independent variables.


cylinders
displacement
horsepower
weight
acceleration
year
origin
mpg01


cylinders
displacement
horsepower
weight
acceleration
year
origin
mpg01

(c) Methods

1. **Linear Discriminant Analysis(LDA)**
    - Assumption: The two classes have a common covariance matrix. $\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log(\pi_k)$ where training data is used to estimate $\pi_k,\ \mu_k,\ \Sigma$
    - Validation: Testing examples are provided to confirm the trained model.
    - Statistical Package: scikit-learn is used for this Linear Discriminant Analysis.
2. **Quadratic Discriminant Analysis(QDA)**
    - Assumption: $\delta_k(x) = -\frac{1}{2}log(|\Sigma^k|) - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + log(\pi_k)$ where training data is used to estimate $\pi_k,\ \mu_k,\ \Sigma$
    - Validation: Testing examples are provided to confirm the trained model.
    - Statistical Package: scikit-learn is used for Quadratic Discriminant Analysis.
3. **Naive Bayes**
    - Assumption: Naive Bayes assumes each predictor is independent from each other. Then, $argmax_k(\pi_k\Pi_{j=1}^p f_{kj})$ where training data is used to estimate $f_{kj}(\cdot)$
    - Validation: Testing examples are provided to confirm the trained model.
    - Statistical Package: scikit-learn is used for Naive Bayes.
4. **Logistic Regression**
    - Assumption: Logistic regression uses probability of each case to build models. It utilizes logit function $g(\pi_i) = log(\frac{\pi_i}{1-\pi_i})$. And the model is $P(Y_i = 1) = \pi_i, P(Y_i = 0) = 1 - \pi_i$ and then $log(\frac{\pi_i}{1-\pi_i}) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_{p-1}x_{ip-1}$
    - Validation: Testing examples are provided to confirm the trained model.
    - Statistical Package: scikit-learn is used for Logistic Regression
5. **K-Nearest Neighbors(KNN)**
    - Assumption: KNN is memory-based, and does not need a model to be fit. It uses training examples to find k examples that are closet to a given point. That is, $d_i = ||x_i - x_0||$ where d_i is distance between x_i and x_0
    - Data Transformation: Since different values at distinctive scales lead to large difference and are hard to measure distance, scaled data is provided for KNN.
    - K values: error rate is calculated by choosing different k values. Then, the lowest error rate means that this k value is better than other k values.
    - Validation: Testing examples are provided to confirm the trained model. Furthermore, an error rate plot is provided to chose K.

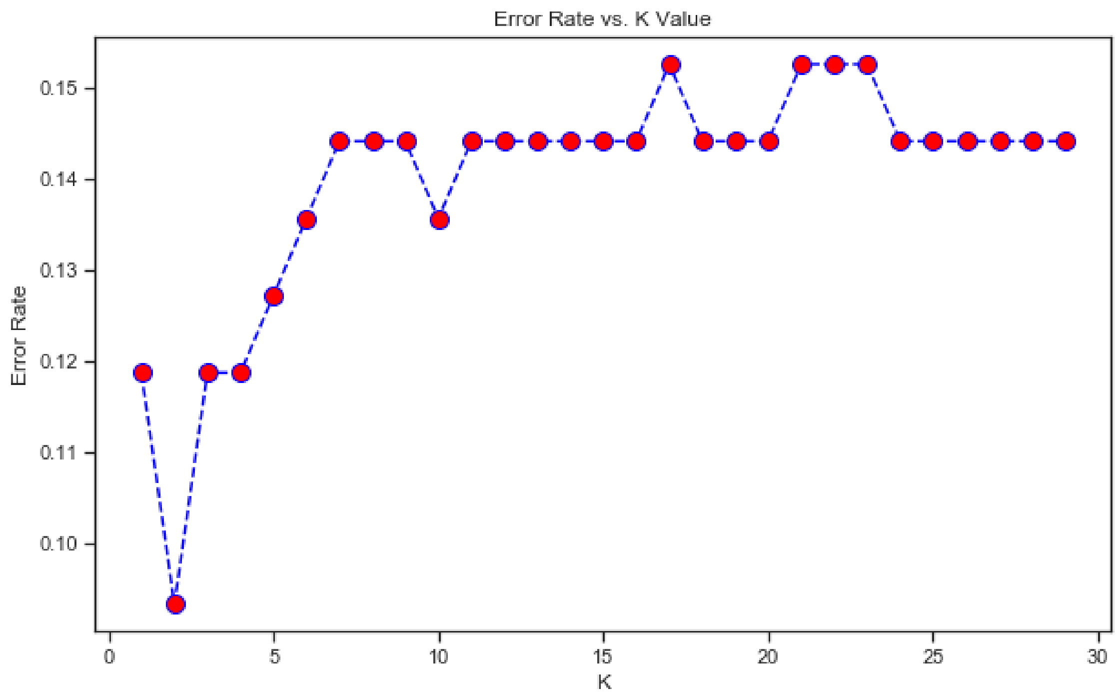
- Statistical Package: scikit-learn is used for K-Nearest Neighbors

(d) Results

The table is provided to show comparison among different classification methods.

Methods	Training Errors	Testing Errors
LDA	0.0839416	0.1610169
QDA 5	0.0839416	0.1694915
Naive Bayes	0.098540145	0.1694915
Logistic Regression	0.098540145	0.13559322

From the above table, LDA, QDA and Naive Bayes have similar testing errors. Since LDA and QDA are based on Naive Bayes, LDA assumes common variance and QDA assumes estimator is normal. For this dataset, there is no so much difference. However, when Logistic Regression is utilized, logistic regression performs better than previous methods based on testing errors. Last but not least, KNN is also a available choice by choosing right K. The error rate figure is provided as follows.



Obviously, $K = 2$ is better than other K values. As a result, a confusion matrix is offered and shows that accuracy is 91% which is really high. And, a trained model from KNN can perfectly classify one class (Usually, this is not possible).

```
WITH K=2
[[54 11]
 [ 0 53]]
              precision    recall  f1-score   support
           0       1.00      0.83      0.91        65
           1       0.83      1.00      0.91        53
```

```
    accuracy                           0.91       118
   macro avg       0.91      0.92      0.91       118
weighted avg       0.92      0.91      0.91       118
```

(e) Findings

After analyzing the dataset, we performed some of classification methods and we found QDA, LDA and Naive Bayes have similar results. The possible reason for this situation might be that we select features correctly or this dataset contains not too much data points. But, Logistic regression is better than previous methods, since they have stronger assumption than previous ones. Previous ones are based on Bayes but Logistic regression is based on probability. Logistic regression has linear model assumption via a link function. Moreover, KNN is also a good candidates, since its accuracy could be increased to 91% in this dataset. Although those methods have low testing errors, if large sample size or high-dimensional data is provided, the result might have significant difference among those methods.

(f) Python Code

0.1 Problem 1

In this problem, you are asked to write a report to summarize your analysis of the popular "Auto MPG" data set in the literature. Much research has been done to analyze this data set, and here the objective of our analysis is to predict whether a given car gets high or low gas mileage based 7 car attributes such as cylinders, displacement, horsepower, weight, acceleration, model year and origin.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

0.1.1 (a) Read Data

Data source is critical because we could not analyze artificial data. So we will briefly introduce data sources. This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association

[2]:
```
df = pd.read_csv("Auto.csv")
```

[3]:
```
df.head()
```

[3]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  ␣
↪\
0  18.0          8         307.0         130    3504          12.0    70
1  15.0          8         350.0         165    3693          11.5    70
2  18.0          8         318.0         150    3436          11.0    70
3  16.0          8         304.0         150    3433          12.0    70
4  17.0          8         302.0         140    3449          10.5    70

   origin
0       1
1       1
2       1
3       1
4       1
```

[4]:
```
df.shape
```

[4]: (392, 8)

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 8 columns):
mpg             392 non-null float64
cylinders       392 non-null int64
displacement    392 non-null float64
horsepower      392 non-null int64
weight          392 non-null int64
acceleration    392 non-null float64
year            392 non-null int64
origin          392 non-null int64
dtypes: float64(3), int64(5)
memory usage: 24.6 KB
```

Dataset Description

- mpg: Miles per gallon (Continuous variable)
- cylinders: Power unit of an engine (Categorical variable)
- displacement: Combined swept volume of the pistons inside the cylinders of an engine(Continuous variable)
- horsepower: Power an engine produces(Continuous variable)
- weight: mass of a vehicle (Continuous variable)
- acceleration: the rate of change of the velocity of a car (Continuous variable)
- year: the time taken after a car is made (Discrete variable)
- origin: 1 is a car made in america, 2 in europe and 3 in asia or other part of the world (Categorical variable)

0.1.2 (b) Cleaning Dataset

Create a binary variable, mpg01, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median.

[6]:
```
median = np.median(df.loc[:, 'mpg'])
median
```

[6]: 22.75

[7]:
```
# create a method for mapping
def truefalse(col):
    if col <= 22.75:
        return 0
```

```
    else:
        return 1
```

[8]:
```
df['mpg01'] = df.loc[:, 'mpg'].map(truefalse)
df.head()
```

[8]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  ␣
↪\
0  18.0          8         307.0         130    3504          12.0    70
1  15.0          8         350.0         165    3693          11.5    70
2  18.0          8         318.0         150    3436          11.0    70
3  16.0          8         304.0         150    3433          12.0    70
4  17.0          8         302.0         140    3449          10.5    70

   origin  mpg01
0       1      0
1       1      0
2       1      0
3       1      0
4       1      0
```

[9]:
```
#replace column mpg with mpg01
df_new = df.drop(columns = 'mpg')
df_new.head()
#rearrange column
cols = df_new.columns.tolist()
cols = cols[-1:] + cols[:-1]
#rearranged columns
df_new = df_new[cols]
df_new.head()
```

[9]:
```
   mpg01  cylinders  displacement  horsepower  weight  acceleration  year␣
↪ \
0      0          8         307.0         130    3504          12.0    70
1      0          8         350.0         165    3693          11.5    70
2      0          8         318.0         150    3436          11.0    70
3      0          8         304.0         150    3433          12.0    70
4      0          8         302.0         140    3449          10.5    70

   origin
0       1
1       1
2       1
3       1
```

```
4          1
```

0.1.3 (c) Exploratory Data Analysis

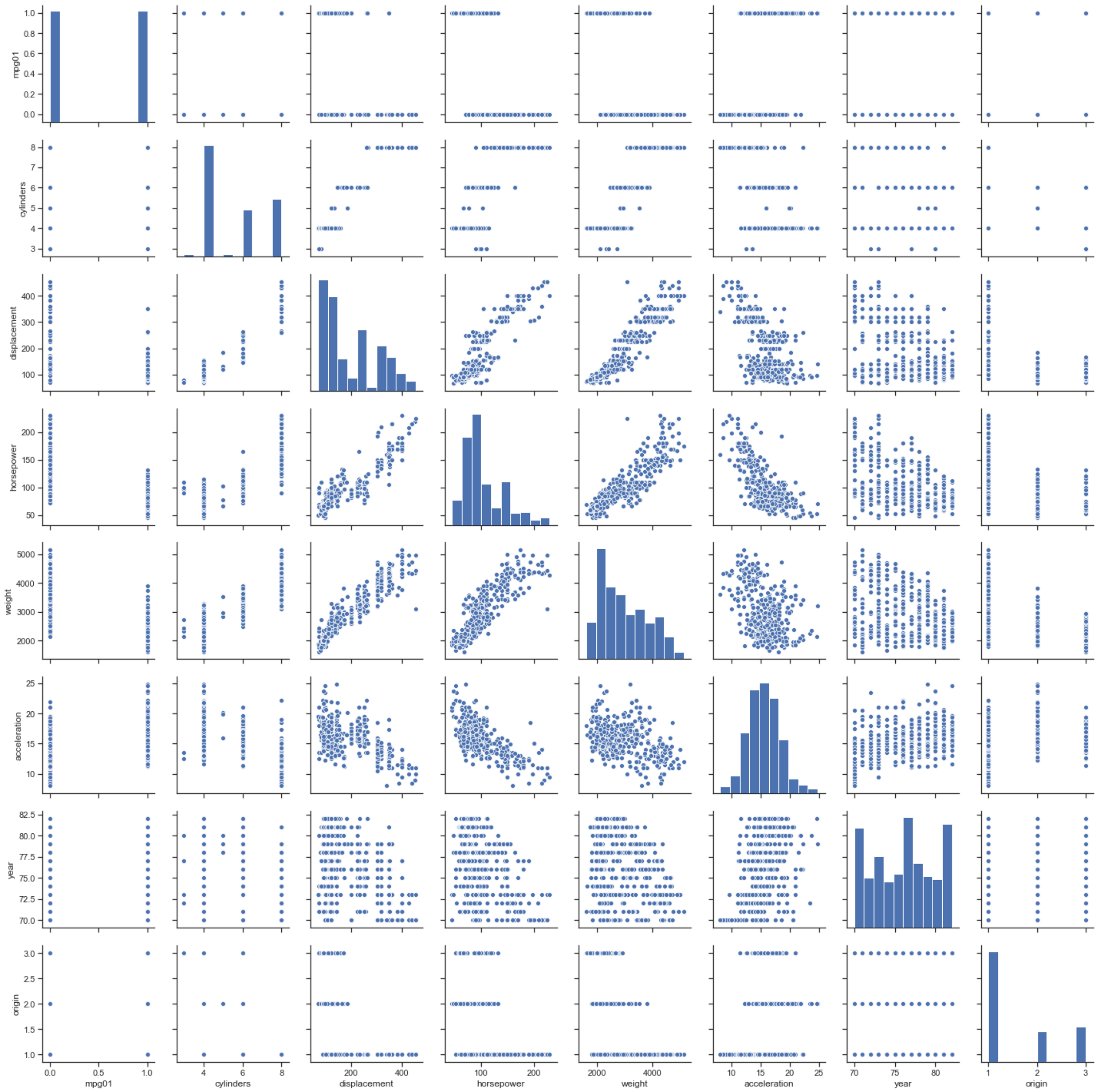
- Explore the data graphically in order to investigate the association between mpg01 and the other features.
- Which of the other features seem most likely to be useful in predicting mpg01?
- Scatterplots and boxplots may be useful tools to answer this question. Describe your findings.

[10]:
```
df_new.describe()
```

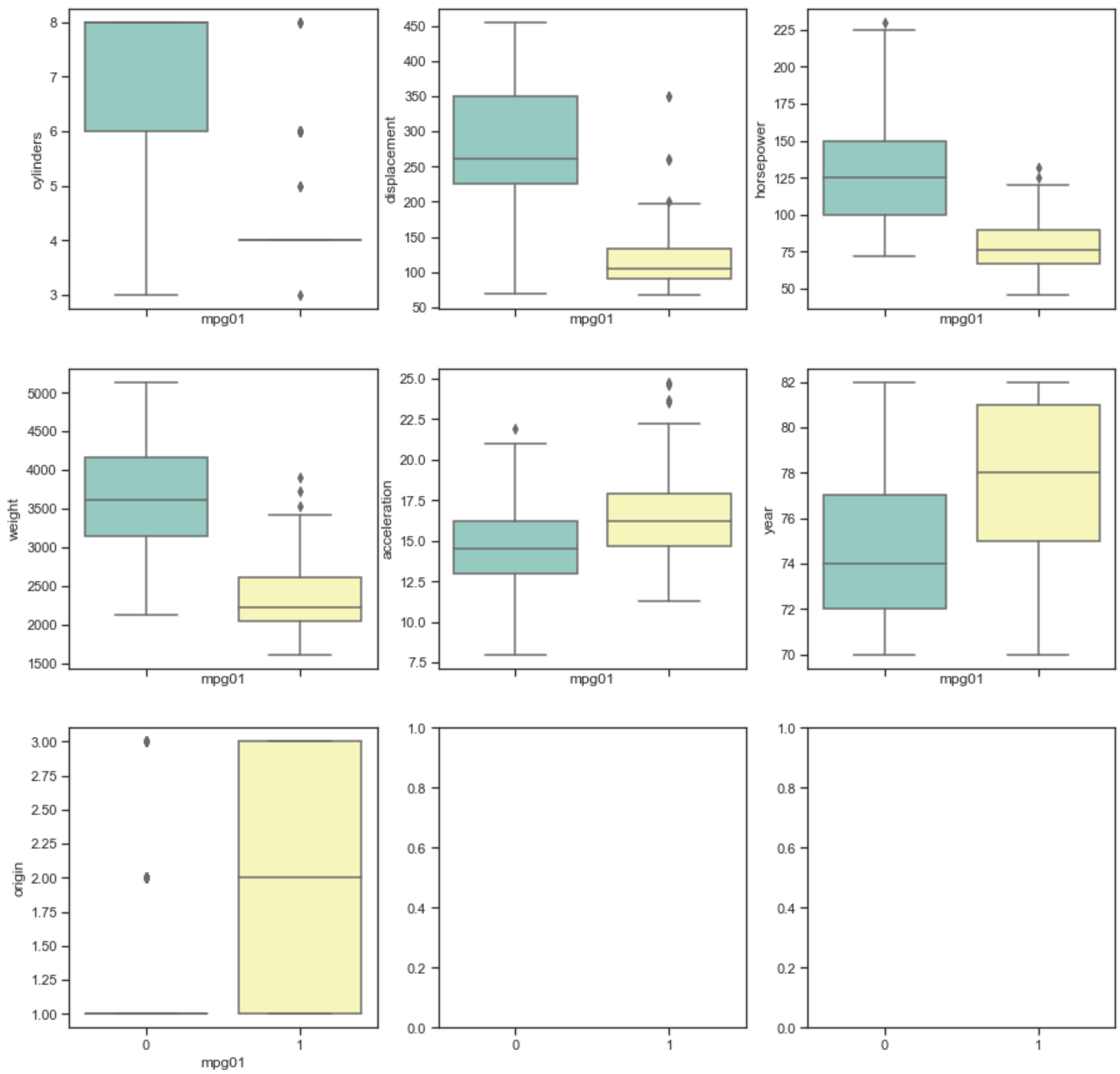
[10]:
```
            mpg01   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean     0.500000    5.471939    194.411990  104.469388  2977.584184
std      0.500639    1.705783    104.644004   38.491160   849.402560
min      0.000000    3.000000     68.000000   46.000000  1613.000000
25%      0.000000    4.000000    105.000000   75.000000  2225.250000
50%      0.500000    4.000000    151.000000   93.500000  2803.500000
75%      1.000000    8.000000    275.750000  126.000000  3614.750000
max      1.000000    8.000000    455.000000  230.000000  5140.000000

       acceleration        year      origin
count    392.000000  392.000000  392.000000
mean      15.541327   75.979592    1.576531
std        2.758864    3.683737    0.805518
min        8.000000   70.000000    1.000000
25%       13.775000   73.000000    1.000000
50%       15.500000   76.000000    1.000000
75%       17.025000   79.000000    2.000000
max       24.800000   82.000000    3.000000
```

[11]:
```
sns.set(style="ticks", color_codes=True)
sns.pairplot(df_new)
```

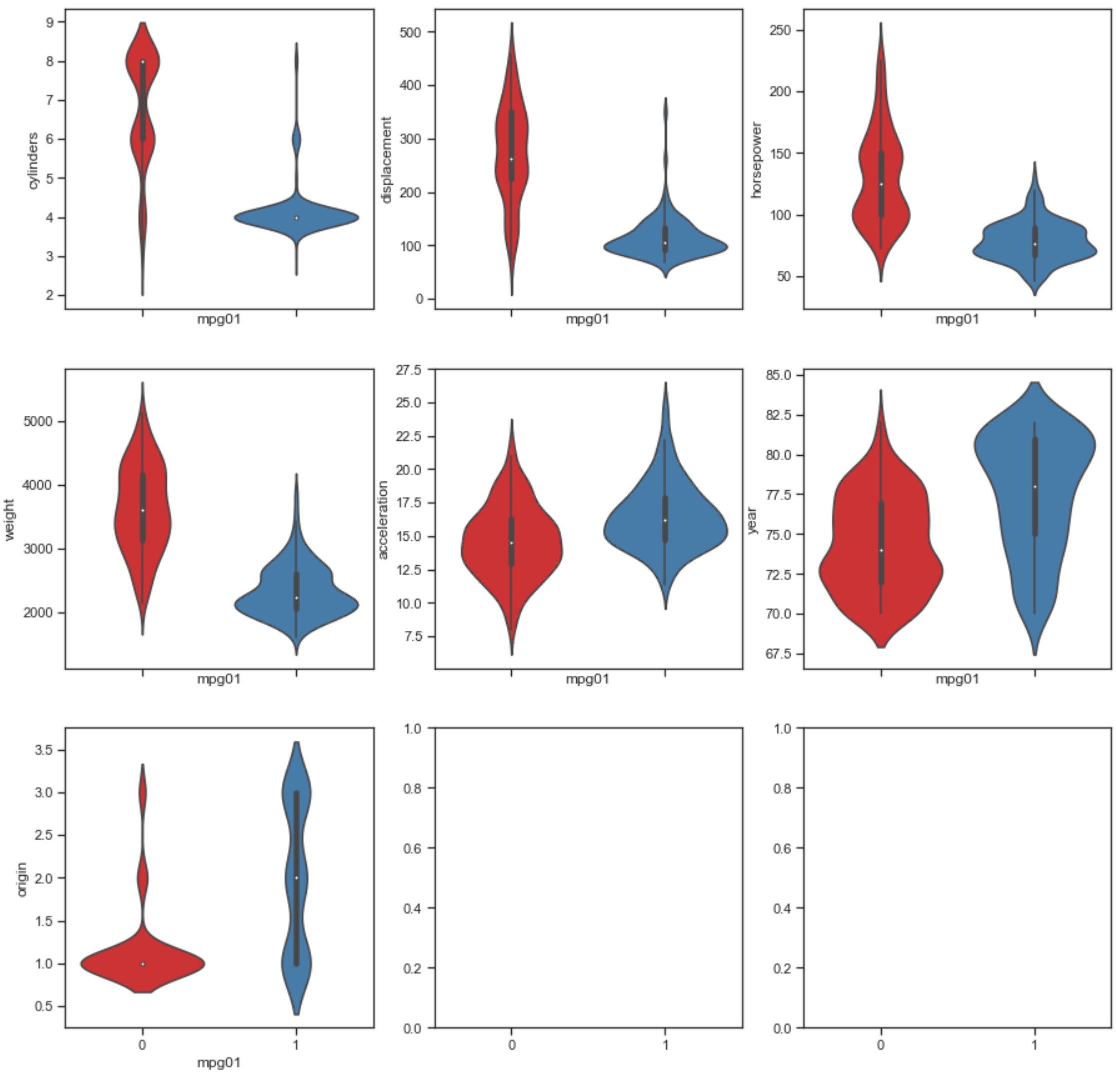
[11]:
```
<seaborn.axisgrid.PairGrid at 0x10d445250>
```



[12]:
```
f, axes = plt.subplots(3, 3, figsize=(15, 15), sharex=True)

cols = df_new.columns
row = 0
count = 0
for i in range(7):
    sns.boxplot(x = "mpg01", y = df_new.columns[i + 1], data=df_new,␣
 ↪palette="Set3", ax = axes[row, (i + 3) % 3])
    count += 1
    if (count == 3):
        count = 0
        row += 1
```



[13]:
```
f, axes = plt.subplots(3, 3, figsize=(15, 15), sharex=True)

cols = df_new.columns
row = 0
count = 0
for i in range(7):
    sns.violinplot(x = "mpg01", y = df_new.columns[i + 1],
 ↪data=df_new,palette='Set1', ax = axes[row, (i + 3) % 3])
    count += 1
    if (count == 3):
        count = 0
        row += 1
```



0.1.4 (d) Dataset Split

Now we will split our dataset into training and testing examples. We use the train_test_split method in scikit-learn package to split the dataset. For this method, it will randomly select some of data points to derive the training and testing dataset.

In scikit-learn package, the method's description is, *"Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner."*

[14]:
```
from sklearn.model_selection import train_test_split
X = df_new.drop(['mpg01'], axis = 1)
y = df_new["mpg01"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state = 42)
```

0.1.5 (e) Traing Model

Perform the following classification methods on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (c). What is the test error of the model obtained? 1. LDA 2. QDA 3. Naive Bayes 4. Logistic Regression 5. KNN with several values of K. Use only the variables that seemed most associated with mpg01 in (c). Which value of K seems to perform the best on this data set?

[18]:
```
# create dataframe that some of features have association with mpg
X_train = X_train.loc[:,['displacement', 'horsepower', 'weight',␣
 ↪'acceleration']]
X_test = X_test.loc[:,['displacement', 'horsepower', 'weight',␣
 ↪'acceleration']]
```

LDA

[59]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf = LinearDiscriminantAnalysis()
clf.fit(X_train, y_train)
y_pred_lda = clf.fit(X_train, y_train).predict(X_test)

trainError = clf.fit(X_train, y_train).predict(X_train)
print("LDA Training Error:", np.mean(trainError != y_train))
print("LDA Testing Error:", np.mean(y_pred_lda != y_test))
```

```
LDA Training Error: 0.08394160583941605
LDA Testing Error: 0.16101694915254236
```

QDA

[60]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
qda = QuadraticDiscriminantAnalysis(store_covariance=True)
y_pred_qda = qda.fit(X_train, y_train).predict(X_test)

QtrainError = qda.fit(X_train, y_train).predict(X_train)
print("QDA Training Error:", np.mean(QtrainError != y_train))
print("QDA Testing Error:", np.mean(y_pred_qda != y_test))
```

```
QDA Training Error: 0.08394160583941605
QDA Testing Error: 0.1694915254237288
```

Naive Bayes

[61]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred_NB = gnb.fit(X_train, y_train).predict(X_test)

NBtrainError = gnb.fit(X_train, y_train).predict(X_train)
print("Naive Baysès Training Error:", np.mean(NBtrainError != y_train))
print("Naive Bayses Testing Error:", np.mean(y_pred_NB != y_test))
```

```
Naive Bayses Training Error: 0.09854014598540146
Naive Bayses Testing Error: 0.1694915254237288
```

Logistic Regression

[62]:
```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression(random_state=0).fit(X_train, y_train)
lgr_y_pred = lgr.predict(X_test)

LgrError = lgr.fit(X_train, y_train).predict(X_train)
print("Logistic Training Error:", np.mean(LgrError != y_train))
print("Logistic Testing Error:", np.mean(lgr_y_pred != y_test))
```

```
Logistic Training Error: 0.09854014598540146
Logistic Testing Error: 0.13559322033898305
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:
 ↪432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:
 ↪432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

KNN

[63]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
scaled_features = scaler.transform(X_train)
df_feat = pd.DataFrame(scaled_features, columns = X_train.columns)
df_feat.head()
```

[63]:

	displacement	horsepower	weight	acceleration
0	-0.854863	-0.285913	-0.725403	0.342749
1	0.031360	-0.522352	-0.477029	0.167473
2	-0.748902	-0.338455	-0.659727	-0.183081
3	0.021727	-0.390997	-0.404188	-0.183081
4	0.513002	-0.128287	0.417358	0.518026

[71]:
```
scaler_test = StandardScaler()
scaler_test.fit(X_test)
scaled_features_test = scaler.transform(X_test)
df_feat_test = pd.DataFrame(scaled_features_test, columns = X_test.
 ↪columns)
df_feat_test.head()
```

[71]:

	displacement	horsepower	weight	acceleration
0	-0.970458	-0.942688	-0.952283	0.868580
1	-0.729636	0.265778	-0.228654	0.062306
2	-1.018622	-1.179127	-1.416791	0.307694
3	-1.018622	-0.916417	-1.231705	1.744964
4	-0.546612	-0.496081	-0.234625	0.027251

[86]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_1 = KNeighborsClassifier(n_neighbors=1)
knn_1.fit(df_feat,y_train)
pred_knn = knn.predict(df_feat_test)
```

[87]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred_knn))
```

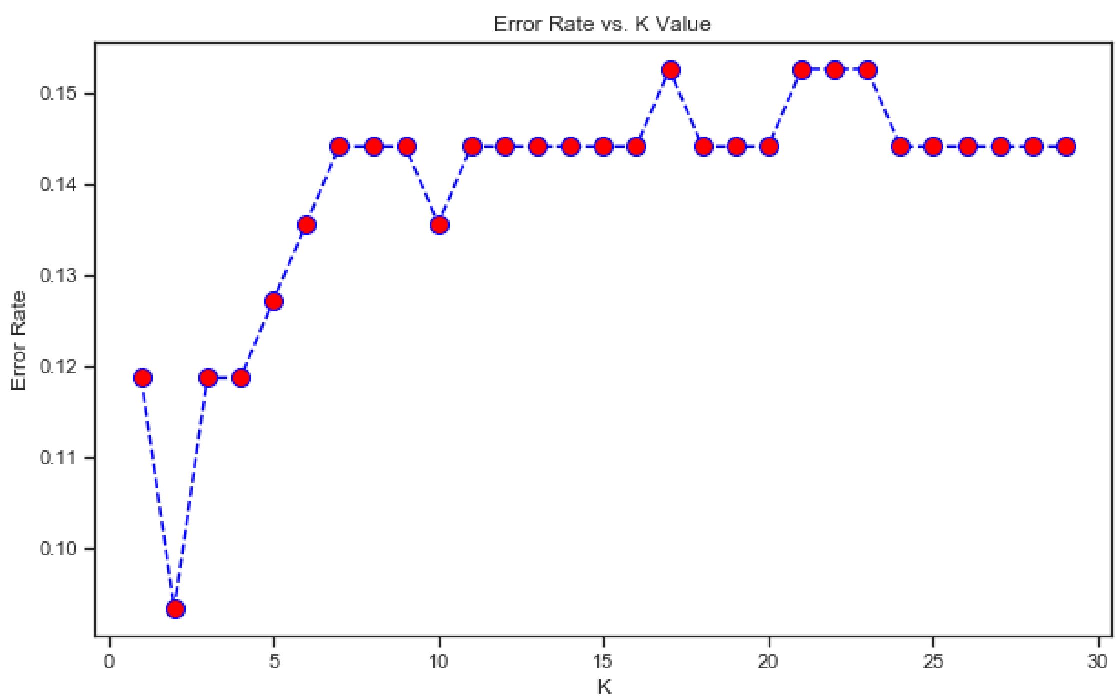
```
[[51 14]
 [ 0 53]]
```

[88]:
```
print(classification_report(y_test,pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        65
           1       0.79      1.00      0.88        53

    accuracy                           0.88       118
   macro avg       0.90      0.89      0.88       118
weighted avg       0.91      0.88      0.88       118
```

[93]:
```
error_rate = []

for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(df_feat,y_train)
    pred_i = knn.predict(df_feat_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,30),error_rate,color='blue', linestyle='dashed',␣
 ↪marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[93]:
```
Text(0, 0.5, 'Error Rate')
```



[90]:
```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn1 = KNeighborsClassifier(n_neighbors=1)

knn1.fit(df_feat,y_train)
pred_k1 = knn.predict(df_feat_test)
```

```
print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred_k1))
print('\n')
print(classification_report(y_test,pred_k1))
```

```
WITH K=1


[[49 16]
 [ 1 52]]


              precision    recall  f1-score   support

           0       0.98      0.75      0.85        65
           1       0.76      0.98      0.86        53

    accuracy                           0.86       118
   macro avg       0.87      0.87      0.86       118
weighted avg       0.88      0.86      0.86       118
```

[91]:
```
# NOW WITH K=2
knn_2 = KNeighborsClassifier(n_neighbors=2)

knn_2.fit(df_feat,y_train)
pred_2 = knn_2.predict(df_feat_test)

print('WITH K=2')
print('\n')
print(confusion_matrix(y_test,pred_2))
print('\n')
print(classification_report(y_test,pred_2))
```

```
WITH K=2


[[54 11]
 [ 0 53]]


              precision    recall  f1-score   support
```

```
           0       1.00      0.83      0.91        65
           1       0.83      1.00      0.91        53

    accuracy                           0.91       118
   macro avg       0.91      0.92      0.91       118
weighted avg       0.92      0.91      0.91       118
```