In [1]:
```
#pip install ISLP
```

In [2]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
summarize)
```

In [3]:
```
from ISLP import confusion_table
from ISLP.models import contrast
from sklearn.discriminant_analysis import \
(LinearDiscriminantAnalysis as LDA, QuadraticDiscriminantAnalysis as QDA)
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [4]:
```
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize,
                         poly)
```

In [5]:
```
Auto = load_data("Auto")
```

14a.

In [6]:
```
mpg01=(Auto['mpg']>Auto['mpg'].median()).astype(int)
mpg01
```

Out[6]:

name	mpg
chevrolet chevelle malibu	0
buick skylark 320	0
plymouth satellite	0
amc rebel sst	0
ford torino	0
...	...
ford mustang gl	1
vw pickup	1
dodge rampage	1
ford ranger	1
chevy s-10	1

392 rows × 1 columns

dtype: int64

In [7]:

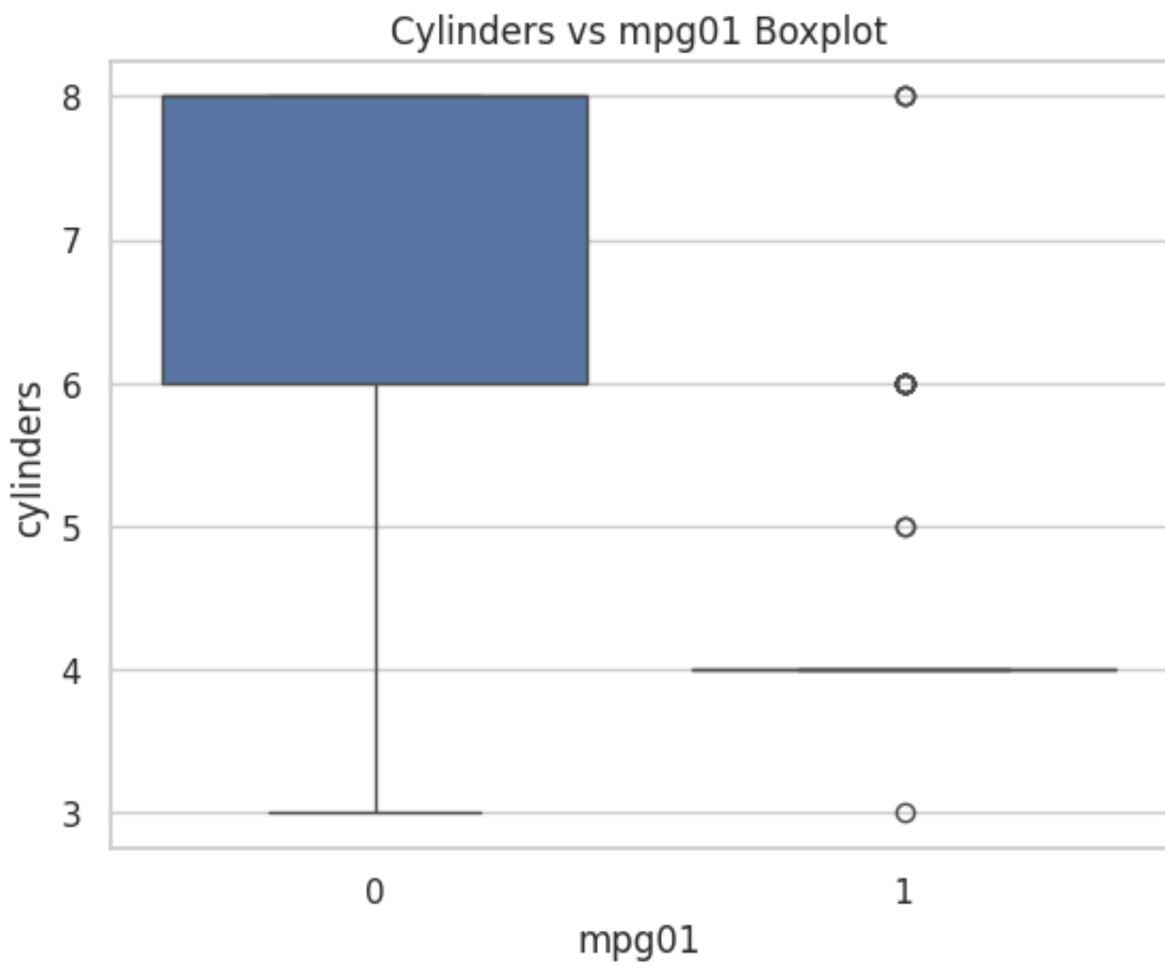
```
Auto['mpg01'] = mpg01
```

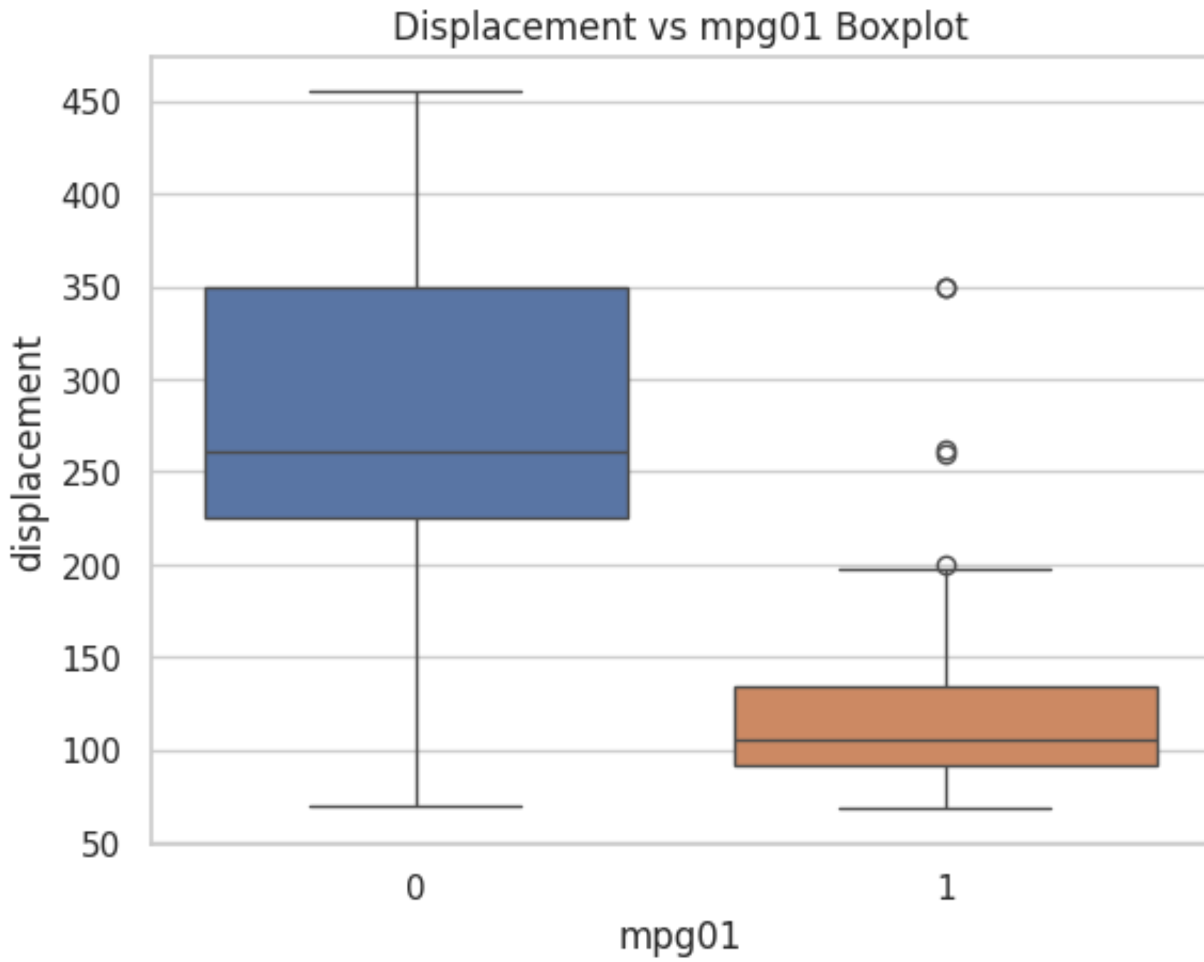
14b.

In [8]:

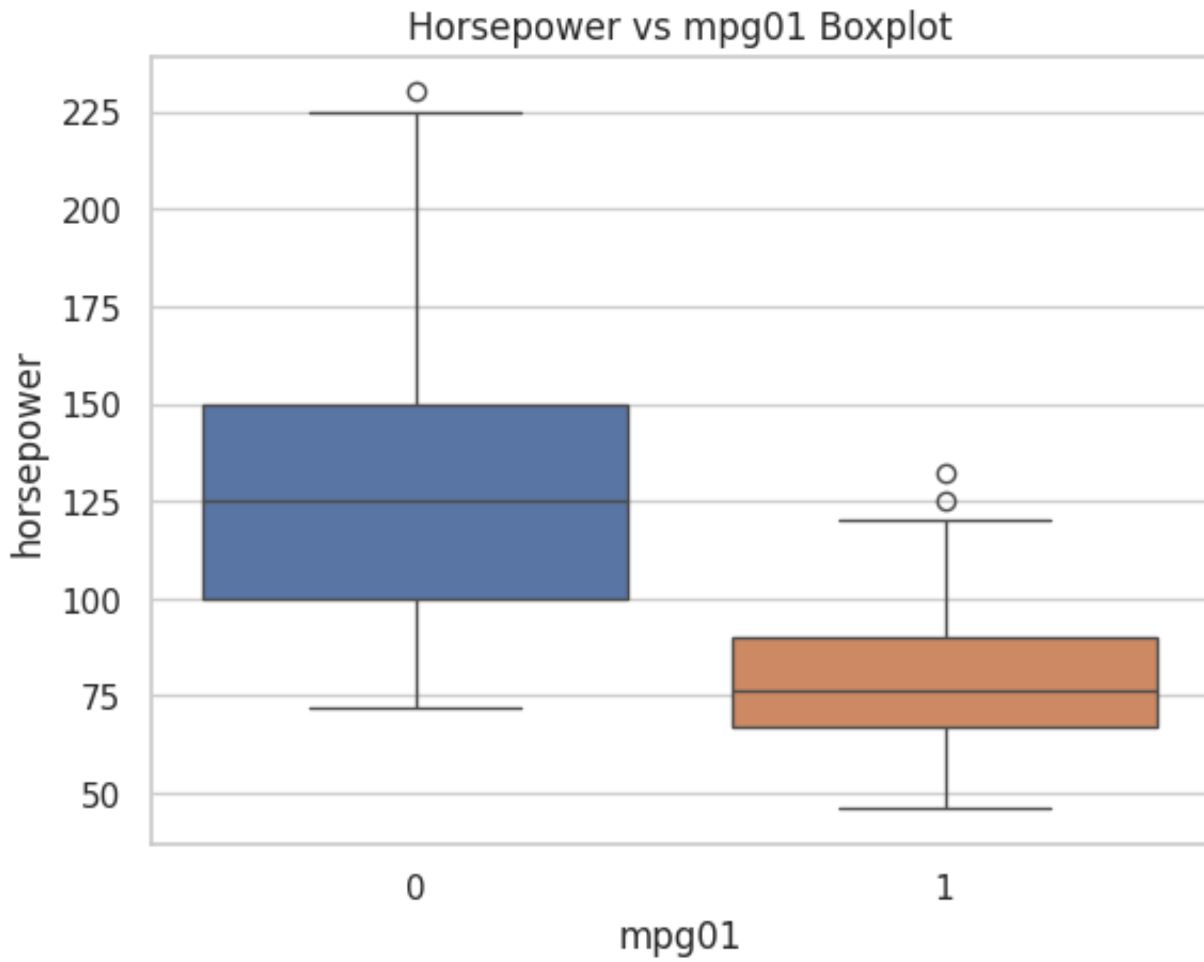
```
#Originally generated by CHATgpt
import matplotlib.pyplot as plt
import seaborn as sns
# Set plot style
sns.set(style="whitegrid")

# Convert 'origin' to categorical with labels
Auto['origin'] = Auto['origin'].replace({1: 'American', 2: 'European', 3: 'J

# Plot 1: Cylinders vs mpg01 Boxplot
g1 = sns.boxplot(data=Auto, x='mpg01', y='cylinders', hue='mpg01')
g1.legend_.remove()
plt.title('Cylinders vs mpg01 Boxplot')
plt.show()

# Plot 2: Displacement vs mpg01 Boxplot
g2 = sns.boxplot(data=Auto, x='mpg01', y='displacement', hue='mpg01')
g2.legend_.remove()
plt.title('Displacement vs mpg01 Boxplot')
plt.show()
```

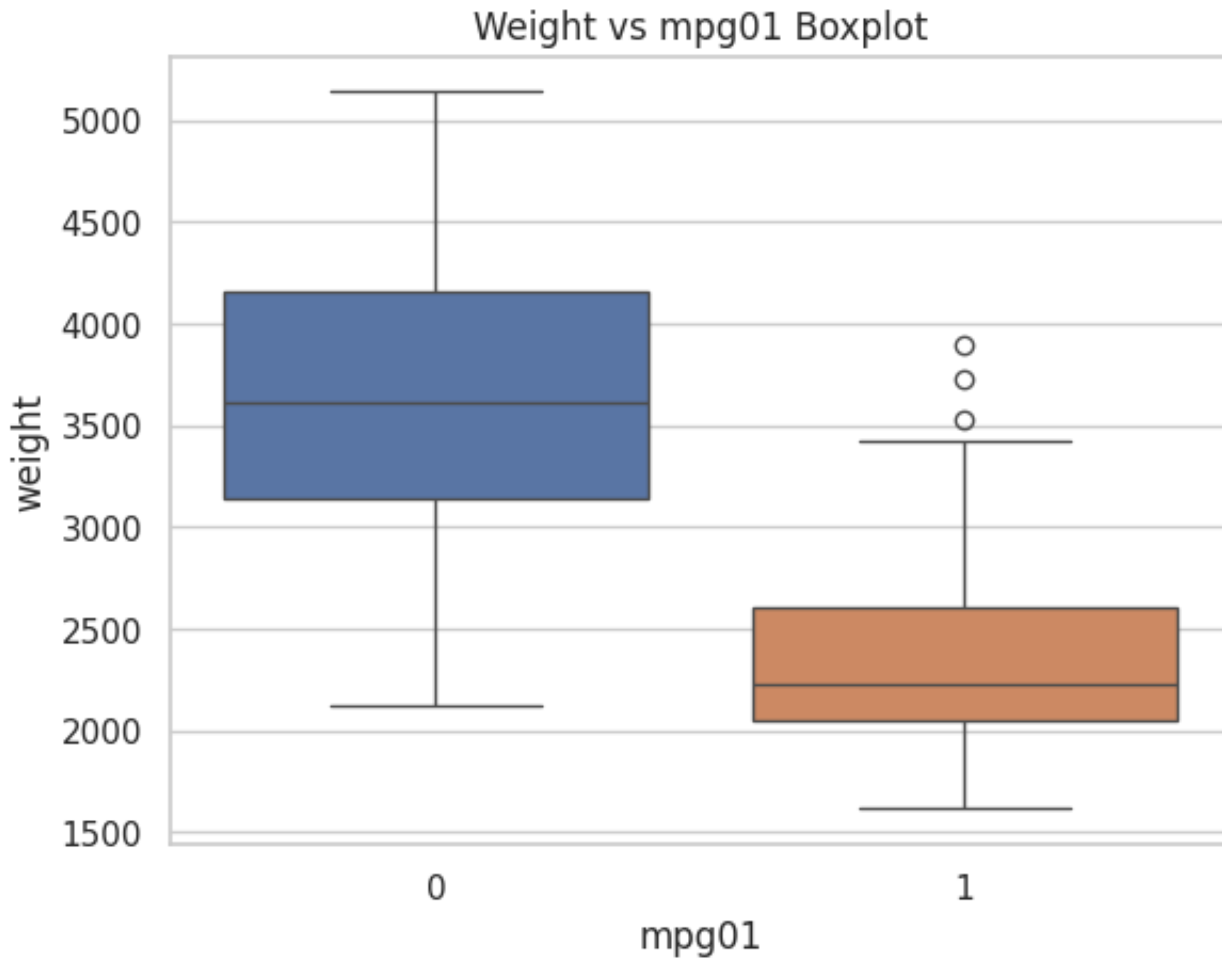
```python
# Plot 3: Horsepower vs mpg01 Boxplot
g3 = sns.boxplot(data=Auto, x='mpg01', y='horsepower', hue='mpg01')
g3.legend_.remove()
plt.title('Horsepower vs mpg01 Boxplot')
plt.show()

# Plot 4: Weight vs mpg01 Boxplot
g4 = sns.boxplot(data=Auto, x='mpg01', y='weight', hue='mpg01')
g4.legend_.remove()
plt.title('Weight vs mpg01 Boxplot')
plt.show()

# Plot 5: Acceleration vs mpg01 Boxplot
g5 = sns.boxplot(data=Auto, x='mpg01', y='acceleration', hue='mpg01')
g5.legend_.remove()
plt.title('Acceleration vs mpg01 Boxplot')
plt.show()

# Plot 6: Year vs mpg01 - Boxplot
g6 = sns.boxplot(data=Auto, x='mpg01', y='year', hue='mpg01')
g6.legend_.remove()
plt.title('Year vs mpg01 Boxplot')
plt.show()

# Plot 7: Origin vs mpg01 Bar plot
g7 = sns.histplot(data=Auto, x='origin', hue='mpg01', multiple='fill', shrir
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: '{:.0%}'.
plt.ylabel('')
plt.title('Origin vs mpg01 Bar plot')
plt.show()
```

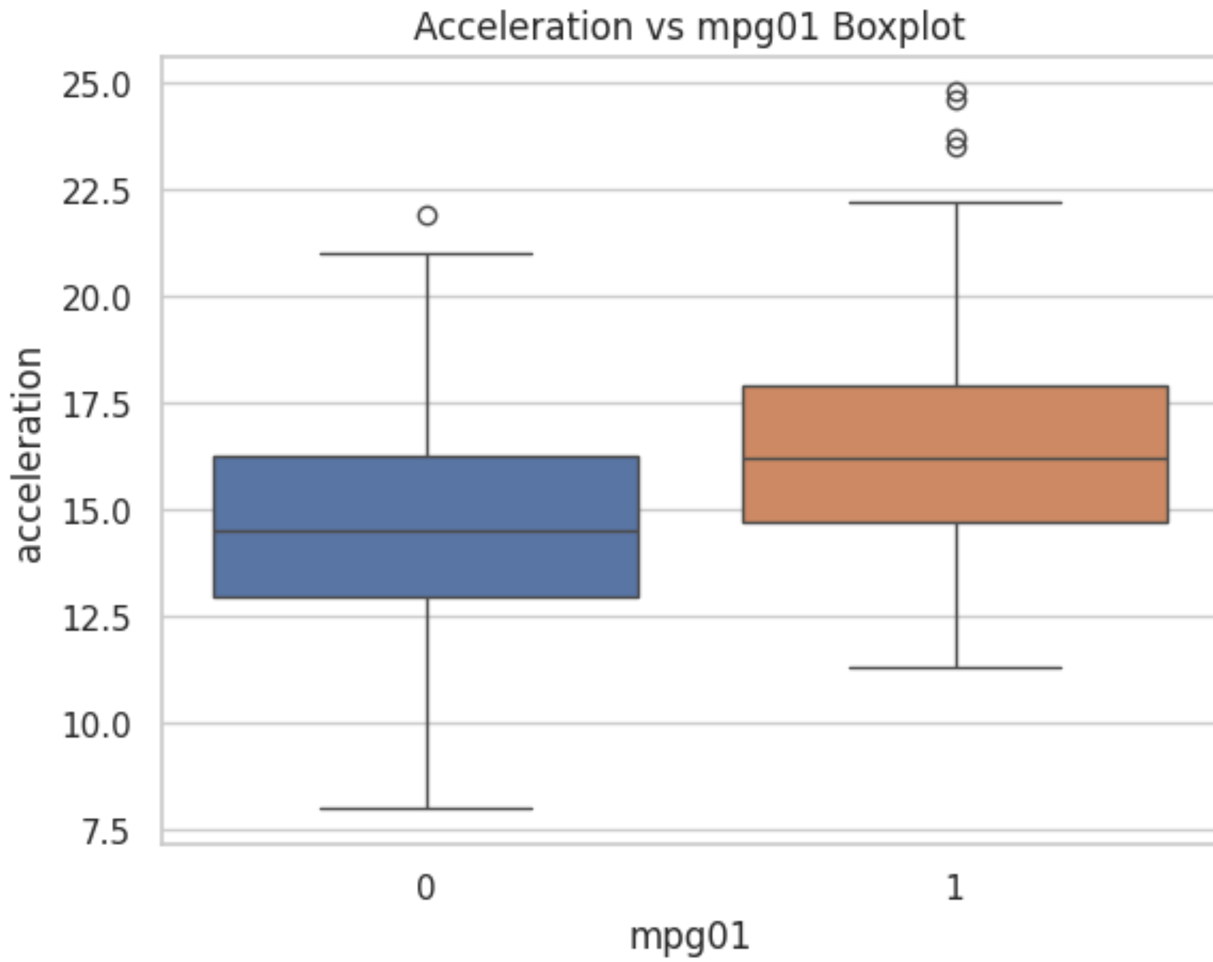

Cylinders vs mpg01 Boxplot
cylinders
8
7
6
5
4
3
0
1
mpg01



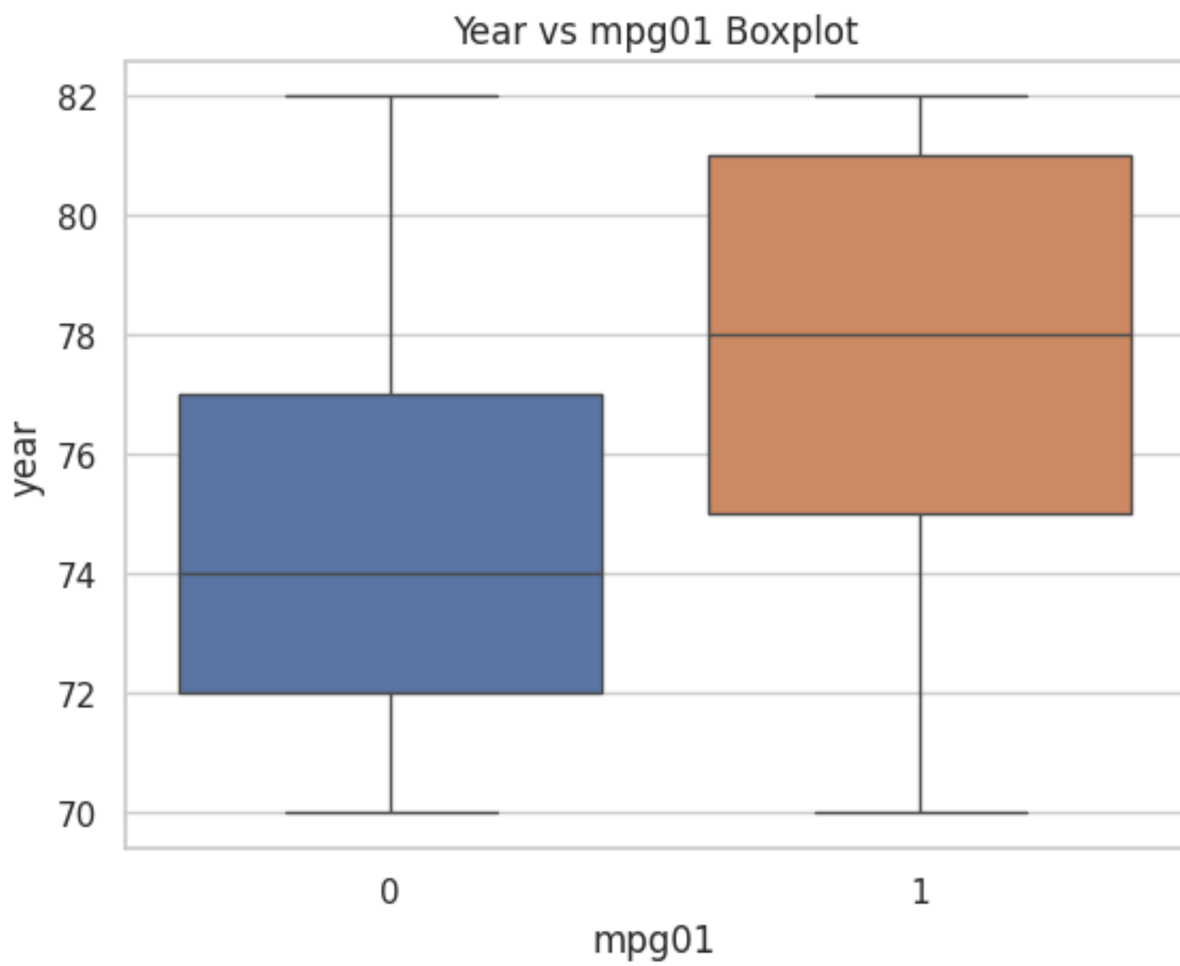
Displacement vs mpg01 Boxplot
displacement
450
400
350
300
250
200
150
100
50
0
1
mpg01


Horsepower vs mpg01 Boxplot
horsepower
225
200
175
150
125
100
75
50
0
1
mpg01

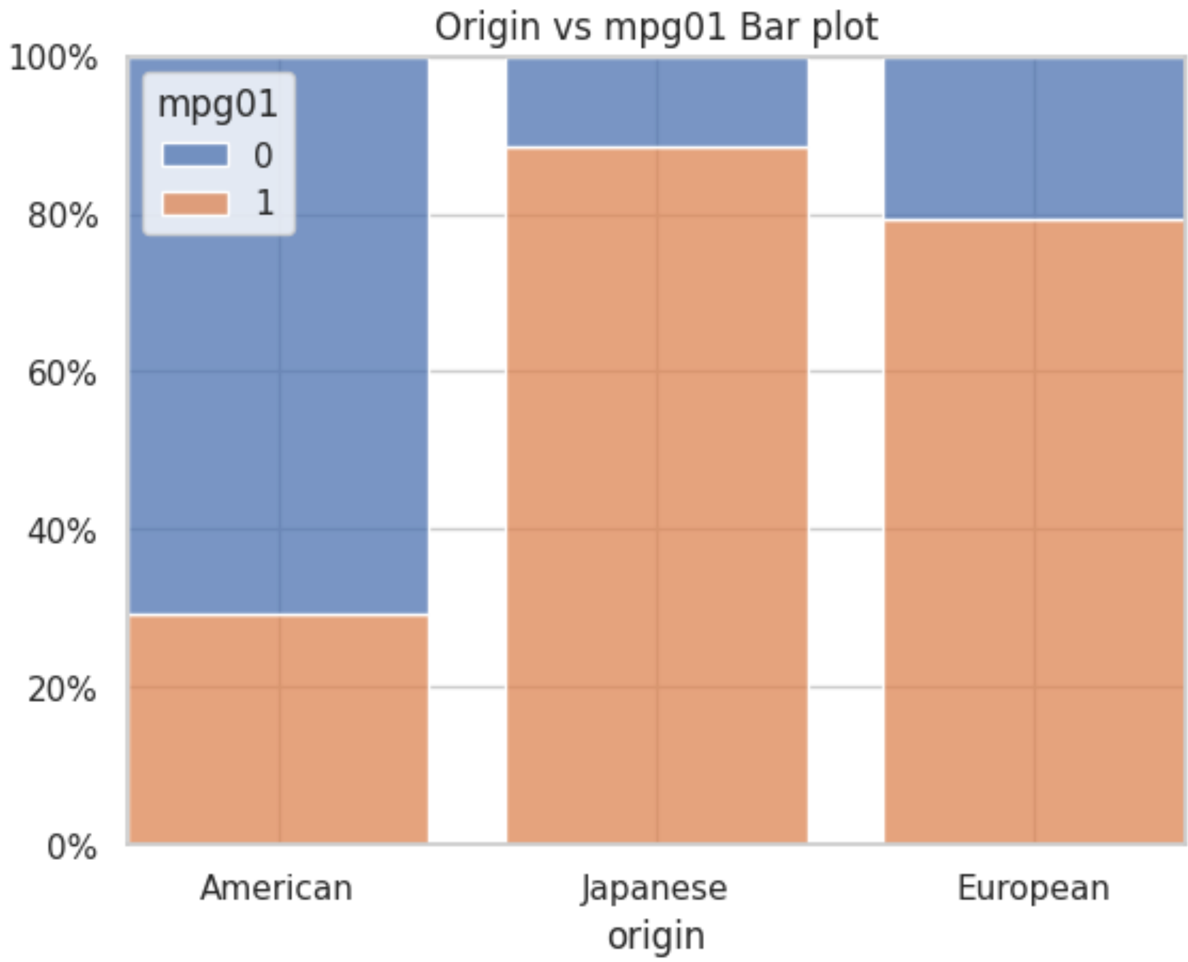

Weight vs mpg01 Boxplot
weight
5000
4500
4000
3500
3000
2500
2000
1500
0
1
mpg01



Acceleration vs mpg01 Boxplot
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
acceleration
0
1
mpg01



Year vs mpg01 Boxplot
82
80
78
76
74
72
70
year
0
1
mpg01



Explanation: It looks like all of the other features are useful in predicting mpg01, but I would say cylinders, displacement, weight, and horsepower are more powerful than the others.

14c.

In [9]:
```
np.random.seed(123)
train, test = train_test_split(Auto, test_size=0.5, random_state=123)
```

14d.

In [10]:
```
# Prepare features and mpg01
X_train = train[['cylinders', 'displacement', 'weight', 'horsepower']]
y_train = train['mpg01']
X_test = test[['cylinders', 'displacement', 'weight', 'horsepower']]
```

In [11]:
```
lda = LDA(store_covariance=True)
lda.fit(X_train, y_train)
predicted_lda = lda.predict(X_test)

# Calculate test error
print(np.mean(predicted_lda != test['mpg01']))
```

0.10204081632653061

14e.

In [12]:
```
qda = QDA(store_covariance=True)
qda.fit(train[['cylinders', 'displacement', 'weight', 'horsepower']], train[
predicted_qda = qda.predict(X_test)
print(np.mean(predicted_qda != test['mpg01']))
```

0.09693877551020408

14f.

In [13]:
```
logit = LogisticRegression(C=1e10, solver='liblinear')
logit.fit(X_train, y_train)
logit_pred = logit.predict(X_test)
print(np.mean(logit_pred != test['mpg01']))
```

0.12755102040816327

14g.

In [14]:
```
NB = GaussianNB()
NB.fit(X_train, y_train)
NB_pred=NB.predict(X_test)
print(np.mean(NB_pred != test['mpg01']))
```

0.10204081632653061

14h.

In [15]:
```
from sklearn.model_selection import GridSearchCV
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print(np.mean(knn_pred != test['mpg01']))
```

0.11734693877551021

The best value of k I could find was 7 since the error starts inccreasing after k=7.

Test error for k=1: 0.16326530612244897

Test error for k=8: 0.12244897959183673

16.

In [16]:
```
Boston = load_data("Boston")
crim01=(Boston['crim']>Boston['crim'].median()).astype(int)
crim01
```

Out[16]:

	crim
0	0
1	0
2	0
3	0
4	0
...	...
501	0
502	0
503	0
504	0
505	0

506 rows × 1 columns

dtype: int64

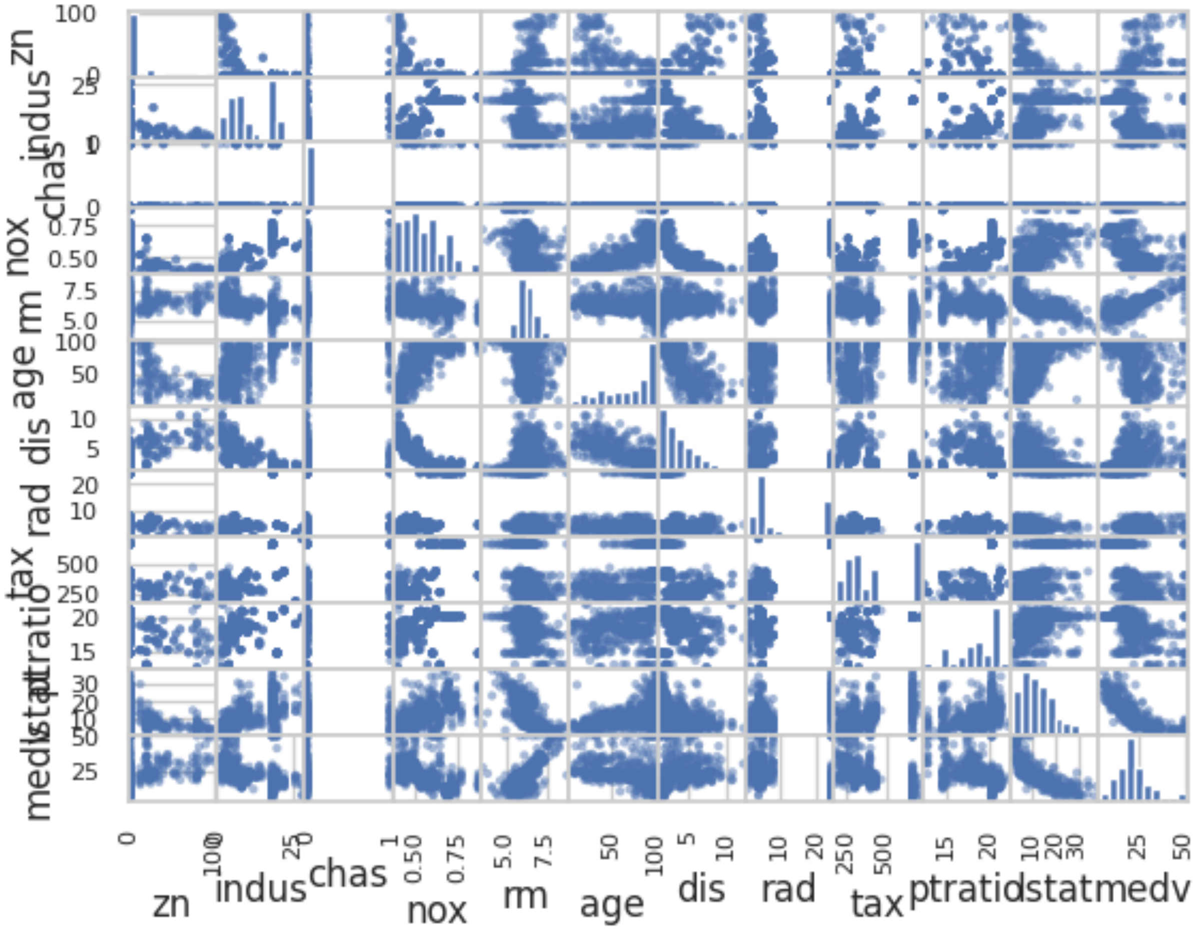
In [17]:
```
Boston['crim01'] = crim01
```

In [18]:
```
np.random.seed(234)
train, test = train_test_split(Boston, test_size=0.5, random_state=234)
```

In [19]:
```
Boston.columns
```

Out[19]:
```
Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'ta
x',
       'ptratio', 'lstat', 'medv', 'crim01'],
      dtype='object')
```

In [20]:
```
pd.plotting.scatter_matrix(Boston[['zn', 'indus', 'chas', 'nox', 'rm', 'age'
       'ptratio', 'lstat', 'medv']]);
```



In [21]:

```
# Prepare features and crim01
BosAll=['zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptr
X_train = train[BosAll]
y_train = train['crim01']
X_test = test[BosAll]
#logit
logit = LogisticRegression(C=1e10, solver='liblinear')
logit.fit(X_train, y_train)
logit_pred = logit.predict(X_test)
print("Logit:"+ str(np.mean(logit_pred != test['crim01'])))
#lda
lda = LDA(store_covariance=True)
lda.fit(X_train, y_train)
predicted_lda = lda.predict(X_test)
print("LDA:"+str(np.mean(predicted_lda != test['crim01'])))
#NB
NB = GaussianNB()
NB.fit(X_train, y_train)
NB_pred=NB.predict(X_test)
print("NB:"+str(np.mean(NB_pred != test['crim01'])))
#KNN
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print("KNN:"+str(np.mean(knn_pred != test['crim01'])))
```

```
Logit:0.11462450592885376
LDA:0.18181818181818182
NB:0.2015810276679842
KNN:0.07114624505928854
```

In [22]:

```
# Prepare features and crim01
BosSub=['zn', 'indus', 'rm', 'dis', 'rad', 'tax','nox']
X_train = train[BosSub]
y_train = train['crim01']
X_test = test[BosSub]
#logit
logit = LogisticRegression(C=1e10, solver='liblinear')
logit.fit(X_train, y_train)
logit_pred = logit.predict(X_test)
print("Logit:"+ str(np.mean(logit_pred != test['crim01'])))
#lda
lda = LDA(store_covariance=True)
lda.fit(X_train, y_train)
predicted_lda = lda.predict(X_test)
print("LDA:"+str(np.mean(predicted_lda != test['crim01'])))
#NB
NB = GaussianNB()
NB.fit(X_train, y_train)
NB_pred=NB.predict(X_test)
print("NB:"+str(np.mean(NB_pred != test['crim01'])))
#KNN
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print("KNN:"+str(np.mean(knn_pred != test['crim01'])))
```

```
Logit:0.1857707509881423
LDA:0.16205533596837945
NB:0.22134387351778656
KNN:0.05533596837944664
```

Findings:
For all predictors, I noticed that the best value of k for KNN is k=6.
KNN also gave the lowest mean error out of the 4 methods for this case.

But since some of the predictors appeared to be correlated with each other, I removed some of them (and created BosSub), which made the logistic and NB test errors to increase, while LDA and KNN test errors decreased (KNN is still the lowest).