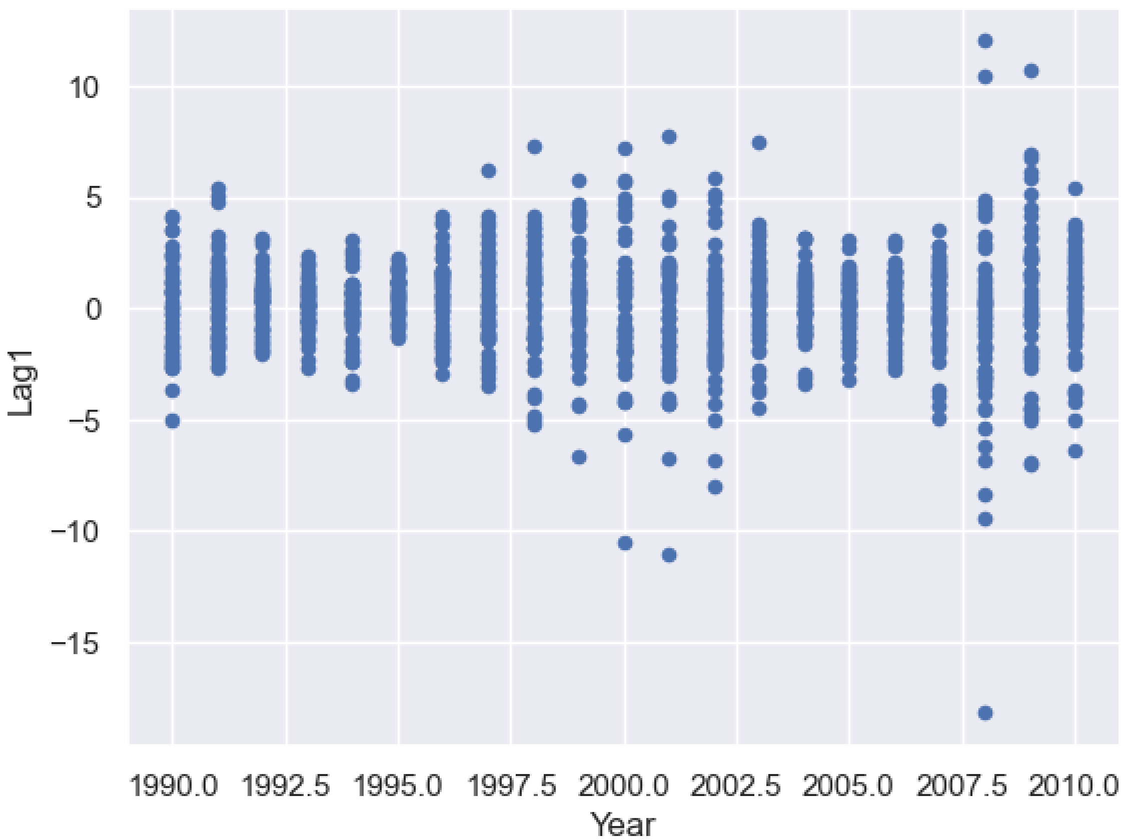
Assignment 3 Applied Questions

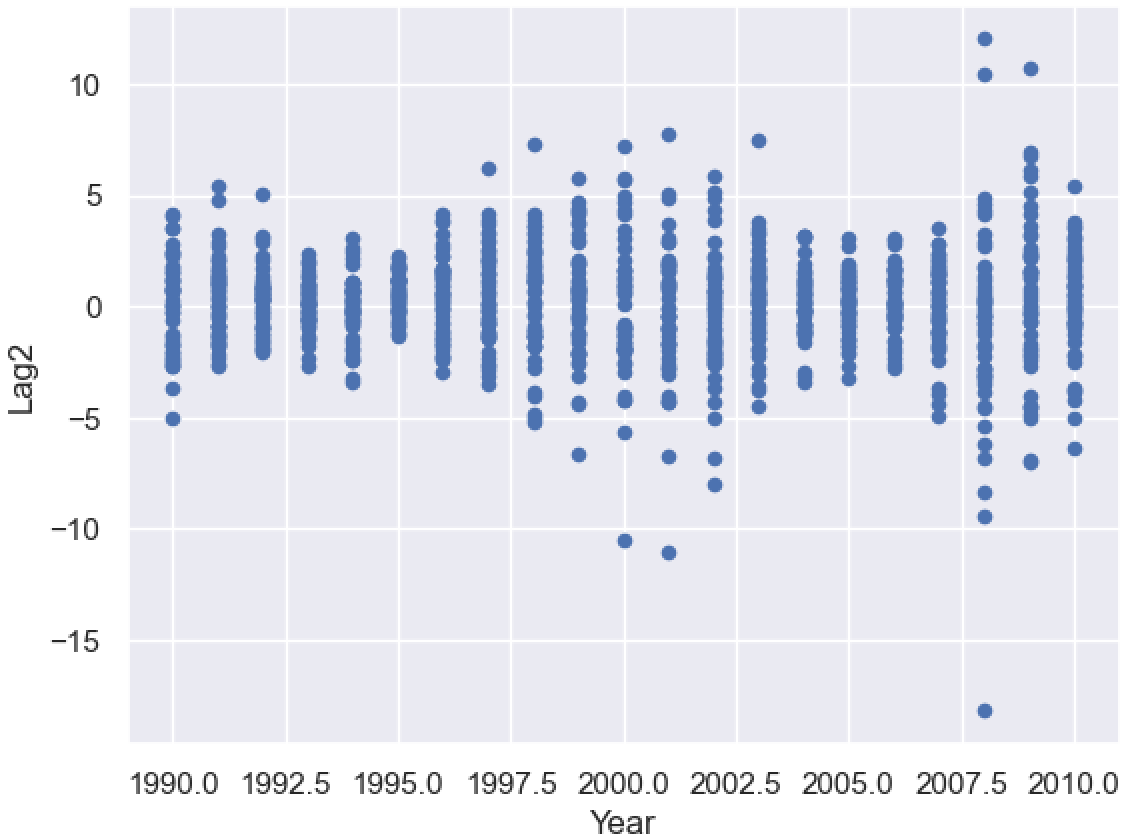
In [ ]:
```
#imports and setup
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import sklearn as sk
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier

sns.set()
```

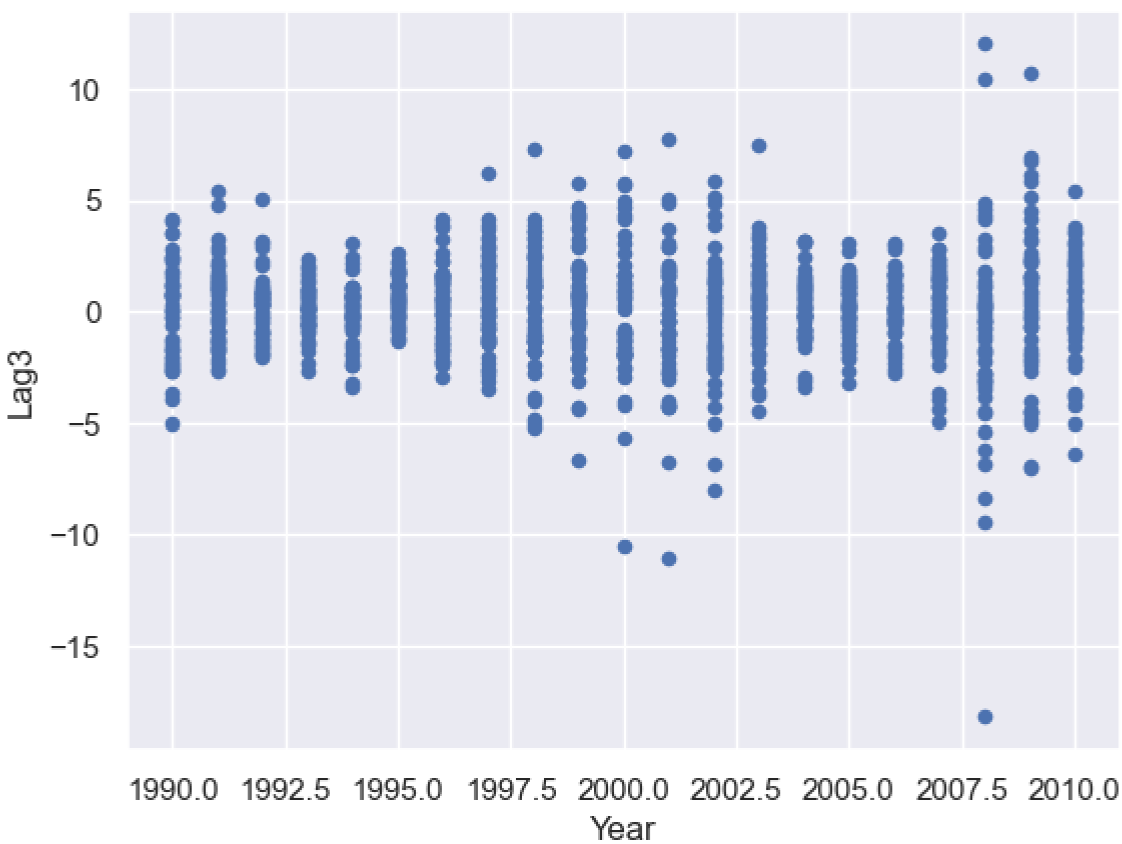
1.

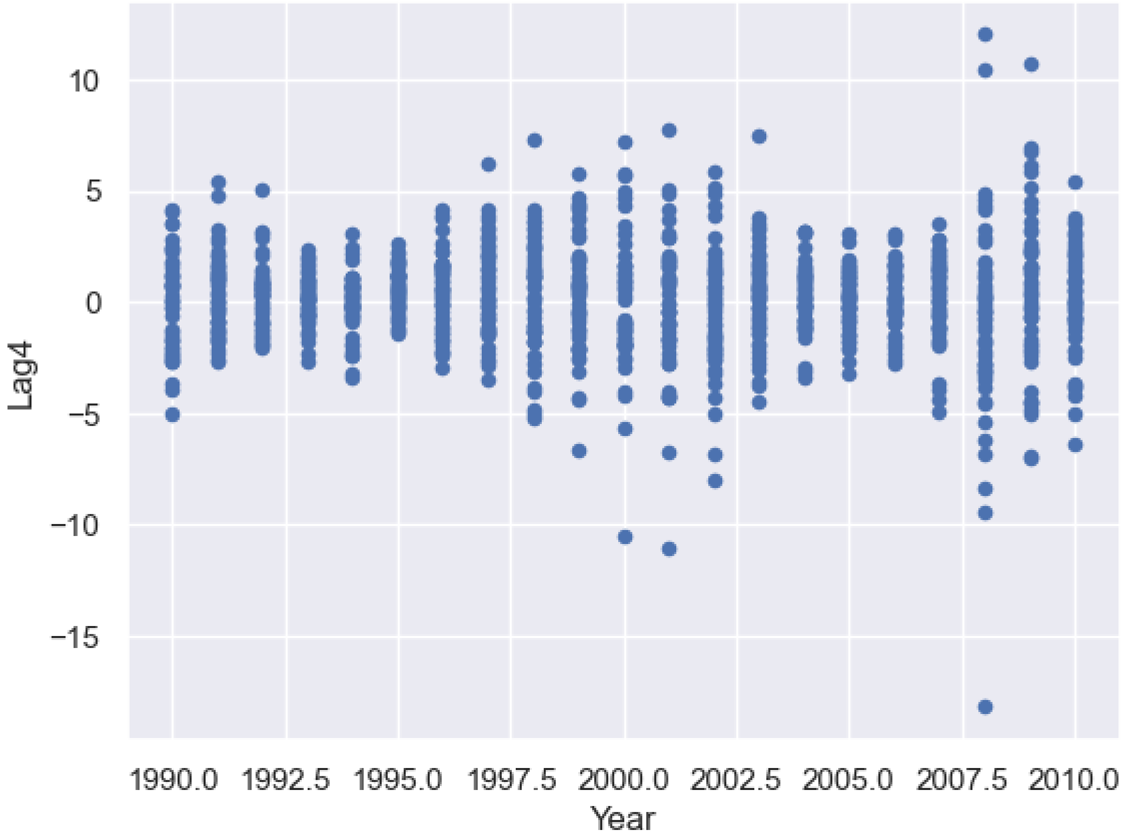
1a.

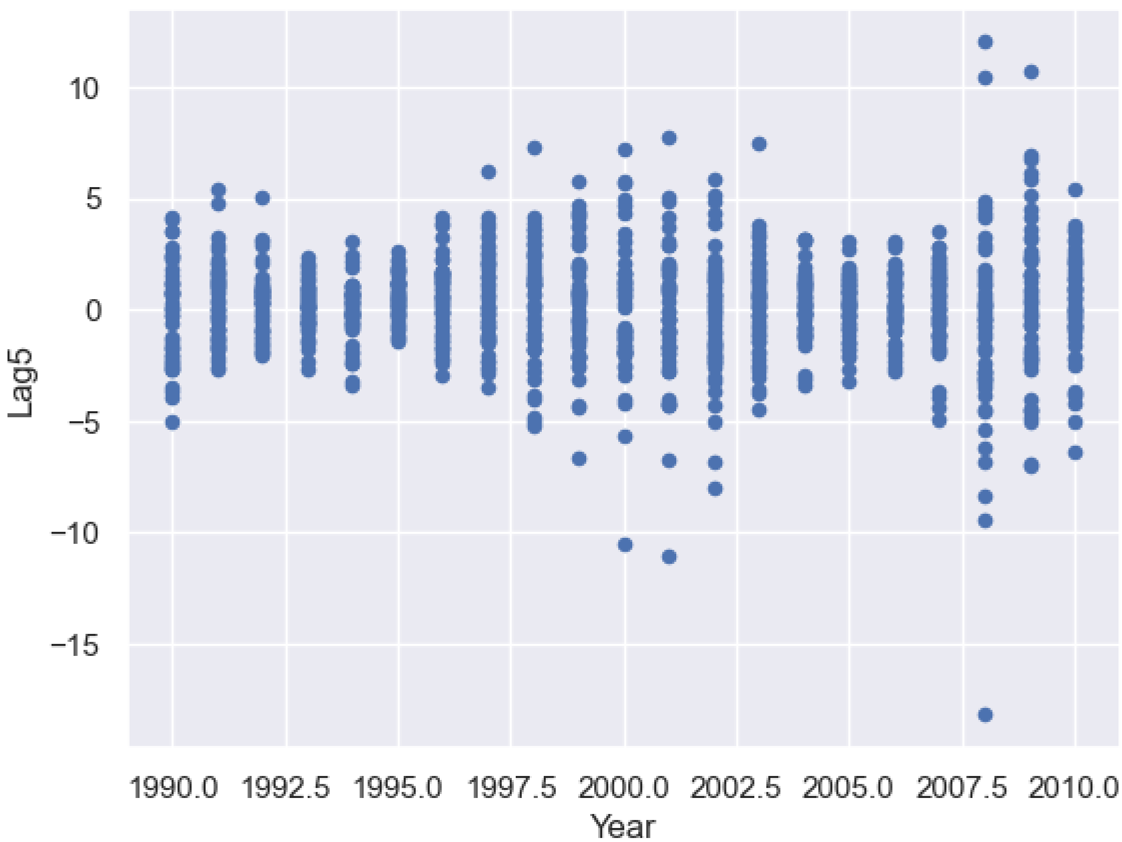
In [ ]:
```
df = pd.read_csv("Weekly.csv")

for col in df.drop(columns = ["Year"]):
    df.plot.scatter(x="Year", y=col)
    plt.show()
df.describe()
```

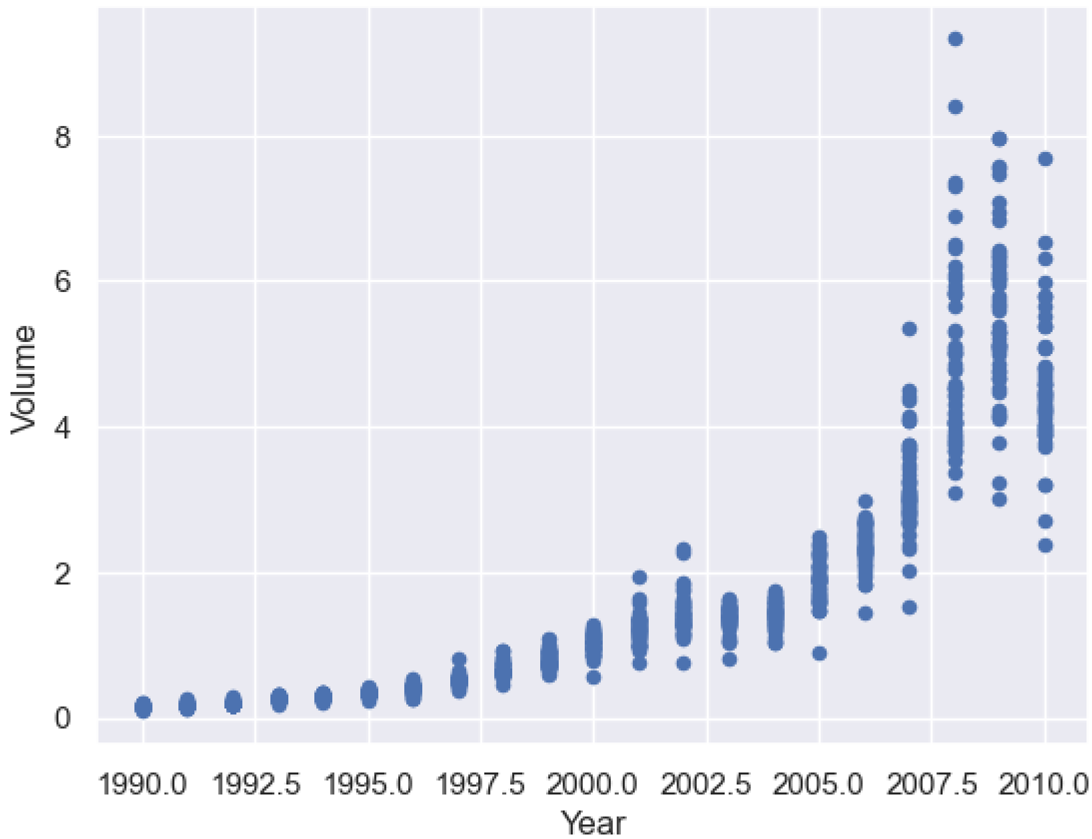



10
5
0
−5
−10
−15
Lag2
1990.0
1992.5
1995.0
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
Year

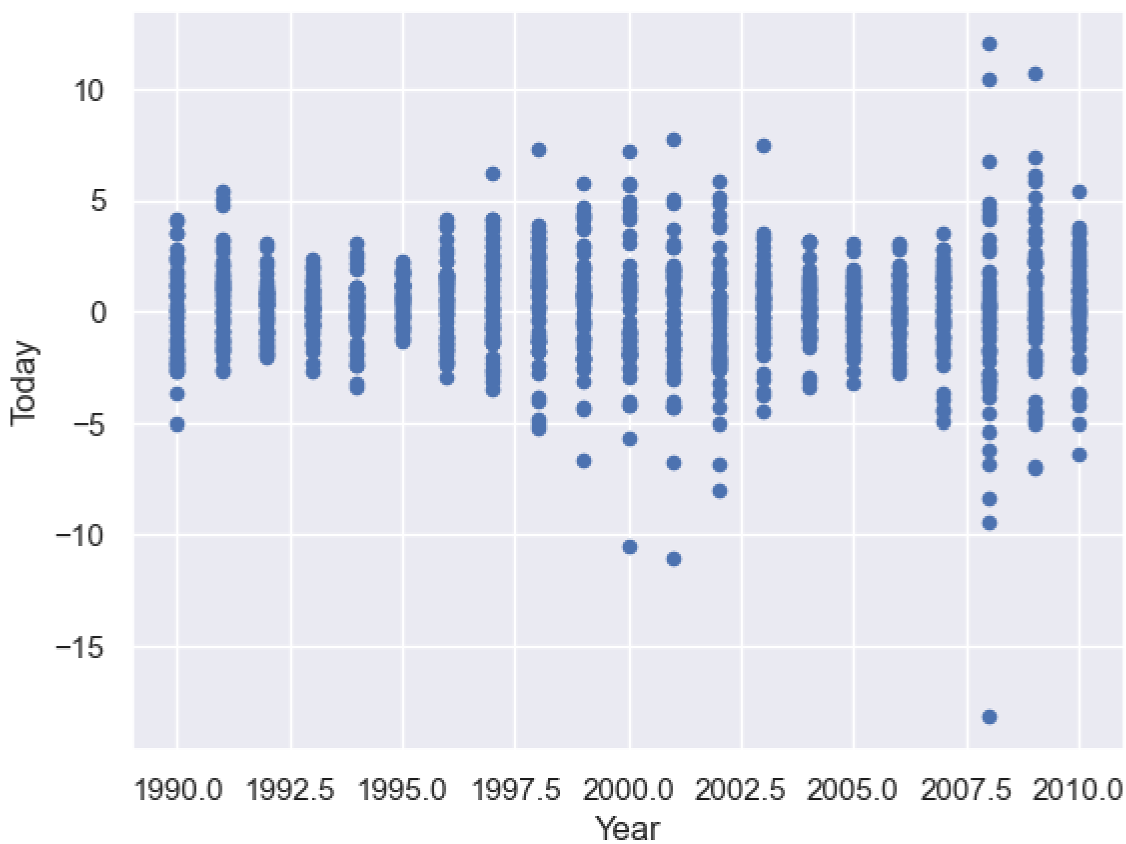

10
5
0
−5
−10
−15
Lag3
1990.0
1992.5
1995.0
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
Year


10
5
0
−5
−10
−15
Lag4
1990.0 1992.5 1995.0 1997.5 2000.0 2002.5 2005.0 2007.5 2010.0
Year

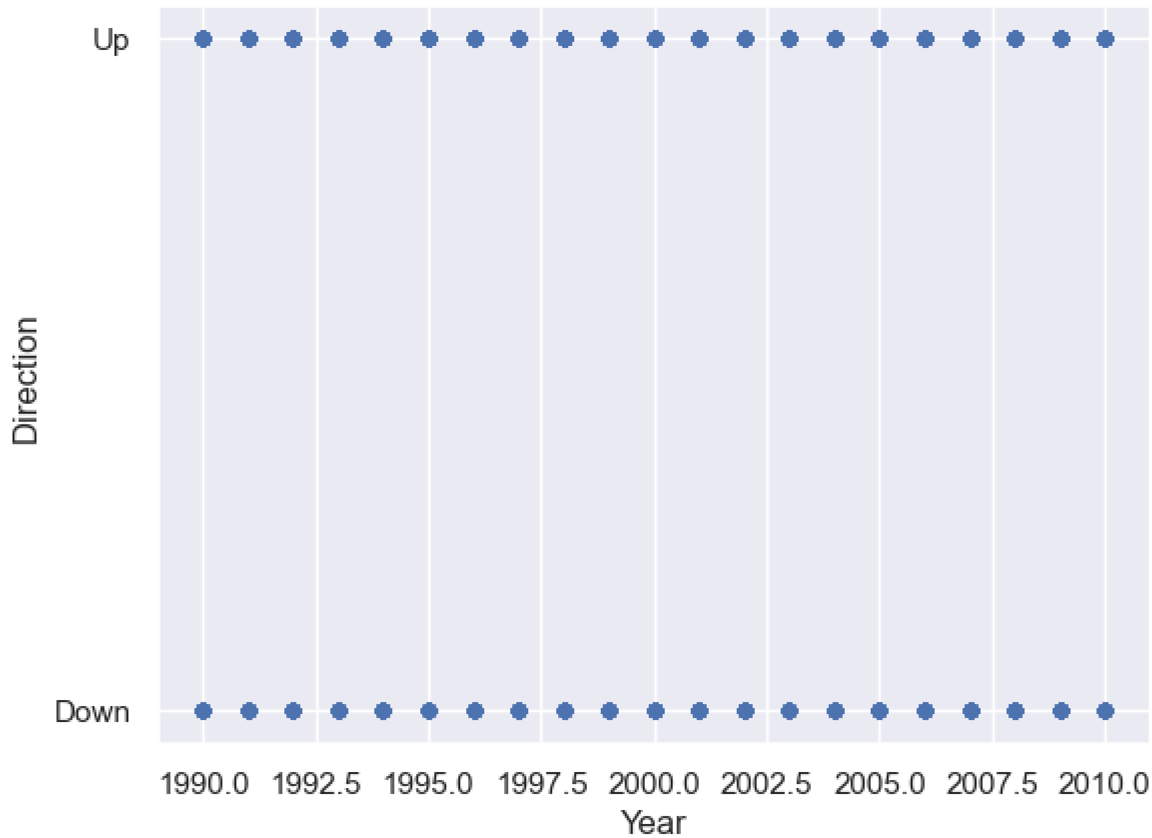

10
5
0
−5
−10
−15
Lag5
1990.0 1992.5 1995.0 1997.5 2000.0 2002.5 2005.0 2007.5 2010.0
Year



Volume
0
2
4
6
8
1990.0
1992.5
1995.0
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
Year



Today
10
5
0
−5
−10
−15
1990.0
1992.5
1995.0
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
Year



Out[ ]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	
count	1089.000000	1089.000000	1089.000000	1089.000000	1089.000000	1089.000000	1089.000000	1
mean	2000.048669	0.150585	0.151079	0.147205	0.145818	0.139893	1.574618	
std	6.033182	2.357013	2.357254	2.360502	2.360279	2.361285	1.686636	
min	1990.000000	-18.195000	-18.195000	-18.195000	-18.195000	-18.195000	0.087465	
25%	1995.000000	-1.154000	-1.154000	-1.158000	-1.158000	-1.166000	0.332022	
50%	2000.000000	0.241000	0.241000	0.241000	0.238000	0.234000	1.002680	
75%	2005.000000	1.405000	1.409000	1.409000	1.409000	1.405000	2.053727	
max	2010.000000	12.026000	12.026000	12.026000	12.026000	12.026000	9.328214	

In [ ]:

```
df.corr()
```

```
C:\Users\Bernhard\AppData\Local\Temp\ipykernel_26484\1134722465.py:1: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of numeri
c_only to silence this warning.
  df.corr()
```

Out[ ]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today
Year	1.000000	-0.032289	-0.033390	-0.030006	-0.031128	-0.030519	0.841942	-0.032460
Lag1	-0.032289	1.000000	-0.074853	0.058636	-0.071274	-0.008183	-0.064951	-0.075032
Lag2	-0.033390	-0.074853	1.000000	-0.075721	0.058382	-0.072499	-0.085513	0.059167
Lag3	-0.030006	0.058636	-0.075721	1.000000	-0.075396	0.060657	-0.069288	-0.071244
Lag4	-0.031128	-0.071274	0.058382	-0.075396	1.000000	-0.075675	-0.061075	-0.007826
Lag5	-0.030519	-0.008183	-0.072499	0.060657	-0.075675	1.000000	-0.058517	0.011013
Volume	0.841942	-0.064951	-0.085513	-0.069288	-0.061075	-0.058517	1.000000	-0.033078
Today	-0.032460	-0.075032	0.059167	-0.071244	-0.007826	0.011013	-0.033078	1.000000

It seems like the lags(1-5) data compared to year are all similar. They also have similar standard deviations, minimums, means, and quartiles.

All of the lags also have similar correlations to year as well.

Year and volume also seem to hav a singificant linear relation.

1b.

In [ ]:

```
X = sm.add_constant(df.drop(columns=['Direction', 'Today', 'Year']))
y = df['Direction'].eq('Up').mul(1)
log_reg = sm.Logit(y, X).fit()
# ll = LogisticRegression(random_state=0).fit(X, y)
log_reg.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.682441
         Iterations 4
```

Out[ ]:

Logit Regression Results

Dep. Variable:	Direction	**No. Observations:**	1089
Model:	Logit	**Df Residuals:**	1082
Method:	MLE	**Df Model:**	6
Date:	Tue, 22 Nov 2022	**Pseudo R-squ.:**	0.006580
Time:	17:55:22	**Log-Likelihood:**	-743.18
converged:	True	**LL-Null:**	-748.10
Covariance Type:	nonrobust	**LLR p-value:**	0.1313

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.2669	0.086	3.106	0.002	0.098	0.435
Lag1	-0.0413	0.026	-1.563	0.118	-0.093	0.010
Lag2	0.0584	0.027	2.175	0.030	0.006	0.111
Lag3	-0.0161	0.027	-0.602	0.547	-0.068	0.036
Lag4	-0.0278	0.026	-1.050	0.294	-0.080	0.024
Lag5	-0.0145	0.026	-0.549	0.583	-0.066	0.037
Volume	-0.0227	0.037	-0.616	0.538	-0.095	0.050

The predictor Lag 2 seems to be statistically significant with an alpha value of 0.05

1c.

In [ ]:

```
ypred = log_reg.predict(X)

#change to discrete
ypred[ypred<0.5]=0
ypred[ypred>=0.5]=1


tn, fp, fn, tp = sk.metrics.confusion_matrix(y, ypred).ravel()
print("True Negative: {}".format(tn))
print("False Positive: {}".format(fp))
print("False Negative: {}".format(fn))
print("True Positive: {}".format(tp))
print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
print("Correct 'Up' Prediction percentage: {}".format((tp)/(tp+fn)))
print("Correct 'Down' Prediction percentage: {}".format((tn)/(tn+fp)))
```

```
True Negative: 54
False Positive: 430
False Negative: 48
True Positive: 557
Fraction of Correct Predictions: 0.5610651974288338
Correct 'Up' Prediction percentage: 0.9206611570247933
Correct 'Down' Prediction percentage: 0.1115702479338843
```

The confusion matrix is telling me that most of the mistakes made by the logistic regression are

False Positives, or falsely identifying a direction as Up when it is actually down.

We can also say that the model is predicting 'Up' more correctly than it is predicting 'Down'. the model predicts 'Up' correctly 92.07% of the time, while the model predicts 'Down' correctly only 11.16% of the time.

1d.

In [ ]:
```
time_period = df[(df['Year'] >= 1990) & (df['Year'] <= 2008)]
y = time_period['Direction'].eq('Up').mul(1)
X = sm.add_constant(time_period['Lag2'])
log_reg = sm.Logit(y, X).fit()
print(log_reg.summary())


time_period2 = df[df['Year']>2008]
tX = sm.add_constant(time_period2['Lag2'])
ty = time_period2['Direction'].eq('Up').mul(1)

ypred = log_reg.predict(tX)

#change to discrete
ypred[ypred<0.5]=0
ypred[ypred>=0.5]=1


tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
print("Total Predictions: {}".format(tn+fp+fn+tp))
print("True Negative: {}".format(tn))
print("False Positive: {}".format(fp))
print("False Negative: {}".format(fn))
print("True Positive: {}".format(tp))
print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
print("Correct 'Up' Prediction percentage: {}".format((tp)/(tp+fn)))
print("Correct 'Down' Prediction percentage: {}".format((tn)/(tn+fp)))
```

```
Optimization terminated successfully.
         Current function value: 0.685555
         Iterations 4
                           Logit Regression Results                           
==============================================================================
Dep. Variable:              Direction   No. Observations:                  985
Model:                          Logit   Df Residuals:                      983
Method:                           MLE   Df Model:                            1
Date:                Tue, 22 Nov 2022   Pseudo R-squ.:                0.003076
Time:                        17:55:22   Log-Likelihood:                -675.27
converged:                       True   LL-Null:                       -677.35
Covariance Type:            nonrobust   LLR p-value:                   0.04123
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2033      0.064      3.162      0.002       0.077       0.329
Lag2           0.0581      0.029      2.024      0.043       0.002       0.114
==============================================================================
Total Predictions: 104
True Negative: 9
False Positive: 34
False Negative: 5
True Positive: 56
Fraction of Correct Predictions: 0.625
Correct 'Up' Prediction percentage: 0.9180327868852459
Correct 'Down' Prediction percentage: 0.20930232558139536
```

1e.

In [ ]:
```
lin_disc = LinearDiscriminantAnalysis()
X = time_period['Lag2'].values.reshape(-1,1) #remove constant row for LDA and QDA
lin_disc.fit(X, y)

tX = time_period2['Lag2'].values.reshape(-1,1)
ypred = lin_disc.predict(tX)

tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
print("Total Predictions: {}".format(tn+fp+fn+tp))
print("True Negative: {}".format(tn))
print("False Positive: {}".format(fp))
print("False Negative: {}".format(fn))
print("True Positive: {}".format(tp))
print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
print("Correct 'Up' Prediction percentage: {}".format((tp)/(tp+fn)))
print("Correct 'Down' Prediction percentage: {}".format((tn)/(tn+fp)))
```

```
Total Predictions: 104
True Negative: 9
False Positive: 34
False Negative: 5
True Positive: 56
Fraction of Correct Predictions: 0.625
Correct 'Up' Prediction percentage: 0.9180327868852459
Correct 'Down' Prediction percentage: 0.20930232558139536
```

1f.

In [ ]:
```
quad_disc = QuadraticDiscriminantAnalysis()
quad_disc.fit(X, y)
ypred = quad_disc.predict(tX)

tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
print("Total Predictions: {}".format(tn+fp+fn+tp))
print("True Negative: {}".format(tn))
print("False Positive: {}".format(fp))
print("False Negative: {}".format(fn))
print("True Positive: {}".format(tp))
print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
print("Correct 'Up' Prediction percentage: {}".format((tp)/(tp+fn)))
print("Correct 'Down' Prediction percentage: {}".format((tn)/(tn+fp)))
```

```
Total Predictions: 104
True Negative: 0
False Positive: 43
False Negative: 0
True Positive: 61
Fraction of Correct Predictions: 0.5865384615384616
Correct 'Up' Prediction percentage: 1.0
Correct 'Down' Prediction percentage: 0.0
```

1g.

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
ypred = knn.predict(tX)

tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
print("Total Predictions: {}".format(tn+fp+fn+tp))
print("True Negative: {}".format(tn))
print("False Positive: {}".format(fp))
print("False Negative: {}".format(fn))
print("True Positive: {}".format(tp))
print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
print("Correct 'Up' Prediction percentage: {}".format((tp)/(tp+fn)))
print("Correct 'Down' Prediction percentage: {}".format((tn)/(tn+fp)))
```

```
Total Predictions: 104
True Negative: 22
False Positive: 21
False Negative: 30
True Positive: 31
Fraction of Correct Predictions: 0.5096153846153846
Correct 'Up' Prediction percentage: 0.5081967213114754
Correct 'Down' Prediction percentage: 0.5116279069767442
```

1h.

The LDA and Logistic regression appear to have the best results on this data. It has the highest fraction of correct predictions (50.96).

1i.

In [ ]:
```
for i in range(1,11):
    knn = KNeighborsClassifier(n_neighbors=i)
```

```
    knn.fit(X, y)
    ypred = knn.predict(tX)

    tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
    print("K={}:  Correct Predictions: {}  Correct 'Up': {}  Correct 'Down': {} ".form
    print("True Negative: {}  False Positive: {}  False Negative: {}  True Positive: {

X = sm.add_constant(time_period.drop(columns=['Direction', 'Today', 'Year']))
log_reg = sm.Logit(y, X).fit()

ypred = log_reg.predict(X)
ypred[ypred<0.5]=0
ypred[ypred>=0.5]=1
tn, fp, fn, tp = sk.metrics.confusion_matrix(y, ypred).ravel()
print("Lin_reg P={}:  Correct Predictions: {}  Correct 'Up': {}  Correct 'Down': {} ".
print("True Negative: {}  False Positive: {}  False Negative: {}  True Positive: {}\n"
```

```
K=1:  Correct Predictions: 0.5096153846153846  Correct 'Up': 0.5081967213114754  Corr
ect 'Down': 0.5116279069767442
True Negative: 22  False Positive: 21  False Negative: 30  True Positive: 31

K=2:  Correct Predictions: 0.47115384615384615  Correct 'Up': 0.29508196721311475  Co
rrect 'Down': 0.7209302325581395
True Negative: 31  False Positive: 12  False Negative: 43  True Positive: 18

K=3:  Correct Predictions: 0.5480769230769231  Correct 'Up': 0.6721311475409836  Corr
ect 'Down': 0.37209302325581395
True Negative: 16  False Positive: 27  False Negative: 20  True Positive: 41

K=4:  Correct Predictions: 0.5769230769230769  Correct 'Up': 0.5573770491803278  Corr
ect 'Down': 0.6046511627906976
True Negative: 26  False Positive: 17  False Negative: 27  True Positive: 34

K=5:  Correct Predictions: 0.5384615384615384  Correct 'Up': 0.6557377049180327  Corr
ect 'Down': 0.37209302325581395
True Negative: 16  False Positive: 27  False Negative: 21  True Positive: 40

K=6:  Correct Predictions: 0.5096153846153846  Correct 'Up': 0.5409836065573771  Corr
ect 'Down': 0.46511627906976744
True Negative: 20  False Positive: 23  False Negative: 28  True Positive: 33

K=7:  Correct Predictions: 0.5480769230769231  Correct 'Up': 0.6885245901639344  Corr
ect 'Down': 0.3488372093023256
True Negative: 15  False Positive: 28  False Negative: 19  True Positive: 42

K=8:  Correct Predictions: 0.5576923076923077  Correct 'Up': 0.6065573770491803  Corr
ect 'Down': 0.4883720930232558
True Negative: 21  False Positive: 22  False Negative: 24  True Positive: 37

K=9:  Correct Predictions: 0.5480769230769231  Correct 'Up': 0.6557377049180327  Corr
ect 'Down': 0.3953488372093023
True Negative: 17  False Positive: 26  False Negative: 21  True Positive: 40

K=10:  Correct Predictions: 0.5673076923076923  Correct 'Up': 0.6065573770491803  Cor
rect 'Down': 0.5116279069767442
True Negative: 22  False Positive: 21  False Negative: 24  True Positive: 37

Optimization terminated successfully.
         Current function value: 0.681388
         Iterations 4
Lin_reg P=9:  Correct Predictions: 0.5624365482233502  Correct 'Up': 0.87132352941176
47  Correct 'Down': 0.18140589569160998
True Negative: 80  False Positive: 361  False Negative: 70  True Positive: 474
```

The best clasifier seems to still be our original Logistic regression just using Lag 2

2.

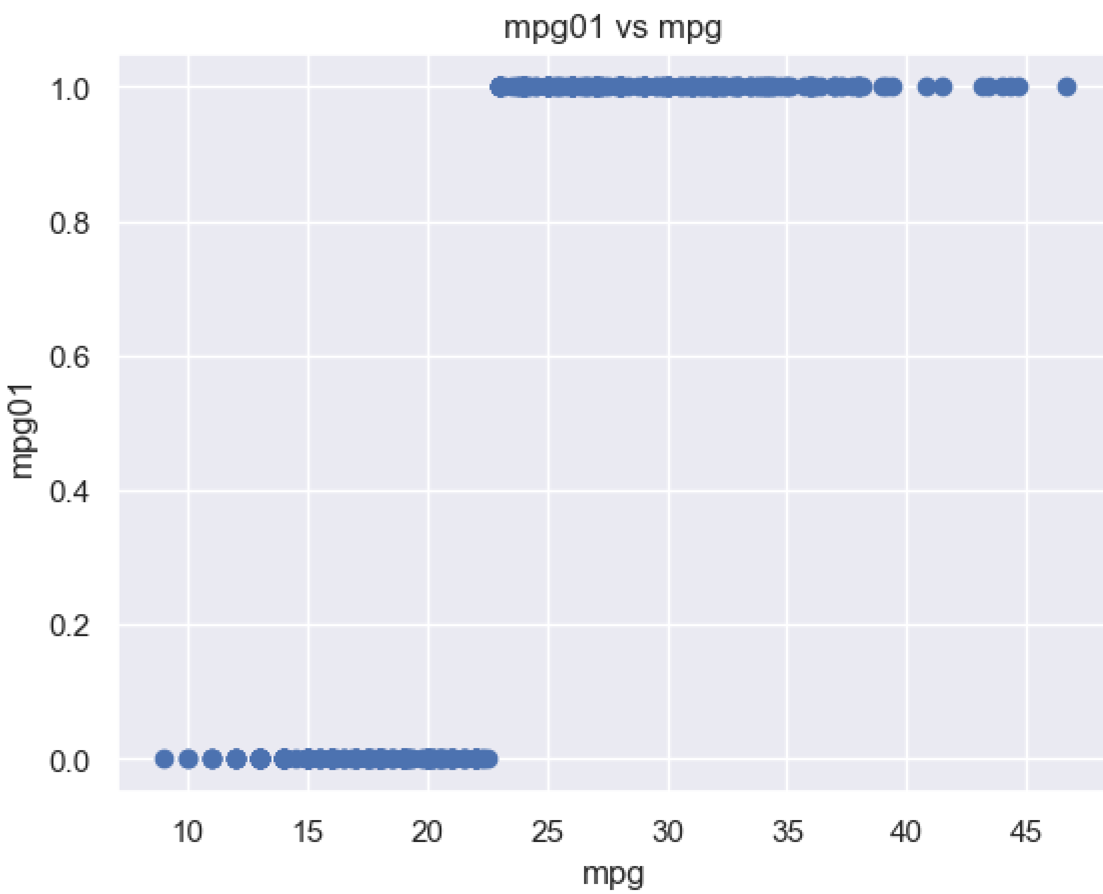
2a.

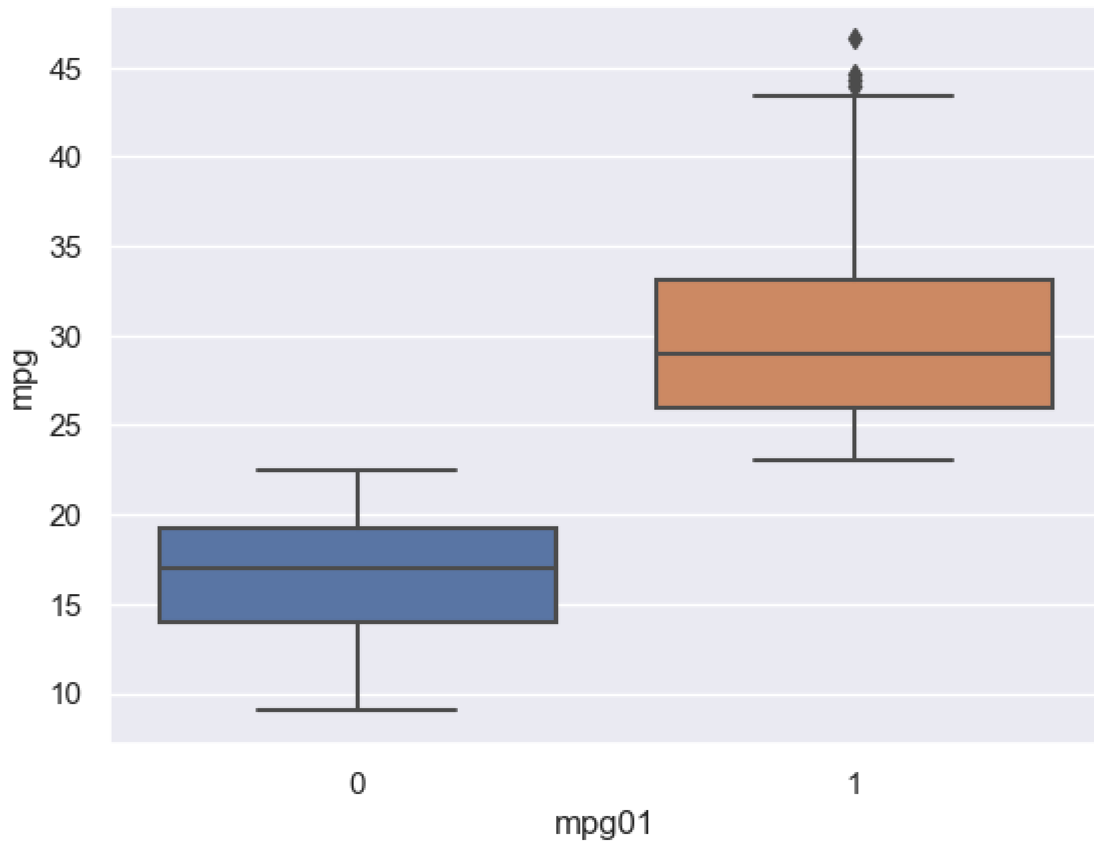
In [ ]:
```
df = pd.read_csv("Auto.csv")
```

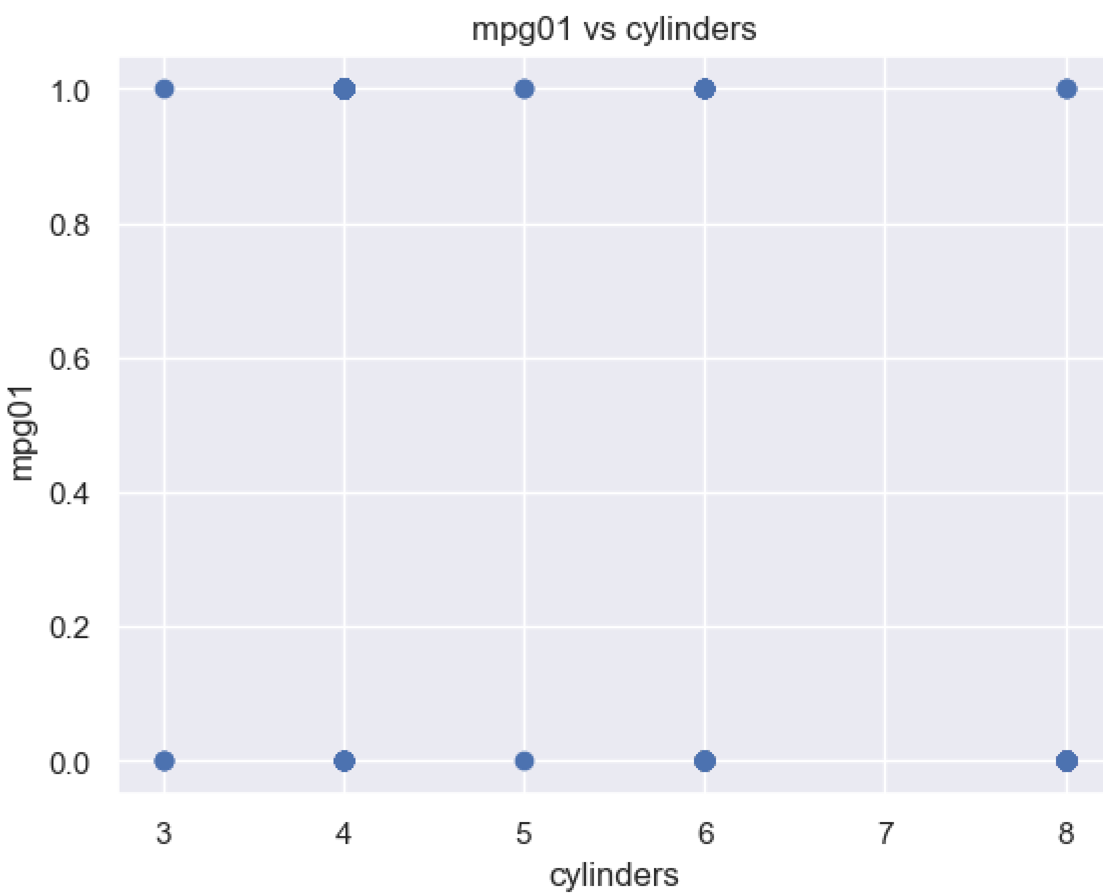
```
df[df.columns[:-2]] = df[df.columns[:-2]].apply(pd.to_numeric, errors='coerce')
df = df.dropna()
df = df.reset_index(drop=True)
mpg01 = df['mpg'].apply(lambda x: 0 if x < df['mpg'].median() else 1)

df2 = df.copy()
df2['mpg01'] = mpg01
```

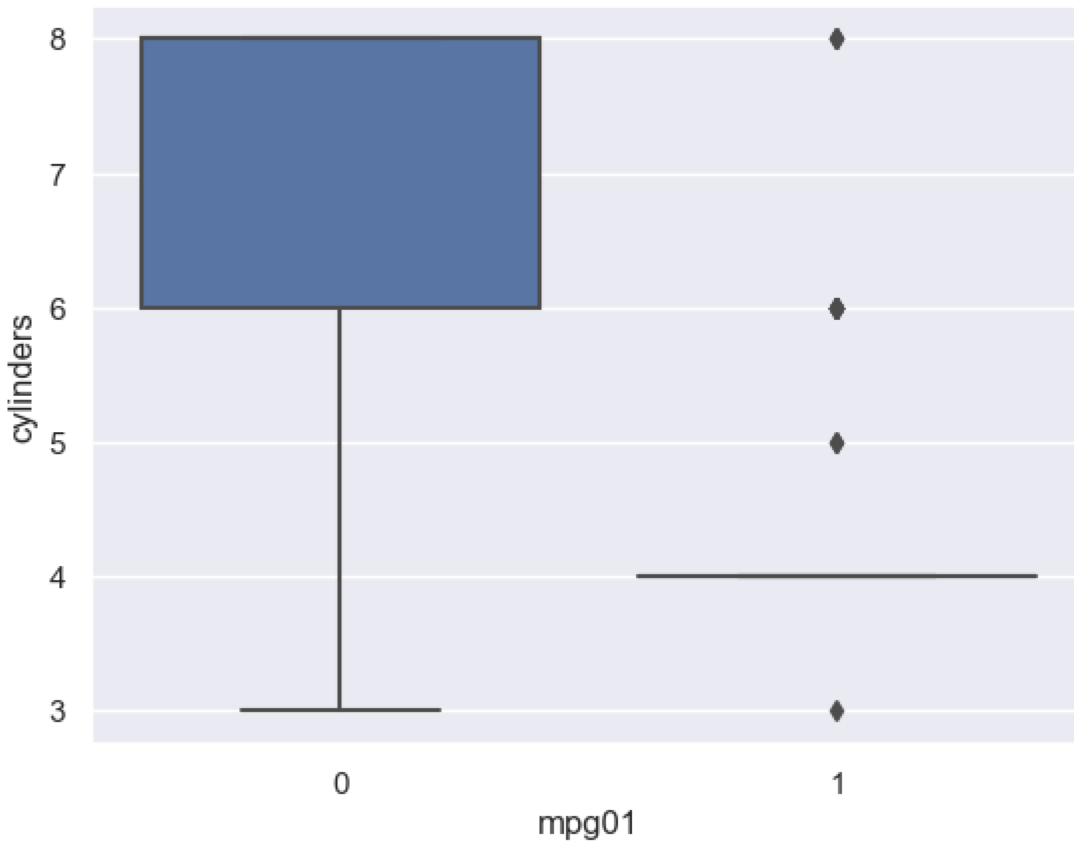
2b.

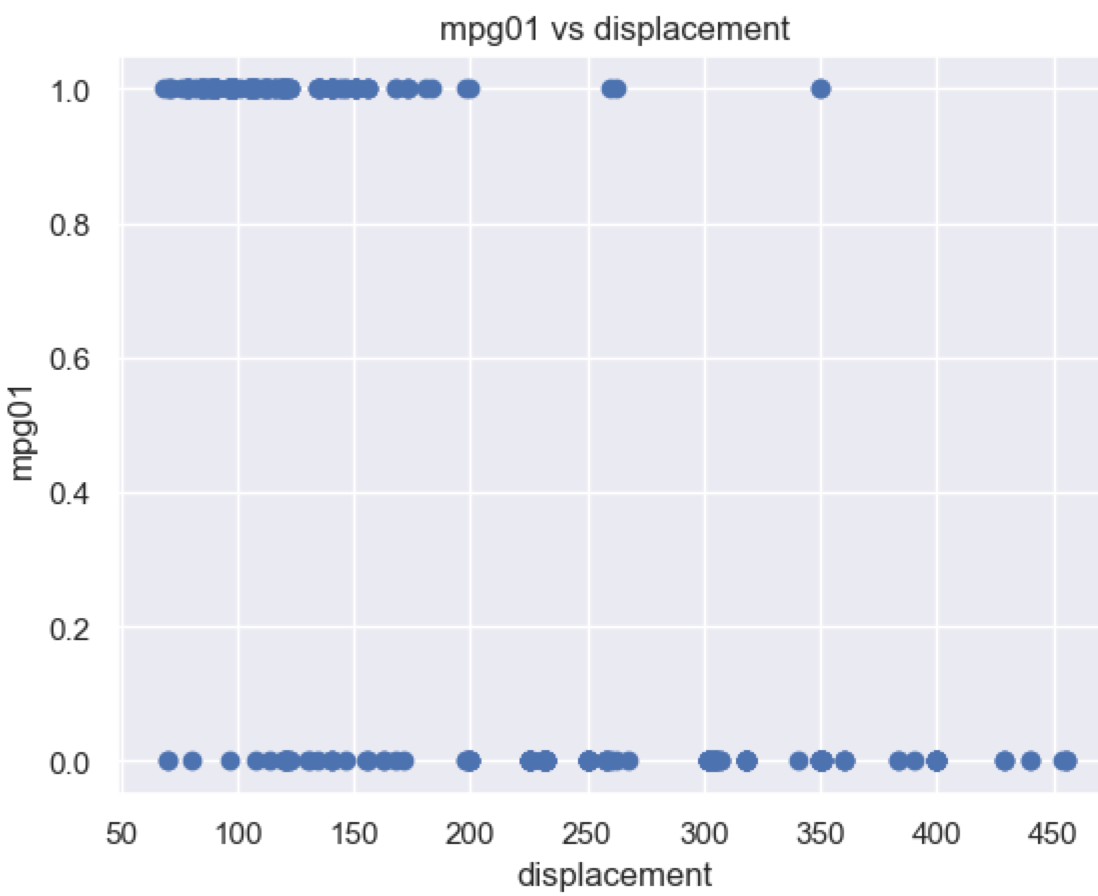
In [ ]:
```
for col in df:
    x = df[col]
    plt.scatter(x, mpg01)
    plt.xlabel(col)
    plt.ylabel('mpg01')
    plt.title('mpg01 vs {}'.format(col))
    plt.show()
    sns.boxplot(x=mpg01, y=df[col])
    plt.xlabel('mpg01')
    plt.show()
```

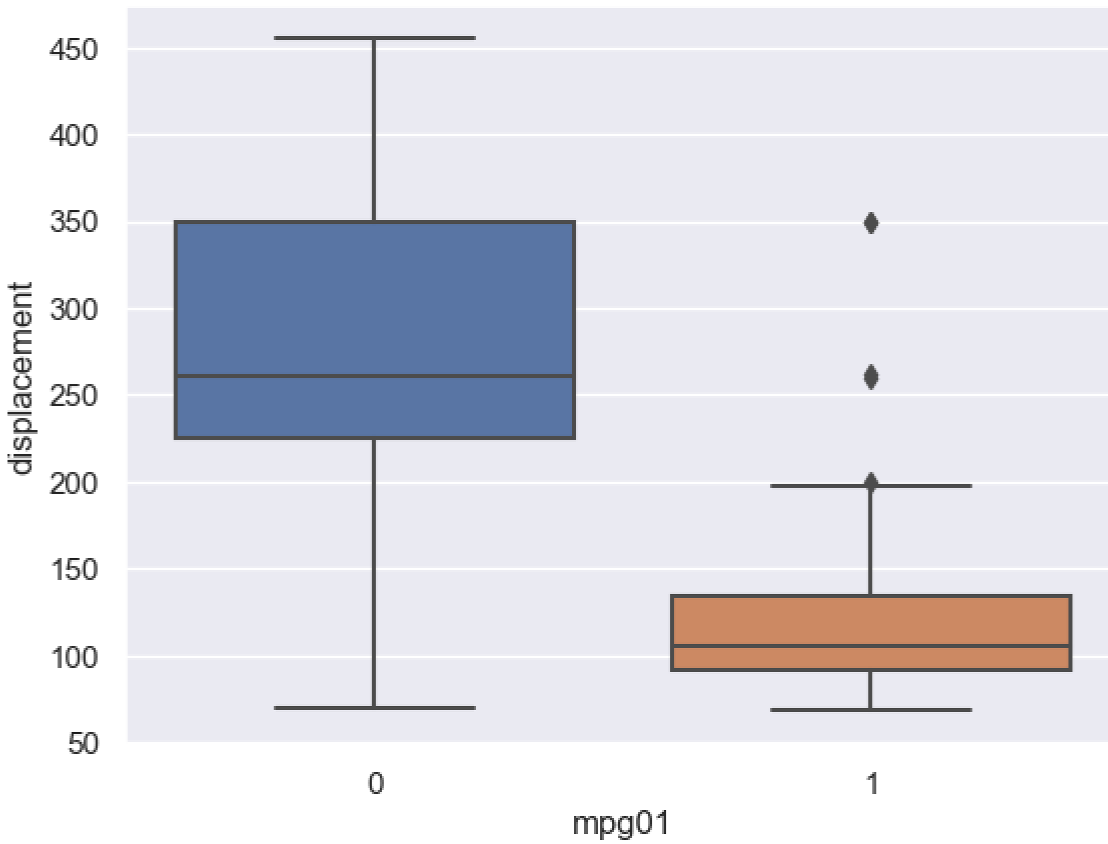


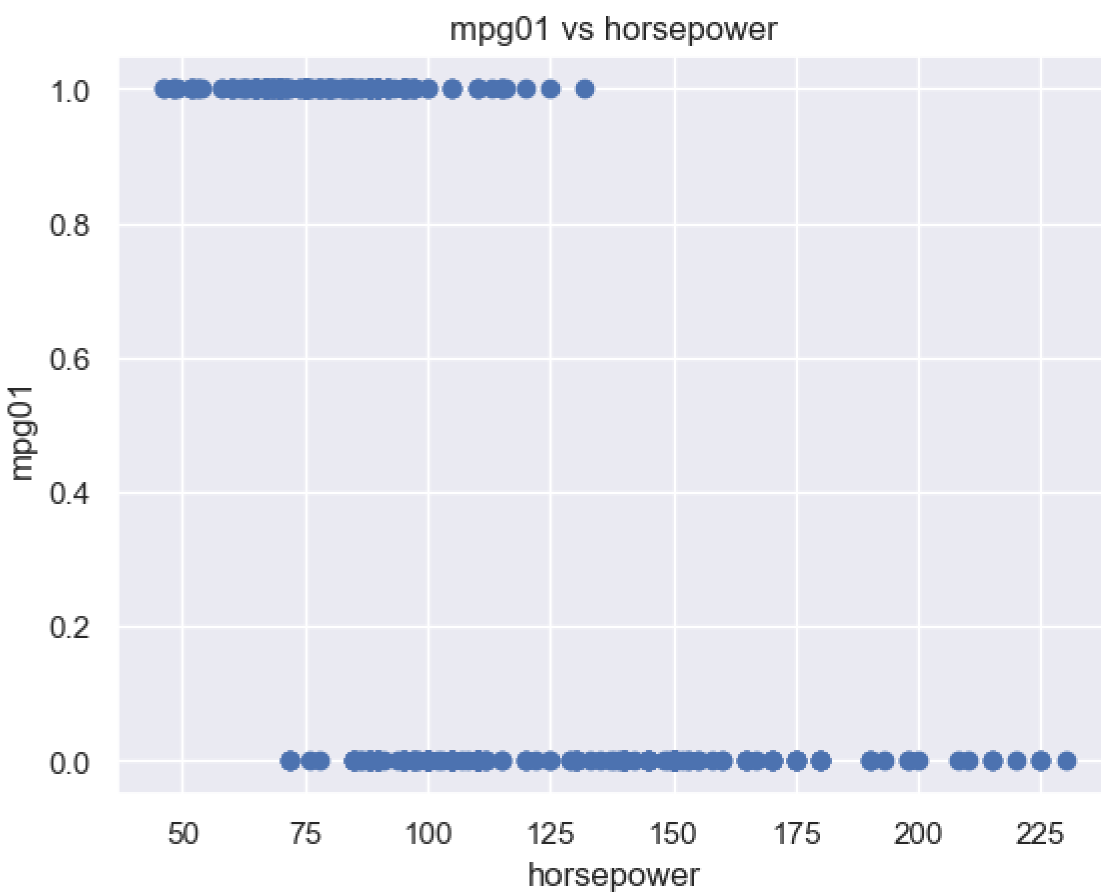

45
40
35
30
25
20
15
10
mpg
0
1
mpg01

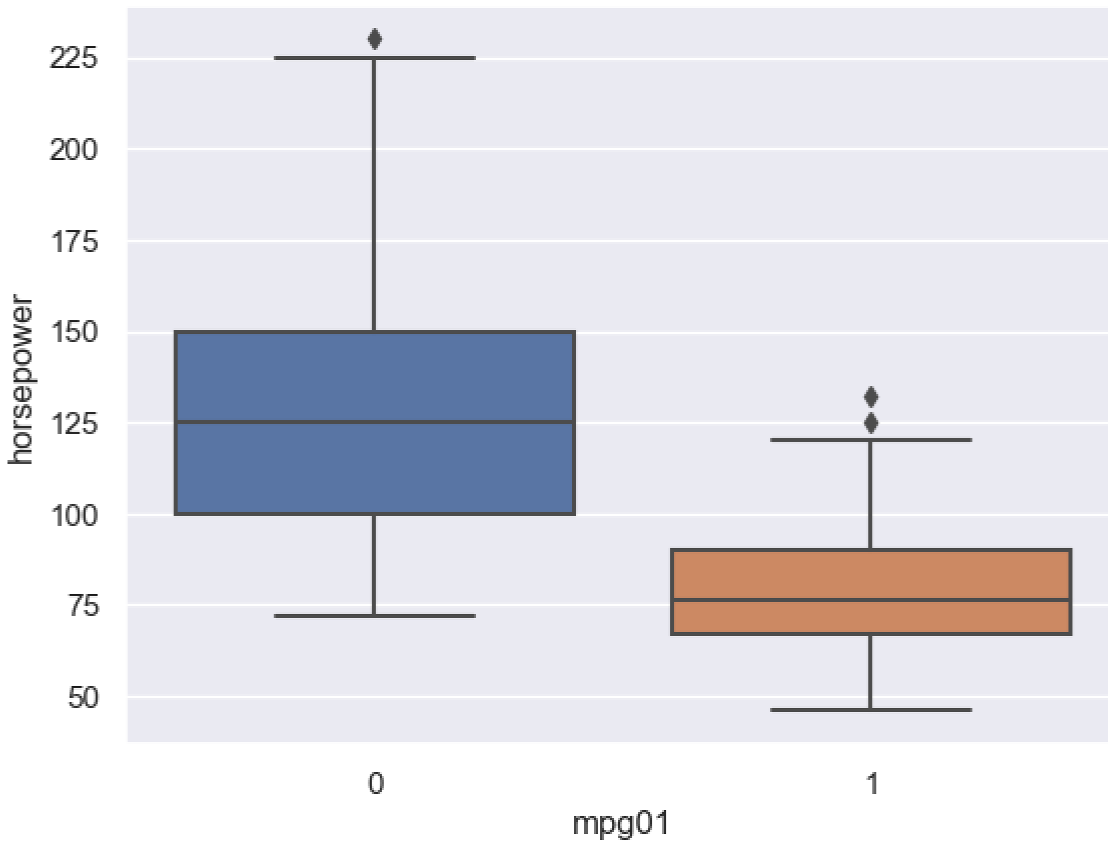

mpg01 vs cylinders
1.0
0.8
0.6
0.4
0.2
0.0
mpg01
3
4
5
6
7
8
cylinders

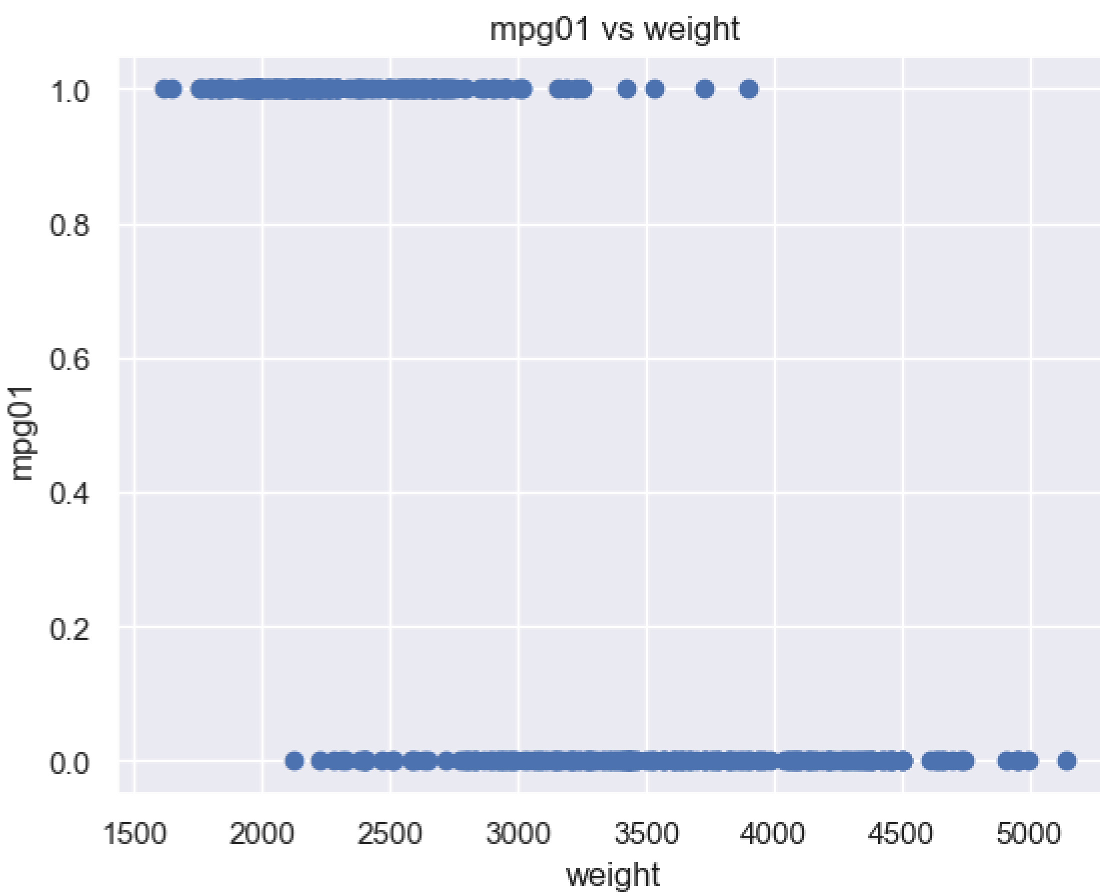

8
7
6
5
4
3
cylinders
0
1
mpg01

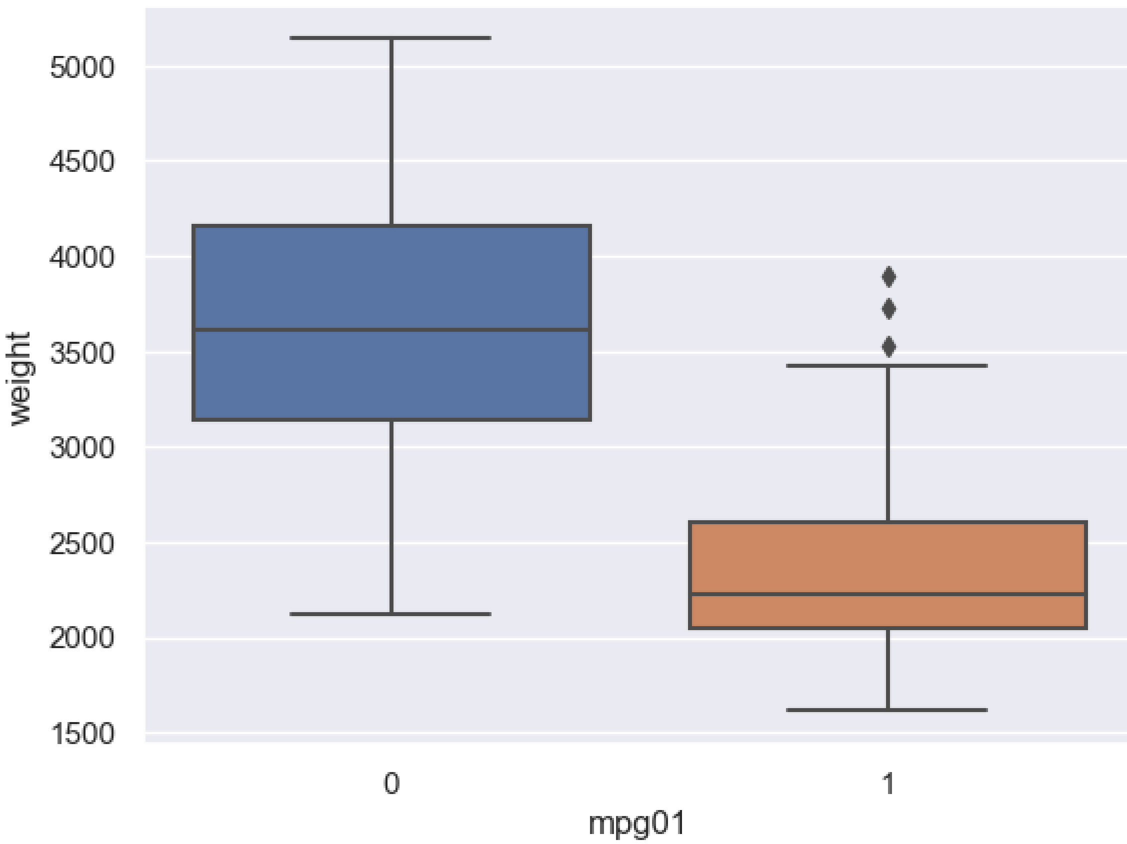

mpg01 vs displacement
1.0
0.8
0.6
0.4
0.2
0.0
mpg01
50
100
150
200
250
300
350
400
450
displacement

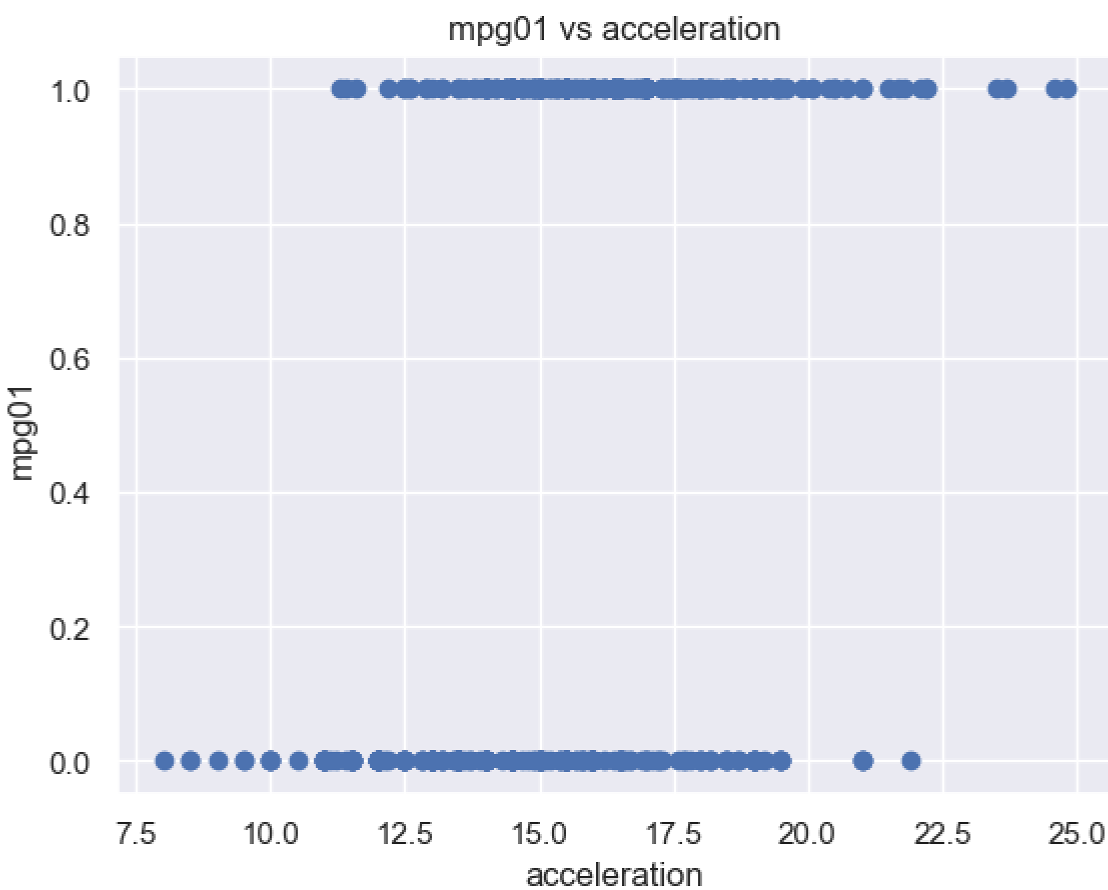

450
400
350
300
250
200
150
100
50
displacement
0
1
mpg01

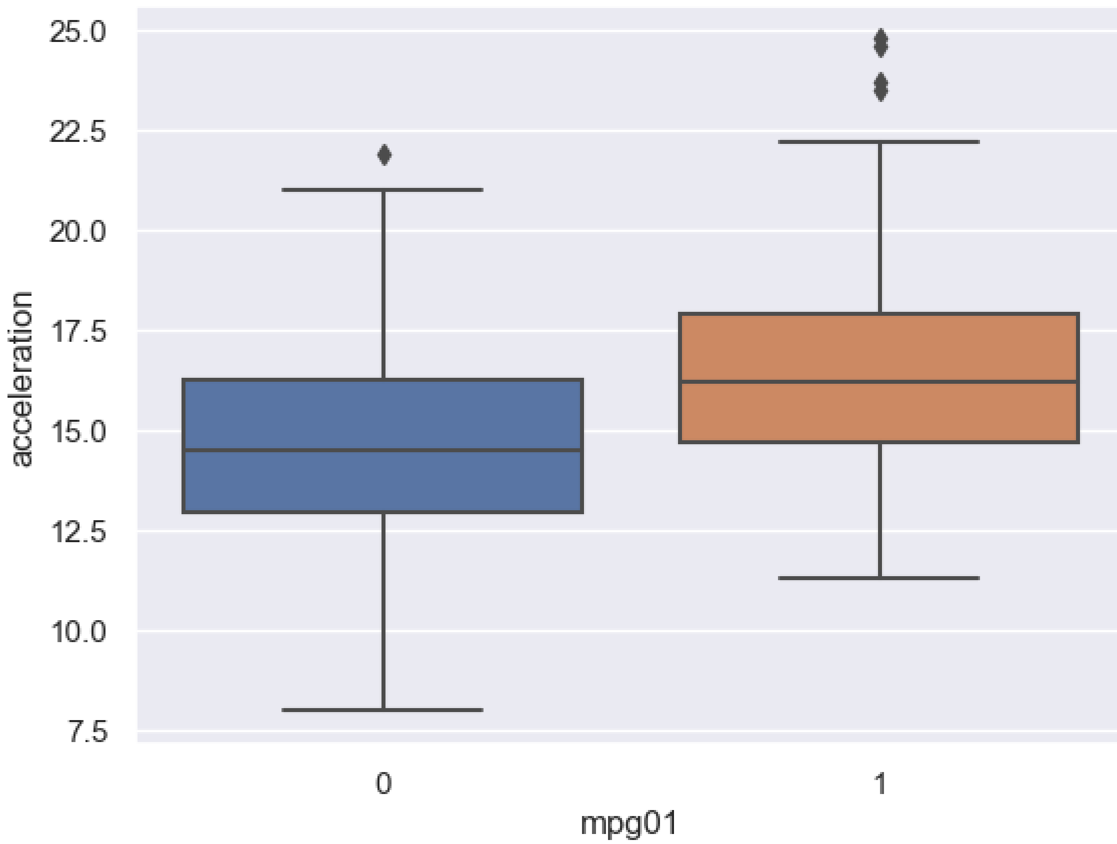

mpg01 vs horsepower
1.0
0.8
0.6
0.4
0.2
0.0
mpg01
50
75
100
125
150
175
200
225
horsepower

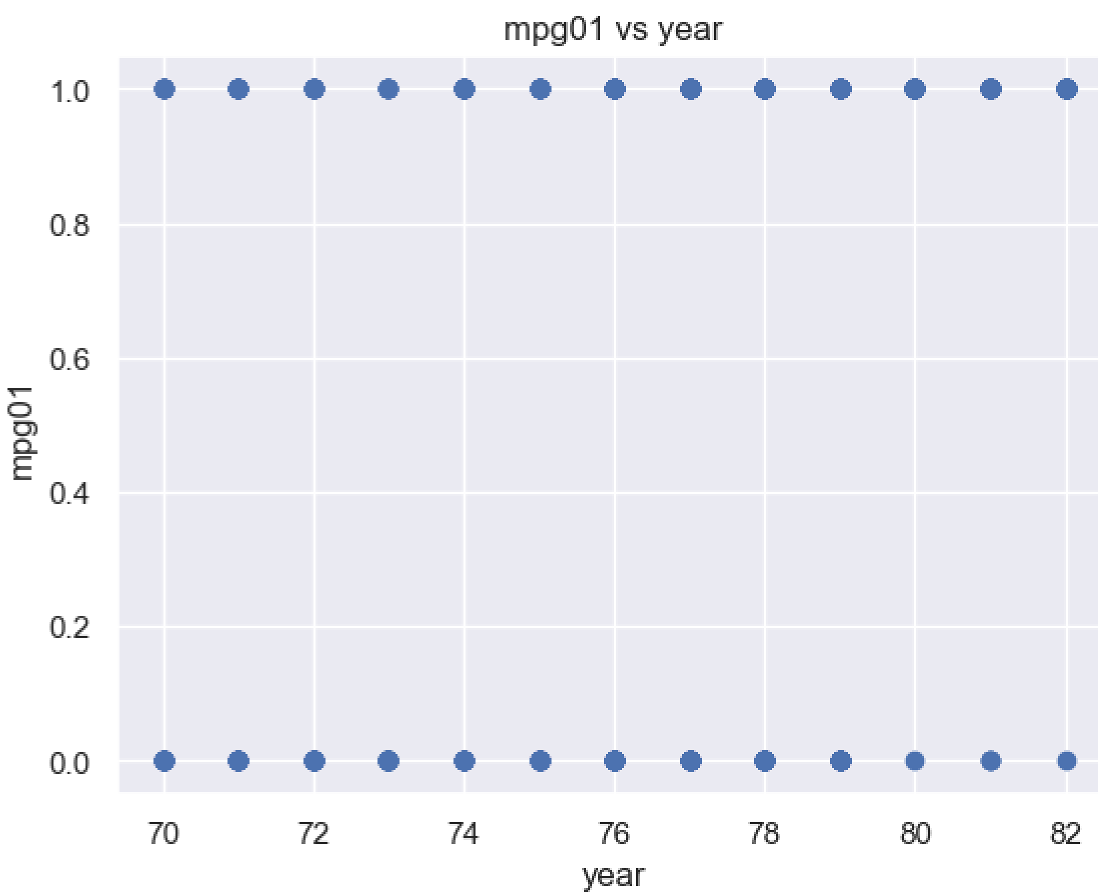

225
200
175
150
125
100
75
50
horsepower
0
1
mpg01

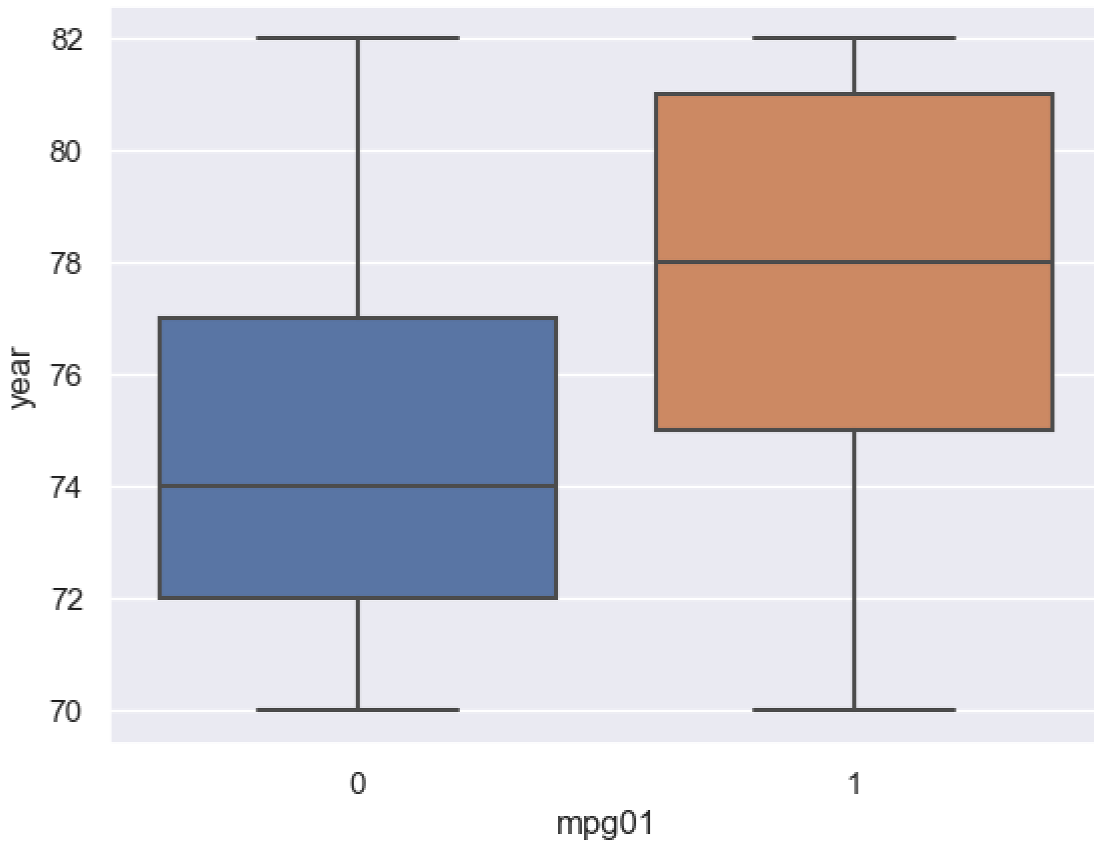

mpg01 vs weight
1.0
0.8
0.6
0.4
0.2
0.0
mpg01
1500
2000
2500
3000
3500
4000
4500
5000
weight

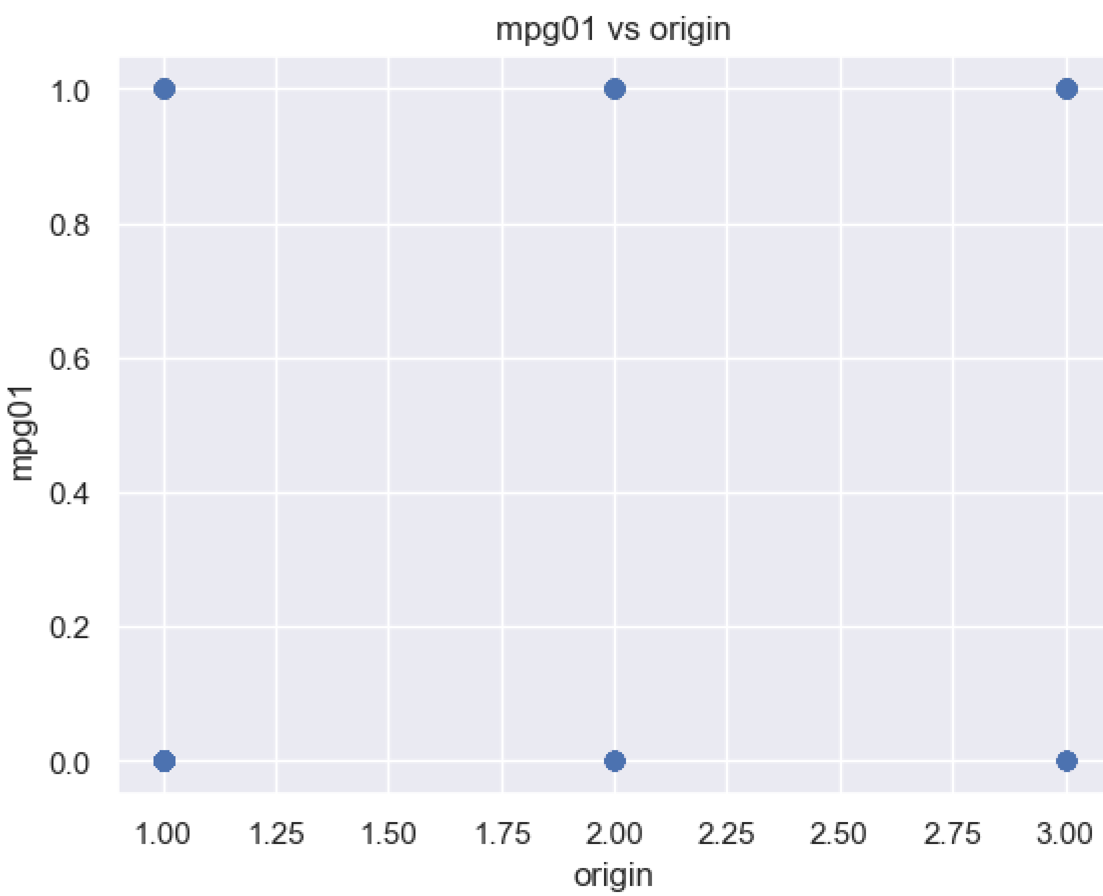

weight
5000
4500
4000
3500
3000
2500
2000
1500
0
1
mpg01

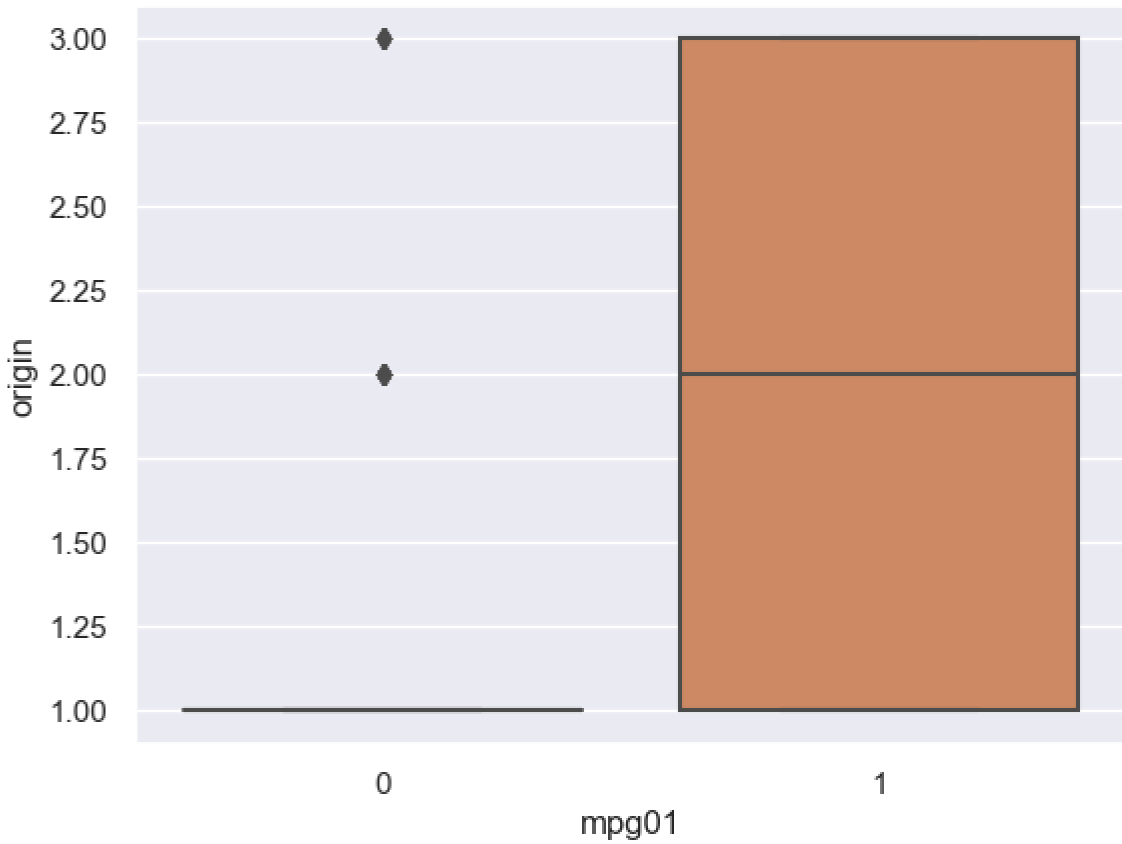

mpg01 vs acceleration
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
acceleration

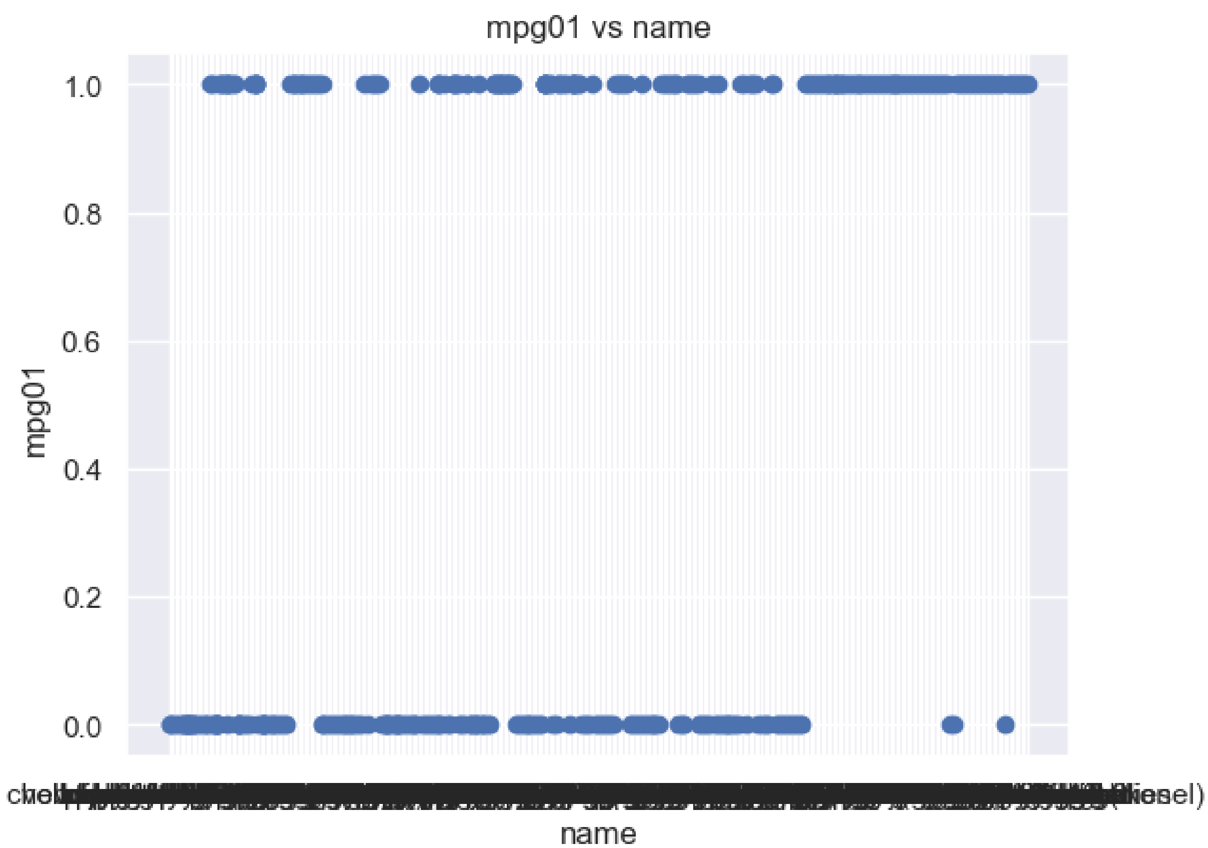

25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
acceleration
0
1
mpg01

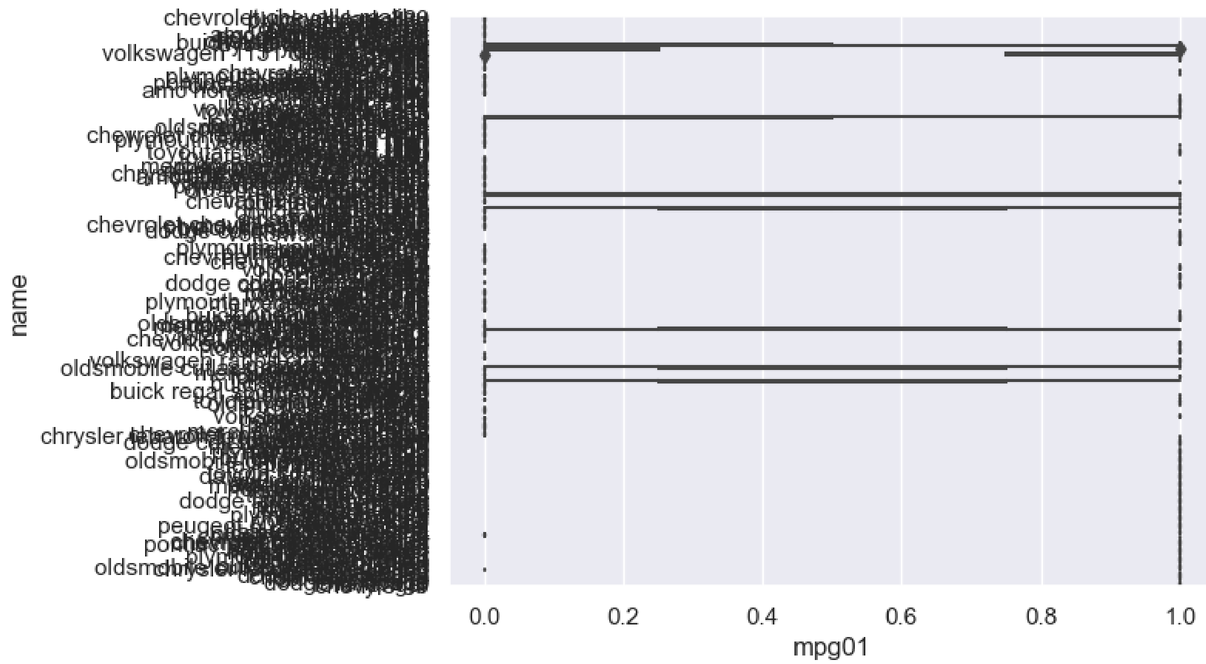

mpg01 vs year
1.0
0.8
0.6
0.4
0.2
0.0
mpg01
70
72
74
76
78
80
82
year


82
80
78
76
74
72
70
year
0
1
mpg01


mpg01 vs origin
1.0
0.8
0.6
0.4
0.2
0.0
mpg01
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
origin


origin
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
1
mpg01


mpg01 vs name
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
name



for the first plot of mpg01 vs mpg, there is a perfect separation of the data into 0 and 1 for mpg01. This makes sense, as mpg is what we created mpg01 off of.

The next plot, cylinders also sees a relatively clear distinction of mpg01. Nearly all of the sutomobiles with 4 cylinders have been classified as mpg01.

Displacement also has a clear distinction of mpg01. Nearly all of the values > 200 have mpg01 of 0, and nearly all of the values of displacement <= 200 have mpg01 of 1.

for horsepower, there is a weak trend with mpg01.

There is a distinction for weight, with most mpg01 being < around 2600, and most mpg01 being above around 2600

The predictor acceleration seems to have no clear indication of mpg01, but we can say that automobiles with a higher acceleration tend to have mpg01 equal to 1

We can also say that automobiles with higher years also tend to hav empg01 equal to 1, while lower years tend to have mpg01 equal to 0

The data also shows us that most of the samples with mpg=0 also have an origin =0.

The features most likely to be useful in predicting mpg01 seem to be mpg, cylinders, displacement and weight.

2c.

In [ ]:
```
train = df2.sample(frac=0.8, random_state=200)
test = df2.drop(train.index)

y = train['mpg01']
X = train[['mpg', 'cylinders', 'displacement', 'weight']]
```

```
tX = test[['mpg', 'cylinders', 'displacement', 'weight']]
ty = test['mpg01']
```

2d.

In [ ]:
```
lin_disc = LinearDiscriminantAnalysis()
lin_disc.fit(X, y)
ypred = lin_disc.predict(tX)
tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()

print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
```

```
Fraction of Correct Predictions: 0.9487179487179487
```

Using the variables cylinders, displacement, weight and mpg the resultant model of the LDA was able to be 94.8% accurate, or have a test error of 5.2%

2e.

In [ ]:
```
quad_disc = QuadraticDiscriminantAnalysis()
quad_disc.fit(X, y)
ypred = quad_disc.predict(tX)
tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()

print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
```

```
Fraction of Correct Predictions: 0.9615384615384616
```

Using the variables cylinders, displacement, weight, and mpg the resultant model of the QDA was able to be 96.15% accurate, or have a test error of 3.85%.

2f.

In [ ]:
```
log_reg = sm.Logit(y, X).fit()
ypred = log_reg.predict(tX)
ypred[ypred<0.5]=0
ypred[ypred>=0.5]=1
tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()

print("Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))
```

```
Optimization terminated successfully.
         Current function value: 0.110820
         Iterations 10
Fraction of Correct Predictions: 0.9487179487179487
```

Using the variables cylinders, displacement, weight, and mpg the resultant model of the Logistic Regression was able to be 94.8% accurate, or have a test error of 5.2%.

2g.

In [ ]:
```
all_pred = []
for i in range(1,11):
    knn = KNeighborsClassifier(n_neighbors=i)
```

```
    knn.fit(X, y)
    ypred = knn.predict(tX)

    tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
    correct_frac = (tn+tp)/(tn+tp+fp+fn)
    all_pred.append(correct_frac)
    print("K = {}, Fraction of Correct Predictions: {}".format(i, correct_frac))
print(max(all_pred))
```

```
K = 1, Fraction of Correct Predictions: 0.8717948717948718
K = 2, Fraction of Correct Predictions: 0.8717948717948718
K = 3, Fraction of Correct Predictions: 0.9230769230769231
K = 4, Fraction of Correct Predictions: 0.8846153846153846
K = 5, Fraction of Correct Predictions: 0.9358974358974359
K = 6, Fraction of Correct Predictions: 0.9230769230769231
K = 7, Fraction of Correct Predictions: 0.9358974358974359
K = 8, Fraction of Correct Predictions: 0.9487179487179487
K = 9, Fraction of Correct Predictions: 0.9358974358974359
K = 10, Fraction of Correct Predictions: 0.9102564102564102
0.9487179487179487
```

Using the variables cylinders, displacement, weight, and mpg the resultant model of the KNN was able to be 94.87% accurate, or have a test error of 5.13%.

The K values that seem to work the best for on this data set are K=9, 7, 6, and 5. All of these K values give the lowest test error for the data I am using.

3.

In [ ]:

```
df = pd.read_csv("Boston.csv")
crim01 = df['crim'].apply(lambda x: 0 if x < df['crim'].median() else 1)
df2 = df.copy()
df2['crim01'] = crim01

subset1 = df2[['crim', 'chas', 'zn', 'indus', 'crim01']]
subset2 = df2[['nox', 'rm', 'age', 'rad', 'lstat', 'crim01']]
subset3 = df2[['tax', 'ptratio', 'medv', 'crim01']]
subsets = [subset1, subset2, subset3]


for subset in subsets:
    print("---------------- {} -------------".format(subset.drop(columns=['crim01']).c
    train = subset.sample(frac=0.6, random_state=200)
    test = subset.drop(train.index)

    y = train['crim01']
    X = train.drop(columns=['crim01'])

    tX = test.drop(columns=['crim01'])
    ty = test['crim01']
    #logistic:
    log_reg = sm.Logit(y, X).fit()
    ypred = log_reg.predict(tX)
    ypred[ypred<0.5]=0
    ypred[ypred>=0.5]=1
```

```
    tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()

    print("Logisitic: Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fr

    #LDA:
    lin_disc = LinearDiscriminantAnalysis()
    lin_disc.fit(X, y)
    ypred = lin_disc.predict(tX)
    tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()

    print("LDA: Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))


    #QDA
    quad_disc = QuadraticDiscriminantAnalysis()
    quad_disc.fit(X, y)
    ypred = quad_disc.predict(tX)
    tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()

    print("QDA: Fraction of Correct Predictions: {}".format((tn+tp)/(tn+tp+fp+fn)))


    #KNN
    all_pred = []
    for i in range(1,11):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X, y)
        ypred = knn.predict(tX)

        tn, fp, fn, tp = sk.metrics.confusion_matrix(ty, ypred).ravel()
        correct_frac = (tn+tp)/(tn+tp+fp+fn)
        all_pred.append(correct_frac)
        print("K = {}, Fraction of Correct Predictions: {}".format(i, correct_frac))
    print(max(all_pred))
```

```
---------------- Index(['crim', 'chas', 'zn', 'indus'], dtype='object') -------------
Optimization terminated successfully.
         Current function value: 0.160218
         Iterations 14
Logisitic: Fraction of Correct Predictions: 0.9257425742574258
LDA: Fraction of Correct Predictions: 0.806930693069307
QDA: Fraction of Correct Predictions: 0.9603960396039604
K = 1, Fraction of Correct Predictions: 0.9504950495049505
K = 2, Fraction of Correct Predictions: 0.9504950495049505
K = 3, Fraction of Correct Predictions: 0.9306930693069307
K = 4, Fraction of Correct Predictions: 0.9405940594059405
K = 5, Fraction of Correct Predictions: 0.9306930693069307
K = 6, Fraction of Correct Predictions: 0.9257425742574258
K = 7, Fraction of Correct Predictions: 0.9257425742574258
K = 8, Fraction of Correct Predictions: 0.9257425742574258
K = 9, Fraction of Correct Predictions: 0.9356435643564357
K = 10, Fraction of Correct Predictions: 0.9207920792079208
0.9504950495049505
---------------- Index(['nox', 'rm', 'age', 'rad', 'lstat'], dtype='object') --------
-----
Optimization terminated successfully.
         Current function value: 0.345373
         Iterations 9
Logisitic: Fraction of Correct Predictions: 0.8663366336633663
LDA: Fraction of Correct Predictions: 0.8663366336633663
QDA: Fraction of Correct Predictions: 0.8168316831683168
K = 1, Fraction of Correct Predictions: 0.8118811881188119
K = 2, Fraction of Correct Predictions: 0.806930693069307
K = 3, Fraction of Correct Predictions: 0.8217821782178217
K = 4, Fraction of Correct Predictions: 0.8415841584158416
K = 5, Fraction of Correct Predictions: 0.8465346534653465
K = 6, Fraction of Correct Predictions: 0.8267326732673267
K = 7, Fraction of Correct Predictions: 0.8366336633663366
K = 8, Fraction of Correct Predictions: 0.8316831683168316
K = 9, Fraction of Correct Predictions: 0.8267326732673267
K = 10, Fraction of Correct Predictions: 0.8217821782178217
0.8465346534653465
---------------- Index(['tax', 'ptratio', 'medv'], dtype='object') -------------
Optimization terminated successfully.
         Current function value: 0.498376
         Iterations 6
Logisitic: Fraction of Correct Predictions: 0.7277227722772277
LDA: Fraction of Correct Predictions: 0.7871287128712872
QDA: Fraction of Correct Predictions: 0.7475247524752475
K = 1, Fraction of Correct Predictions: 0.8910891089108911
K = 2, Fraction of Correct Predictions: 0.905940594059406
K = 3, Fraction of Correct Predictions: 0.8910891089108911
K = 4, Fraction of Correct Predictions: 0.8910891089108911
K = 5, Fraction of Correct Predictions: 0.8663366336633663
K = 6, Fraction of Correct Predictions: 0.8613861386138614
K = 7, Fraction of Correct Predictions: 0.8613861386138614
K = 8, Fraction of Correct Predictions: 0.8613861386138614
K = 9, Fraction of Correct Predictions: 0.8514851485148515
K = 10, Fraction of Correct Predictions: 0.8564356435643564
0.905940594059406
```

Due to these findings, we saw that the best fit for the crim01 was the first set of predictors, which were crim, chas, zn, indus. We were able to predict crim01 with a 95% accuracy with the KNN model, and a 96% accuracy with the QDA model. I also noticed that the KNN model performed consistently well accross all sets of predictors, boasting a 90% or greater accuracy rate with all of the predictor sets that were used.