Question 4.5

A

The LDA will do better on the testing set since QDA might cause overfitting, however, since QDA is more flexible, it should do better on the training set.

B

QDA will do better in both traning and tetsing sets.

C ¶

QDA's prediction accuracy would improve relative to LDA. In QDA and other flexible models, the accuracy usually increases with the sample size, however, in LDA the model becomes more stable and accurate with smaller sample sizes.

D

FALSE, if the Bayes decision boundary is linear, there is a higher chance of overfitting the model when using QDA, therefore the error using LDA should be smaller in this scenario.

Question 4.12

A

$\hat{Pr}(Y = orange|X = x) = \frac{exp(β̂0+β̂1x)}{1+exp(β̂0+β̂1x)}$

log($\hat{Pr}\frac{(Y=orange|X=x)}{1-Pr(Y=orange|X=x))}$ = β̂0 + β̂1x

B

$$\hat{Pr}(Y = orange|X = x) = \frac{exp(\hat{\alpha}_{orange0} + \hat{\alpha}_{orange1x})}{exp(\hat{\alpha}_{orange0} + \hat{\alpha}_{orange1x}) + exp(\hat{\alpha}_{apple0} + \hat{\alpha}_{apple1x})}$$

$$log(\hat{Pr}\frac{(Y = orange|X = x)}{1 - Pr(Y = orange|X = x)}) = (\hat{\alpha}_{orange0} + \hat{\alpha}_{orange1x}) - (\hat{\alpha}_{apple0} + \hat{\alpha}_{apple1x})$$

C

$$\hat{\beta}_0 = \hat{\alpha}_{orange0} - \hat{\alpha}_{apple0}$$
$$\hat{\beta}_1 x = \hat{\alpha}_{orange1} x - \hat{\alpha}_{apple1} x$$

D

$$\hat{\beta}_0 = \hat{\alpha}_{orange0} - \hat{\alpha}_{apple0} = 1.2 - 3 = -1.8$$

$$\hat{\beta}_1 x = \hat{\alpha}_{orange1} x - \hat{\alpha}_{apple1} x = -2 - 0.6 = -2.6$$

E

In [ ]:

```

```

Question 4.14

In [414]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import glm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, precis
import statsmodels.formula.api as smf
import statsmodels.api as sm
from patsy import dmatrices
import sklearn.linear_model as skl_lm
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold, c
from sklearn.preprocessing import PolynomialFeatures
import math
from scipy.stats import ttest_ind
from scipy.stats import ttest_1samp
import scipy.stats
%matplotlib inline
```

In [51]:

```
df = pd.read_csv("desktop/Auto.csv")
```

In [343]:
```
df
```

Out[343]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg0
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	
...	...	...	...	...	...	...	...	...	...	.
387	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	
388	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	
389	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	
390	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	
391	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	

392 rows × 10 columns

In [53]:
```
df.mpg.median()
```

Out[53]: 22.75

In [54]:
```
df["mpg01"] = (df["mpg"] >= df["mpg"].median()).astype(int)
```

In [55]:
```
df
```

Out[55]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg0
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	
...	...	...	...	...	...	...	...	...	...	.
387	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	
388	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	
389	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	
390	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	
391	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	

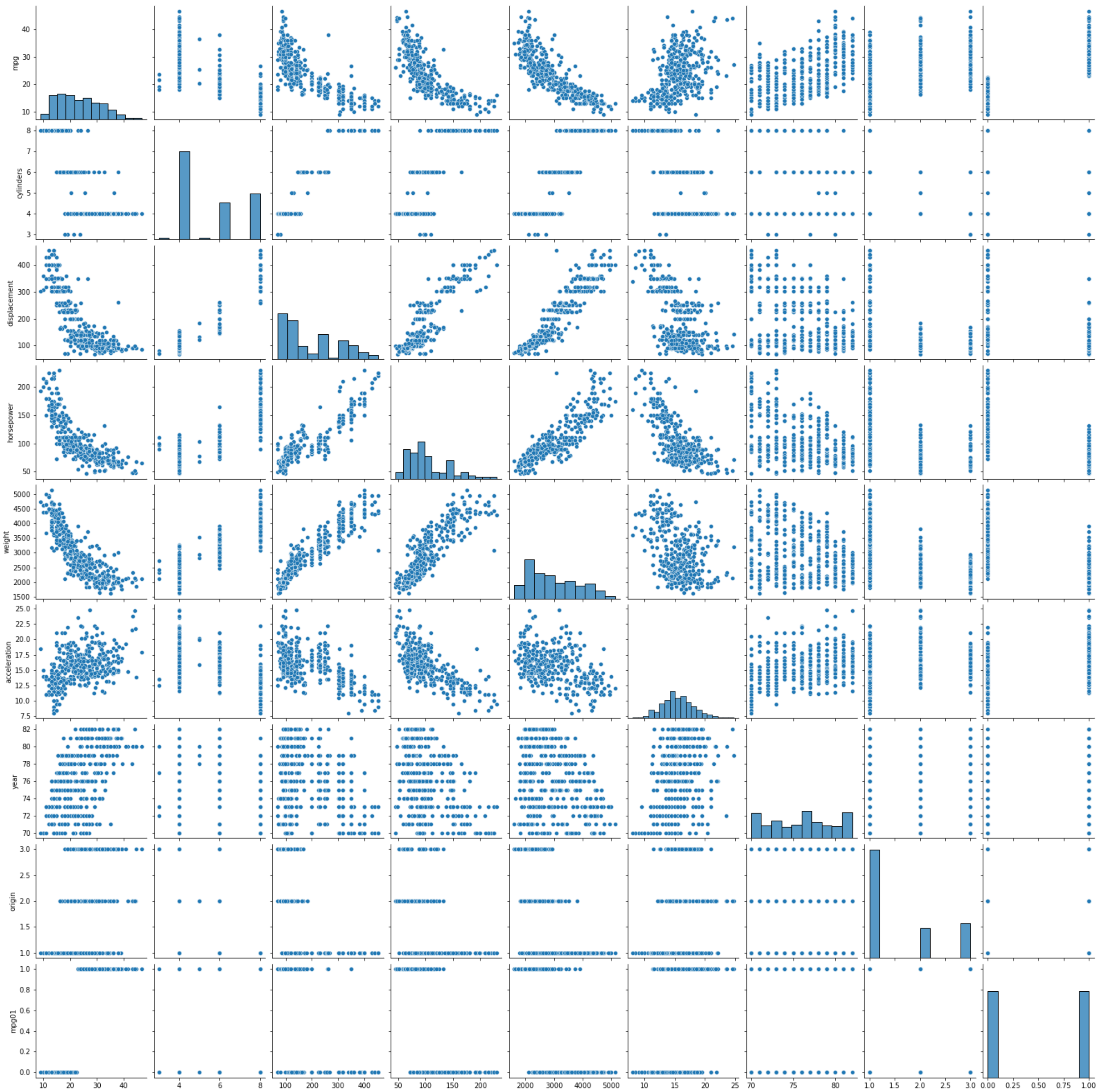
392 rows × 10 columns

In [56]:
```
df.corr()
```

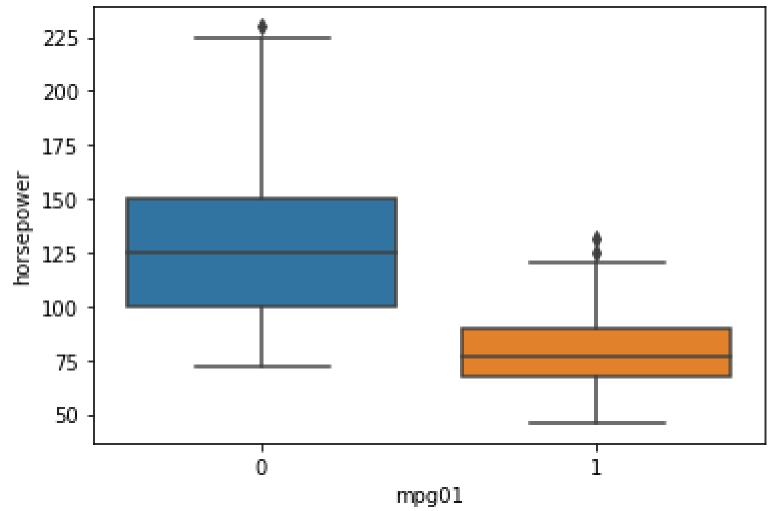
Out[56]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	
mpg	1.000000	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	0.580541	
cylinders	-0.777618	1.000000	0.950823	0.842983	0.897527	-0.504683	-0.345647	-
displacement	-0.805127	0.950823	1.000000	0.897257	0.932994	-0.543800	-0.369855	-
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361	-
weight	-0.832244	0.897527	0.932994	0.864538	1.000000	-0.416839	-0.309120	-
acceleration	0.423329	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	0.290316	
year	0.580541	-0.345647	-0.369855	-0.416361	-0.309120	0.290316	1.000000	
origin	0.565209	-0.568932	-0.614535	-0.455171	-0.585005	0.212746	0.181528	
mpg01	0.836939	-0.759194	-0.753477	-0.667053	-0.757757	0.346822	0.429904	

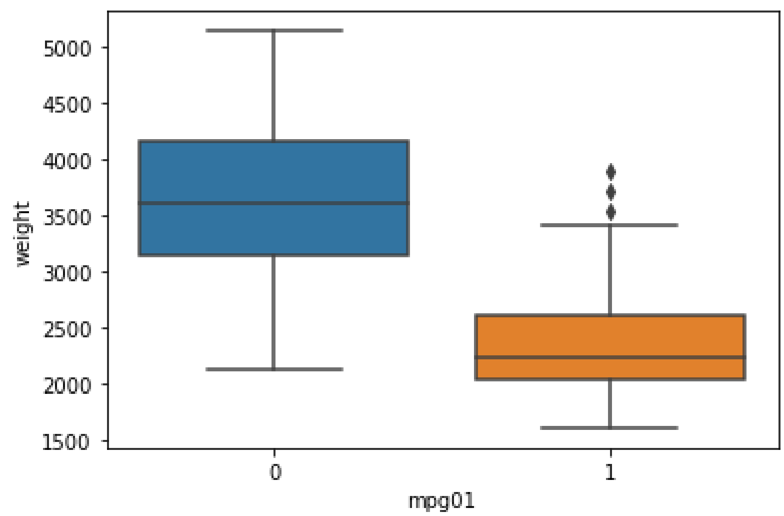
In [57]:
```
sns.pairplot(df);
```



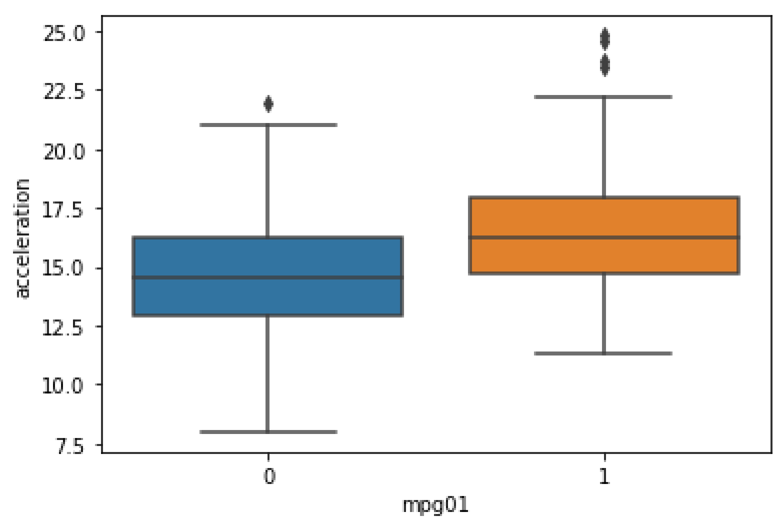
In [58]:
```
sns.boxplot(x='mpg01', y='horsepower', data=df);
```



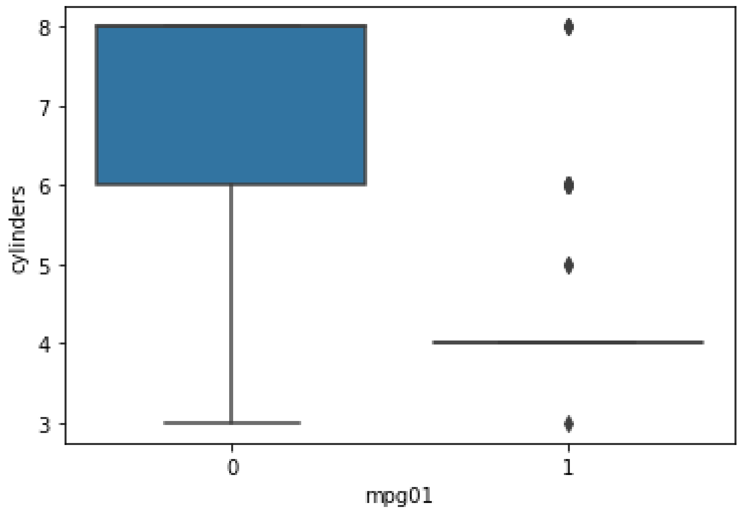
In [59]:
```
sns.boxplot(x='mpg01', y='weight', data=df);
```



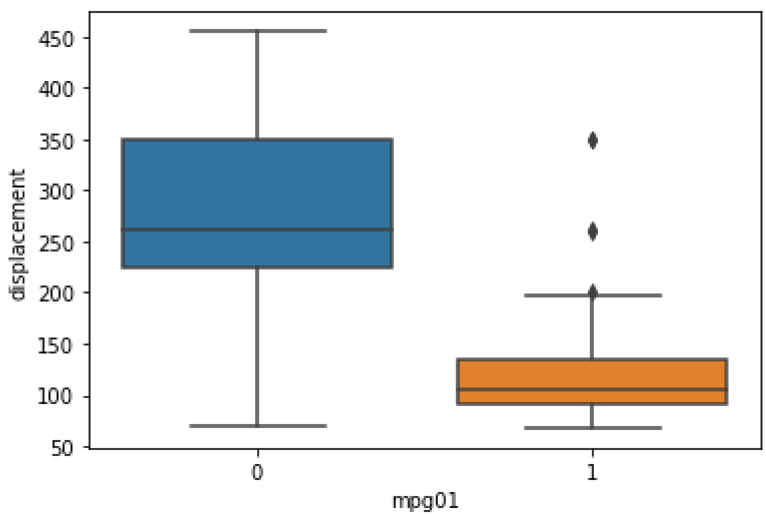
In [60]:
```
sns.boxplot(x='mpg01', y='acceleration', data=df);
```



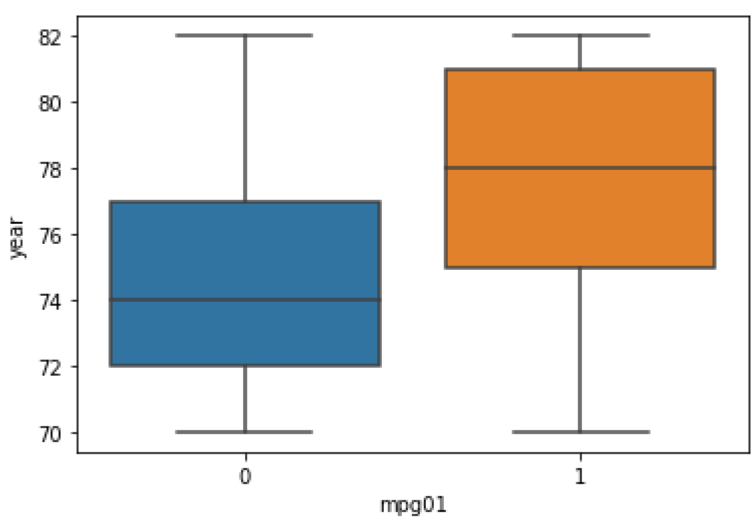
In [61]:
```
sns.boxplot(x='mpg01', y='cylinders', data=df);
```



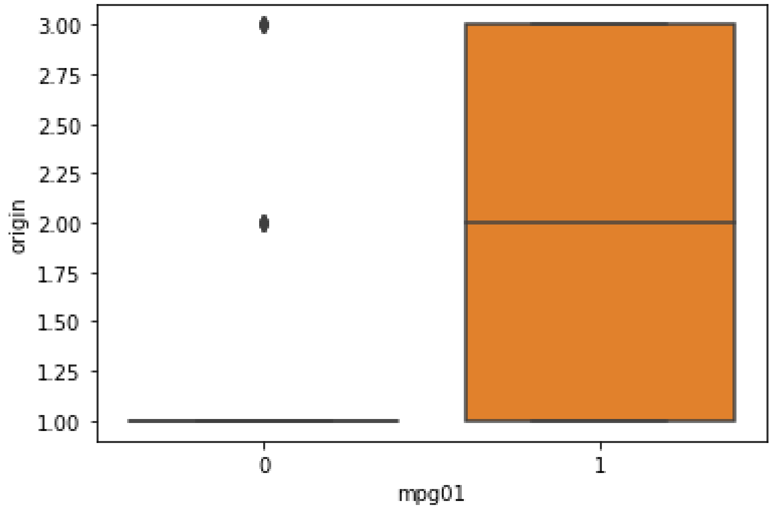
In [62]:
```
sns.boxplot(x='mpg01', y='displacement', data=df);
```



In [63]:
```
sns.boxplot(x='mpg01', y='year', data=df);
```



In [64]:
```
sns.boxplot(x='mpg01', y='origin', data=df);
```



Based on the correlations, boxplots and pairplots, I will pick certain variables that seem to be good predictors for mpg01 and run a logistic model to check the significance.

In [65]:
```
y, X = dmatrices('mpg01 ~ weight + horsepower + cylinders + displacement ',
logit = sm.Logit(y, X)
results_logit = logit.fit()
print(results_logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.264373
         Iterations 9
                           Logit Regression Results
==========================================================================
=====
Dep. Variable:                 mpg01   No. Observations:
392
Model:                         Logit   Df Residuals:
387
Method:                          MLE   Df Model:
4
Date:               Wed, 13 Apr 2022   Pseudo R-squ.:
0.6186
Time:                       15:47:23   Log-Likelihood:                    -1
03.63
converged:                      True   LL-Null:                           -2
71.71
Covariance Type:           nonrobust   LLR p-value:                     1.70
6e-71
==========================================================================
=======
                   coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-------
Intercept       11.7966      1.709      6.902      0.000       8.447
15.146
weight          -0.0019      0.001     -2.812      0.005      -0.003
-0.001
horsepower      -0.0421      0.014     -3.015      0.003      -0.070
-0.015
cylinders       -0.0129      0.346     -0.037      0.970      -0.691
0.665
displacement    -0.0130      0.008     -1.579      0.114      -0.029
0.003
==========================================================================
=======
```

According to the p-values, horsepower and weight are the only statistically significant variables.

In [66]:
```
#Partitioning the dataset
df_7076 = df[(df['year'] >=70) & (df['year'] <=76)]
df_7782 = df[(df['year'] >=77) & (df['year'] <=82)]
```

In [67]:
```
#training set
X1 = df_7076[['weight','horsepower']]
lda = LinearDiscriminantAnalysis()
lda.fit(X1,df_7076['mpg01'])
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                           solver='svd', store_covariance=False, tol=0.0001)
```

Out[67]:
```
LinearDiscriminantAnalysis()
```

In [68]:
```
# Testing Set
X1_test = df_7782[['weight','horsepower']]

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], lda.predict(X1_test))
print(conf_mat)
lda.score(X1_test, df_7782['mpg01'])
print('Accuracy =', lda.score(X1_test, df_7782['mpg01']))
```

```
[[ 50   4]
 [ 22 102]]
Accuracy = 0.8539325842696629
```

In [69]:
```
X1 = df_7076[['weight','horsepower']]
qda = QuadraticDiscriminantAnalysis()
qda.fit(X1,df_7076['mpg01'])
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                              store_covariance=False, tol=0.0001)
```

Out[69]:
```
QuadraticDiscriminantAnalysis()
```

In [70]:
```
# Testing Set
X1_test = df_7782[['weight','horsepower']]

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], qda.predict(X1_test))
print(conf_mat)
qda.score(X1_test, df_7782['mpg01'])
print('Accuracy =', qda.score(X1_test, df_7782['mpg01']))
```

```
[[52  2]
 [26 98]]
Accuracy = 0.8426966292134831
```

In [71]:
```
X1 = df_7076[['weight','horsepower']]
lr = LogisticRegression()

mod = lr.fit(X1,df_7076['mpg01'])
```

In [72]:
```
# Testing Set
X1_test = df_7782[['weight','horsepower']]

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], lr.predict(X1_test))
print(conf_mat)
#overall fraction of correct predictions
lr.score(X1_test, df_7782['mpg01'])
print('Accuracy =', lr.score(X1_test, df_7782['mpg01']))
```

```
[[53  1]
 [37 87]]
Accuracy = 0.7865168539325843
```

In [73]:
```
# Training set
X1 = df_7076[['weight','horsepower']]

# Naive Bayes Fit
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X1, df_7076['mpg01'])
```

Out[73]:
```
GaussianNB()
```

In [74]:
```
# Testing Set
X1_test = df_7782[['weight','horsepower']]
y_pred = classifier.predict(X1_test)

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], y_pred)
print(conf_mat)
accuracy_score(df_7782['mpg01'], y_pred)
print('Accuracy =', accuracy_score(df_7782['mpg01'], y_pred))
```

```
[[ 53   1]
 [ 23 101]]
Accuracy = 0.8651685393258427
```

In [75]:
```
# Training set
X1 = df_7076[['weight','horsepower']]

# KNN Fit
nbrs = KNeighborsClassifier(n_neighbors=1)
nbrs.fit(X1,df_7076['mpg01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Out[75]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=1)
```

In [76]:
```
# Testing Set
X1_test = df_7782[['weight','horsepower']]

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], nbrs.predict(X1_test))
print(conf_mat)
nbrs.score(X1_test, df_7782['mpg01'])
print('Accuracy =', nbrs.score(X1_test, df_7782['mpg01']))
```

```
[[50  4]
 [30 94]]
Accuracy = 0.8089887640449438
```

In [77]:
```
# Training set
X1 = df_7076[['weight','horsepower']]

# KNN Fit
nbrs = KNeighborsClassifier(n_neighbors=2)
nbrs.fit(X1,df_7076['mpg01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

Out[77]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=2)
```

In [78]:
```
# Testing Set
X1_test = df_7782[['weight','horsepower']]

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], nbrs.predict(X1_test))
print(conf_mat)
nbrs.score(X1_test, df_7782['mpg01'])
print('Accuracy =', nbrs.score(X1_test, df_7782['mpg01']))
```

```
[[53  1]
 [48 76]]
Accuracy = 0.7247191011235955
```

In [172]:
```
# Training set
X1 = df_7076[['weight','horsepower']]

# KNN Fit
nbrs = KNeighborsClassifier(n_neighbors=4)
nbrs.fit(X1,df_7076['mpg01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='uniform')
```

Out[172]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=4)
```

In [173]:
```python
# Testing Set
X1_test = df_7782[['weight','horsepower']]

# Confusion matrix
conf_mat = confusion_matrix(df_7782['mpg01'], nbrs.predict(X1_test))
print(conf_mat)
nbrs.score(X1_test, df_7782['mpg01'])
print('Accuracy =', nbrs.score(X1_test, df_7782['mpg01']))
```

```
[[52  2]
 [45 79]]
Accuracy = 0.7359550561797753
```

K=1 performed better compared to other values of K. Naive Bayes had the highest accuracy and lowest error compared to all other fits.

Question 4.16

In [81]:
```python
df1 = pd.read_csv("desktop/Boston.csv")
```

In [157]:
```python
df1
```

0	1	0.00632	18.0	2.31	0	0.538	6.575	65.199997	4.0900	1.0	296	15.3	396.9
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.900002	4.9671	2.0	242	17.8	396.9
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.099998	4.9671	2.0	242	17.8	392.8
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.799999	6.0622	3.0	222	18.7	394.6
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.200001	6.0622	3.0	222	18.7	396.9
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	502	0.06263	0.0	11.93	0	0.573	6.593	69.099998	2.4786	1.0	273	21.0	391.9
502	503	0.04527	0.0	11.93	0	0.573	6.120	76.699997	2.2875	1.0	273	21.0	396.9
503	504	0.06076	0.0	11.93	0	0.573	6.976	91.000000	2.1675	1.0	273	21.0	396.9
504	505	0.10959	0.0	11.93	0	0.573	6.794	89.300003	2.3889	1.0	273	21.0	393.4
505	506	0.04741	0.0	11.93	0	0.573	6.030	80.800003	2.5050	1.0	273	21.0	396.9

506 rows × 16 columns

In [138]:
```python
df1["crim01"] = (df1["crim"] >= df1["crim"].median()).astype(int)
```

In [136]:
```python
df1['crim01'] = df1['crim01'].map({'df1["crim"] <= df1["crim"].median()':0,
```

In [344]:
```
df1
```

Out[344]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	bla
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.199997	4.0900	1.0	296	15.3	396
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.900002	4.9671	2.0	242	17.8	396
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.099998	4.9671	2.0	242	17.8	392
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.799999	6.0622	3.0	222	18.7	394
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.200001	6.0622	3.0	222	18.7	396
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	502	0.06263	0.0	11.93	0	0.573	6.593	69.099998	2.4786	1.0	273	21.0	391
502	503	0.04527	0.0	11.93	0	0.573	6.120	76.699997	2.2875	1.0	273	21.0	396
503	504	0.06076	0.0	11.93	0	0.573	6.976	91.000000	2.1675	1.0	273	21.0	396
504	505	0.10959	0.0	11.93	0	0.573	6.794	89.300003	2.3889	1.0	273	21.0	393
505	506	0.04741	0.0	11.93	0	0.573	6.030	80.800003	2.5050	1.0	273	21.0	396

506 rows × 16 columns

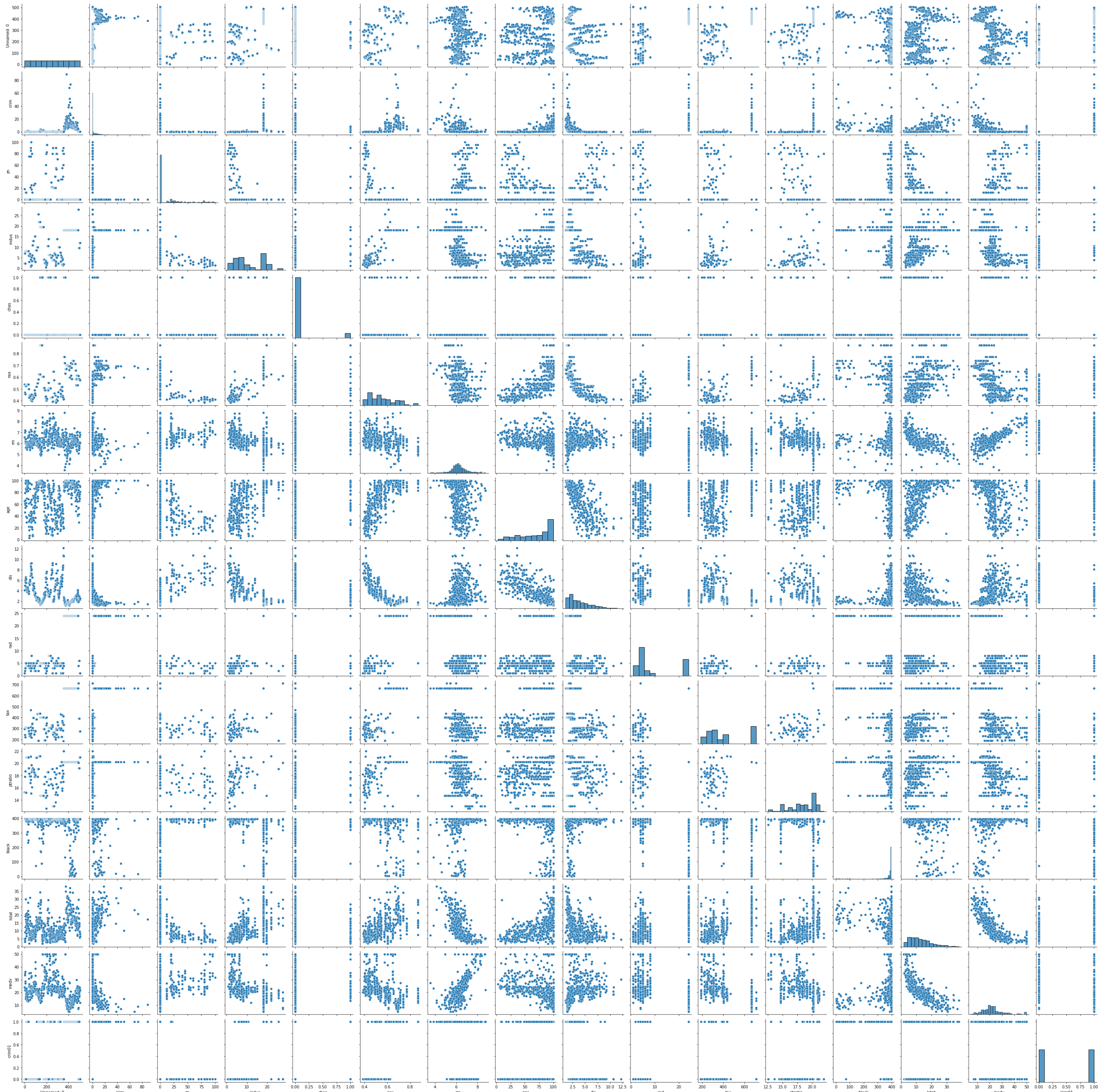
In [ ]:
```
df1.
```

In [106]:
```
df1.corr()
```

Out[106]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age
Unnamed: 0	1.000000	0.407407	-0.103393	0.399439	-0.003759	0.398736	-0.079971	0.203784
crim	0.407407	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734
zn	-0.103393	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537
indus	0.399439	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779
chas	-0.003759	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518
nox	0.398736	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470
rm	-0.079971	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265
age	0.203784	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000
dis	-0.302211	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881
rad	0.686002	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022
tax	0.666626	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456
ptratio	0.291074	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515
black	-0.295041	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534
lstat	0.258465	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339
medv	-0.226604	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955
crim01	0.369430	0.409395	-0.436151	0.603260	0.070097	0.723235	-0.156372	0.613940

In [111]:
```
sns.pairplot(df1);
```



Based on high correlation and observable patterns with crim01, I chose the variables listen in the regression below.

In [114]:
```
y1, X2 = dmatrices('crim01 ~ indus + nox + dis + tax + lstat + rad ', data=
logit = sm.Logit(y1, X2)
results1_logit = logit.fit()
print(results1_logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.244068
         Iterations 10
                           Logit Regression Results
==========================================================================
=====
Dep. Variable:                 crim01   No. Observations:
506
Model:                          Logit   Df Residuals:
499
Method:                           MLE   Df Model:
6
Date:                Wed, 13 Apr 2022   Pseudo R-squ.:
0.6479
Time:                        16:14:19   Log-Likelihood:                 -1
23.50
converged:                       True   LL-Null:                        -3
50.73
Covariance Type:            nonrobust   LLR p-value:                  5.36
1e-95
==========================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-----
Intercept    -24.5694      3.733     -6.582      0.000     -31.886      -1
7.253
indus         -0.0583      0.043     -1.355      0.175      -0.143
0.026
nox           43.9943      6.822      6.449      0.000      30.623       5
7.365
dis            0.1562      0.146      1.073      0.283      -0.129
0.442
tax           -0.0072      0.002     -3.071      0.002      -0.012       -
0.003
lstat          0.0170      0.031      0.555      0.579      -0.043
0.077
rad            0.6051      0.118      5.141      0.000       0.374
0.836
==========================================================================
=====

Possibly complete quasi-separation: A fraction 0.28 of observations can b
e
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

Based on the p-values, I will continue with indus, nox, and rad.

In [113]:
```
df1_50 = df1[(df1['age'] >=0) & (df1['age'] <=50)]
df1_100 = df1[(df1['age'] >=51) & (df1['age'] <=100)]
```

In [115]:
```
#training set
X2 = df1_50[['nox','indus','rad']]
lda = LinearDiscriminantAnalysis()
lda.fit(X2,df1_50['crim01'])
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
              solver='svd', store_covariance=False, tol=0.0001)
```

Out[115]:
```
LinearDiscriminantAnalysis()
```

In [116]:
```
# Testing Set
X2_test = df1_100[['nox','indus','rad']]

# Confusion matrix
conf_mat = confusion_matrix(df1_100['crim01'], lda.predict(X2_test))
print(conf_mat)
lda.score(X2_test, df1_100['crim01'])
print('Accuracy =', lda.score(X2_test, df1_100['crim01']))
```

```
[[ 86  36]
 [ 40 197]]
Accuracy = 0.7883008356545961
```

In [178]:
```
# Training set
X2 = df1_50[['nox','indus','rad']]

# Naive Bayes Fit

classifier = GaussianNB()
classifier.fit(X2, df1_50['crim01'])
```

Out[178]:
```
GaussianNB()
```

In [179]:
```
# Testing Set
X2_test = df1_100[['nox','indus','rad']]
y1_pred = classifier.predict(X2_test)

# Confusion matrix
conf_mat = confusion_matrix(df1_100['crim01'], y1_pred)
print(conf_mat)
accuracy_score(df1_100['crim01'], y1_pred)
print('Accuracy =', accuracy_score(df1_100['crim01'], y1_pred))
```

```
[[ 86  36]
 [ 30 207]]
Accuracy = 0.8161559888579387
```

In [186]:
```
Specificity_NB = 207/(207+36)
```

In [187]:
```
Specificity_NB
```

Out[187]:
```
0.8518518518518519
```

In [121]:
```
X2 = df1_50[['nox','indus','rad']]
lr = LogisticRegression()
mod1 = lr.fit(X2,df1_50['crim01'])
```

In [122]:
```
# Testing Set
X2_test = df1_100[['nox','indus','rad']]

# Confusion matrix
conf_mat = confusion_matrix(df1_100['crim01'], lr.predict(X2_test))
print(conf_mat)

lr.score(X2_test, df1_100['crim01'])
print('Accuracy =', lr.score(X2_test, df1_100['crim01']))
```

```
[[110  12]
 [ 63 174]]
Accuracy = 0.7910863509749304
```

In [180]:
```
# Training set
X2 = df1_50[['nox','indus','rad']]

# KNN Fit
nbrs1 = KNeighborsClassifier(n_neighbors=1)
nbrs1.fit(X2,df1_50['crim01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Out[180]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=1)
```

In [181]:
```

X2_test = df1_100[['nox','indus','rad']]


conf_mat = confusion_matrix(df1_100["crim01"], nbrs1.predict(X2_test))
print(conf_mat)
nbrs1.score(X2_test, df1_100['crim01'])
print('Accuracy =', nbrs1.score(X2_test, df1_100['crim01']))
```

```
[[ 92  30]
 [ 58 179]]
Accuracy = 0.754874651810585
```

In [182]:
```
# Training set
X2 = df1_50[['nox','indus','rad']]

# KNN Fit
nbrs2 = KNeighborsClassifier(n_neighbors=2)
nbrs2.fit(X2,df1_50['crim01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

Out[182]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=2)
```

In [183]:
```
X2_test = df1_100[['nox','indus','rad']]


conf_mat = confusion_matrix(df1_100["crim01"], nbrs2.predict(X2_test))
print(conf_mat)
nbrs2.score(X2_test, df1_100['crim01'])
print('Accuracy =', nbrs2.score(X2_test, df1_100['crim01']))
```

```
[[118   4]
 [ 62 175]]
Accuracy = 0.8161559888579387
```

In [184]:
```
Specificity_KNN2 = 175/(175+4)
```

In [185]:
```
Specificity_KNN2
```

Out[185]:
```
0.9776536312849162
```

In [174]:
```
# Training set
X2 = df1_50[['nox','indus','rad']]

# KNN Fit
nbrs3 = KNeighborsClassifier(n_neighbors=3)
nbrs3.fit(X2,df1_50['crim01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
```

Out[174]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=3)
```

In [175]:
```
X2_test = df1_100[['nox','indus','rad']]


conf_mat = confusion_matrix(df1_100["crim01"], nbrs3.predict(X2_test))
print(conf_mat)
nbrs3.score(X2_test, df1_100['crim01'])
print('Accuracy =', nbrs3.score(X2_test, df1_100['crim01']))
```

```
[[ 89  33]
 [ 59 178]]
Accuracy = 0.7437325905292479
```

In [176]:
```
# Training set
X2 = df1_50[['nox','indus','rad']]

# KNN Fit
nbrs4 = KNeighborsClassifier(n_neighbors=4)
nbrs4.fit(X2,df1_50['crim01'])
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='uniform')
```

Out[176]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=4)
```

In [177]:
```
X2_test = df1_100[['nox','indus','rad']]


conf_mat = confusion_matrix(df1_100["crim01"], nbrs4.predict(X2_test))
print(conf_mat)
nbrs4.score(X2_test, df1_100['crim01'])
print('Accuracy =', nbrs4.score(X2_test, df1_100['crim01']))
```

```
[[101  21]
 [ 86 151]]
Accuracy = 0.7019498607242339
```

Since KNN with K=2 and the Naive Bayes models had the same exact accuracy, I decided choose a model based on specificity and therefore, I picked KNN with K=2.

Question 5.7

In [188]:
```
df2 = pd.read_csv("desktop/Weekly.csv")
```

In [346]:
```
df2['Direction'] = df2['Direction'].map({'Down':0, 'Up':1})
```

In [347]:
```
df2
```

Out[347]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	0
1	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	0
2	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	1
3	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	1
4	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	1
...	...	...	...	...	...	...	...	...	...
1084	2010	-0.861	0.043	-2.173	3.599	0.015	3.205160	2.969	1
1085	2010	2.969	-0.861	0.043	-2.173	3.599	4.242568	1.281	1
1086	2010	1.281	2.969	-0.861	0.043	-2.173	4.835082	0.283	1
1087	2010	0.283	1.281	2.969	-0.861	0.043	4.454044	1.034	1
1088	2010	1.034	0.283	1.281	2.969	-0.861	2.707105	0.069	1

1089 rows × 9 columns

In [348]:
```python
mod1 = smf.glm(formula='Direction ~ Lag1 + Lag2', data=df2, family=sm.famil
print(mod1.summary())
```

```
                 Generalized Linear Model Regression Results
==========================================================================
=====
Dep. Variable:              Direction   No. Observations:
1089
Model:                            GLM   Df Residuals:
1086
Model Family:                Binomial   Df Model:
2
Link Function:                  logit   Scale:
1.0000
Method:                          IRLS   Log-Likelihood:                 -7
44.11
Date:                Thu, 14 Apr 2022   Deviance:                        1
488.2
Time:                        10:55:53   Pearson chi2:                  1.0
9e+03
No. Iterations:                     4
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-----
Intercept      0.2212      0.061      3.599      0.000       0.101
0.342
Lag1          -0.0387      0.026     -1.477      0.140      -0.090
0.013
Lag2           0.0602      0.027      2.270      0.023       0.008
0.112
==========================================================================
=====
```

In [349]:
```
mod2 = smf.glm(formula='Direction ~ Lag1 + Lag2', data=df2.drop(0), family=
print(mod2.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
=====
Dep. Variable:              Direction   No. Observations:
1088
Model:                            GLM   Df Residuals:
1085
Model Family:                Binomial   Df Model:
2
Link Function:                  logit   Scale:
1.0000
Method:                          IRLS   Log-Likelihood:                    -7
43.26
Date:                Thu, 14 Apr 2022   Deviance:                           1
486.5
Time:                        10:55:55   Pearson chi2:                     1.0
9e+03
No. Iterations:                     4
Covariance Type:            nonrobust
==============================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
Intercept      0.2232      0.061      3.630      0.000       0.103
0.344
Lag1          -0.0384      0.026     -1.466      0.143      -0.090
0.013
Lag2           0.0608      0.027      2.291      0.022       0.009
0.113
==============================================================================
=====
```

In [382]:
```
prediction = []

["Up" if x < 0.5 else "Down" for x in predictions]
```

In [388]:
```
predictions_nominal = [ 0 if x < 0.5 else 1 for x in predictions]
predictions_nominal
```

```
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
```

In [389]:
```
predictions_nominal[0]
```

Out[389]: 1

No, the observation was not correctly classified

In [410]:
```
error = []


for i in range(0,len(df2)-1):
    mod3 = smf.glm(formula='Direction ~ Lag1 + Lag2', data=df2.drop(i), fam
    predictions_nominal[i] = [ 1 if mod3.predict()[i] > 0.5 else 0] #for i
    if (predictions_nominal[i]==df2.iloc[i, 8]):
        error.append(0)
    else:
        error.append(1)

#mod3.summary()
```

In [411]:
```
error
```

```
 1,
 1,
 0,
 1,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 0,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
```

In [412]:
```
np.mean(error)
```

Out[412]:
```
0.5045955882352942
```

In [*]:
```
p_order = np.arange(1,11)
r_state = np.arange(0,10)


mod3 = lr.fit(df2[["Lag1","Lag2"]],df2['Direction'])
loo = LeaveOneOut()
loo.get_n_splits(df2)
scores = list()

for i in p_order:
    poly = PolynomialFeatures(i)
    X_poly = poly.fit_transform(df2[["Lag1","Lag2"]])
    score = cross_val_score(mod3, X_poly, df2.Direction, cv=loo, scoring='n
    scores.append(score)
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/l
inear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  n_iter_i = _check_optimize_result(
/Users/youssefmahmoud/opt/anaconda3/lib/python3.8/site-packages/sklearn/l
inear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to converg
e (status=1):
```

Question 5.9

In [381]:
```
df1
```

Out[381]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	bl
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.199997	4.0900	1.0	296	15.3	396
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.900002	4.9671	2.0	242	17.8	396
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.099998	4.9671	2.0	242	17.8	392
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.799999	6.0622	3.0	222	18.7	394
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.200001	6.0622	3.0	222	18.7	396
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	502	0.06263	0.0	11.93	0	0.573	6.593	69.099998	2.4786	1.0	273	21.0	391
502	503	0.04527	0.0	11.93	0	0.573	6.120	76.699997	2.2875	1.0	273	21.0	396
503	504	0.06076	0.0	11.93	0	0.573	6.976	91.000000	2.1675	1.0	273	21.0	396
504	505	0.10959	0.0	11.93	0	0.573	6.794	89.300003	2.3889	1.0	273	21.0	393
505	506	0.04741	0.0	11.93	0	0.573	6.030	80.800003	2.5050	1.0	273	21.0	396

506 rows × 16 columns

In [228]:
```
mu_medv = df1.medv.mean()
```

In [229]:
```
mu_medv
```

Out[229]: 22.532806324110698

In [235]:
```
std_medv = df1.medv.std()
```

In [236]:
```
std_medv
```

Out[236]: 9.19710408737982

In [237]:
```
SE_medv = (std_medv/506**0.5)
```

In [238]:
```
SE_medv
```

Out[238]: 0.4088611474975351

In [213]:
```
medv = df1["medv"]
```

In [218]:
```
T = 506
```

In [219]:
```
np.random.choice(medv, replace = True, size = T)
```

Out[219]:
```
array([16.7, 13.2, 24.6, 24.8, 27.5, 18.2, 20.6, 22.3, 15. , 24.2, 29.1,
       23.6, 15.7, 18.7, 13. , 19.9, 16.1, 23.1, 22.8, 48.5, 11.7, 30.8,
       14.5, 50. , 11.7, 21.7, 25. , 16.8, 21. , 19.1, 26.6, 50. , 20.8,
       30.3, 24.6, 50. , 20.8, 17.5, 17.1, 34.9, 11.9, 24.7, 31.6, 19. ,
       14.4, 10.4, 25.2, 27.5, 23. , 20. ,  7. , 19.4, 23.9, 19.4, 17.9,
       21.8, 35.4, 26.6, 13.1, 14.9, 20.6, 34.6, 23.6, 14.1, 13.4, 12.1,
       19. , 22.9, 16.3, 24.5, 20.4, 20.8, 22.6, 16.1, 24.1, 42.8, 22.5,
       19.8, 25. , 22.2, 14.1, 19.9, 22.8, 34.9, 13.3, 14.1, 19.5, 21.5,
       34.9, 19.3, 24.1, 16.7, 10.5, 48.3, 14.4, 13.1, 23.9, 24.7, 33. ,
       15.6,  7.4, 31.5, 31.1, 11.7, 24.5, 13.1,  6.3, 34.7, 19.3, 10.9,
        5. , 50. , 22.2, 11.7, 31.6, 18.9, 17.1, 15.1, 14.3, 15.2, 15.2,
       16.8,  7. , 22.3, 21. , 17.4, 30.3,  5. , 15.6, 13.8, 16.1, 19.1,
        7.4, 16.6, 48.8, 19.4, 17.5, 13.8, 13.6, 50. , 12. , 19.6, 19.1,
       18.1,  7.4, 20.2, 22.9, 20.8, 44.8, 36.2, 20.6, 37.6, 18.6, 23.8,
       24.4, 22.6, 13.1, 23.4, 50. , 23.8, 35.1, 17.7, 21.2, 18.7, 17.4,
       22.6, 17.4, 11.9, 17.1, 22.8, 24.8, 21.9, 13.9,  5. , 43.1, 13.8,
       21.1, 32. , 23.7, 33.3, 24.8, 30.5, 16.2, 28.7, 24.4, 32.4, 23.7,
       20. , 25. , 14.5, 23.8, 23.8, 22.4, 31.7, 20.9, 24.3, 13.8, 19.7,
       11.7, 29.8, 36.1, 29.1, 14.8, 16.2, 14.9, 50. , 10.4, 24.2, 28.5,
       16.1, 34.7, 13.8, 39.8, 12.6, 20.8, 36.5, 22.2, 26.6, 16.8, 23.1,
       18.8, 12.7, 10.9, 21.5, 17.8, 24.4, 13.1, 18.6, 24.7, 22.8, 18.7,
       24.7, 24.4, 22.6,  7.2, 10.2, 21.7, 13.8, 23.2, 10.4, 17.8, 31.1,
       17.1, 23.4,  5. , 17.8, 20.7, 11.7, 24.3, 14.1, 16.4, 12.8, 23.9,
       19.1, 16.1,  7.4, 44. , 23.2, 23.1, 37.9, 28.7, 14.9, 22. , 15. ,
       24.8, 15.2, 19.1, 24.7, 20. , 20.4, 20.2, 35.4, 10.9, 20.4, 23.6,
       18.4, 19.5, 14.8, 50. , 35.1, 20.9, 18.4, 23.2, 50. , 50. , 16.8,
       35.4, 31.5, 21.4, 18.7, 24.5, 22.2, 10.5, 36.2, 16.1, 15.6,  7.2,
       21.4, 22.5, 21.8, 18.5, 27.5, 27.9, 23.7, 23.3, 26.5, 19.5, 36. ,
       36.2, 20.4, 14.6, 21.7, 14.3,  8.4, 23.7, 13.2, 21.9, 16.4, 19.3,
       50. ,  8.3, 23.6, 20.1, 14.5, 17.6, 42.3, 50. , 24.6, 18.6, 23.1,
        5. , 16.7, 12.1, 26.2, 19.4, 18.9, 17.2, 19.8, 20.8, 18.8, 38.7,
       50. , 11.8, 15.6,  9.7, 29.1, 10.2, 25.2, 23.9, 13.5, 15. , 19.7,
       25. , 50. , 19.5, 29. , 22. , 15. , 32.5, 21.2, 30.3, 22.2, 27.5,
       13.3, 25.1, 17.1, 31.5, 13.8, 18.7, 19.9, 42.3, 25. ,  9.7, 24.8,
       20. , 16.2, 12.8, 29.6, 23.1, 32.5, 20.1, 19.1, 24.1, 16.2, 24.7,
       16.1, 39.8, 23.1, 28.4, 24.3, 32.2, 20. , 20.8, 32. , 31. , 17.8,
       48.8,  6.3, 24.5, 31.7, 43.8, 35.2, 15.7, 24.2, 15.1, 19.2, 36.1,
        8.1, 23.3, 23.7, 27.5, 13.6, 14.5, 20.3, 35.4, 50. , 14.8, 29.9,
        8.3, 17.4, 20.6, 22.2, 21.4, 23.2, 20.6, 32.9, 10.9, 17.7, 19.6,
       23.4, 28.6, 22.1, 23.1, 22.1, 33.2, 21.4, 48.8, 14.5, 22.9, 23.7,
       16.8, 48.5, 19.4, 24.8, 20.9, 14. , 33.1, 19.8, 12.8, 23.1, 22.5,
       19.7, 18.9, 31.2, 36.4, 16.2, 13.9, 21.7, 48.8, 12.7, 22. , 23. ,
        7.5, 25. , 24.4, 17.9, 16.2, 19.9, 21.8, 10.5, 15.2, 21.4, 17.1,
       20.6, 17.8, 20.9, 15. , 29.8, 13.8, 20.3, 26.6,  8.8, 18.7, 15.1,
       29. , 24.5, 20. , 19.6, 20.4, 13.8, 23.3, 19.6, 13.8, 34.9, 21.2,
       33.1, 13.9, 23.3, 28.6, 25.3, 16.5,  7.4, 27.1, 33.2,  8.4, 19.8])
```

In [241]:
```
np.random.seed(123) # for reproducible randomness
B = 50000
boot_samples = np.zeros((T, B))
for i in range(B):
    boot_samples[:, i] = np.random.choice(medv, replace = True, size = T) #

boot_samples[:, 0:5] # first 5 bootstrap samples (first 5 columns of boot_s
```

Out[241]:
```
array([[27.5, 20.5, 31.6, 31.6, 10.2],
       [11.3, 21.1, 11.9, 23.1, 20.1],
       [20.4, 50. , 45.4, 23.3, 35.1],
       ...,
       [30.1, 18.2, 32.9,  7.4, 32. ],
       [48.3, 19.7,  7.2, 20.9, 17.2],
       [23. , 18.5, 18.2, 14.5, 16.7]])
```

In [269]:
```
mu_hat, SE_hat = np.mean(medv), (np.std(medv, ddof = 1))/math.sqrt(T)
print("The sample mean and standard error are {:.4f} and {:.4f}, resp.".for
# {} refers to a variable, and :.4f rounds the number off to 4 decimal plac
print("=====================================================================

boot_means = np.mean(boot_samples, axis = 0) # apply function np.mean() to
boot_SEs = (np.std(boot_samples, ddof = 1, axis = 0))/math.sqrt(T) # apply

print('First 5 bootstrap sample means: ', np.round(boot_means[:5], 4)) # ro
print('First 5 bootstrap sample standard errors: ', np.round(boot_SEs[:5],
```

```
The sample mean and standard error are 22.5328 and 0.4089, resp.
======================================================================
First 5 bootstrap sample means:  [22.1239 22.8585 22.734  21.9553 22.603
4]
First 5 bootstrap sample standard errors:  [0.3852 0.4117 0.4103 0.3888
0.4161]
```

The sample SE calculated in part(b) is very close to the bootstrap Standard Errors.

In [249]:
```
Medv_confint = [mu_medv-(2*SE_medv),mu_medv+(2*SE_medv)]
```

In [250]:
```
Medv_confint
```

Out[250]:
```
[21.715084029115626, 23.35052861910577]
```

In [264]:
```
t_test = ttest_1samp(medv, 22.538 )
```

In [265]:
```
t_test.pvalue
```

Out[265]:
```
0.9898699319560071
```

Based on the P-value, we fail to reject that 22.538 is the mean of medv, which is consistent with our findings.

In [266]:
```
med_medv = df1.medv.median()
```

In [267]:
```
med_medv
```

Out[267]: 21.2

In [272]:
```
np.random.seed(123) # for reproducible randomness
B = 50000
boot_samples1 = np.zeros((T, B))
for i in range(B):
    boot_samples1[:, i] = np.random.choice(medv, replace = True, size = T)

boot_samples1[:, 0:6] # first 5 bootstrap samples (first 5 columns of boot_
```

Out[272]:
```
array([[27.5, 20.5, 31.6, 31.6, 10.2, 33.8],
       [11.3, 21.1, 11.9, 23.1, 20.1, 15.6],
       [20.4, 50. , 45.4, 23.3, 35.1, 13.8],
       ...,
       [30.1, 18.2, 32.9,  7.4, 32. , 36.1],
       [48.3, 19.7,  7.2, 20.9, 17.2, 30.1],
       [23. , 18.5, 18.2, 14.5, 16.7, 14.6]])
```

In [290]:
```
med_medv = np.median(medv)
print("The sample median {:.4f}".format(med_medv))
# {} refers to a variable, and :.4f rounds the number off to 4 decimal plac
print("=================================================================

boot_medians = np.median(boot_samples1, axis = 0) # apply function np.mean(
print('First 5 bootstrap sample medians: ', np.round(boot_medians[:5], 4))
```

```
The sample median 21.2000
======================================================================
First 5 bootstrap sample medians:  [20.8   21.7   21.7   21.15 21.   ]
```

In [293]:
```
Median_SE = np.std(boot_medians, ddof = 1)
```

In [294]:
```
Median_SE
```

Out[294]: 0.37793954673997743

In [288]:
```
mu_10th = np.percentile(medv,10)
```

In [289]:
```
mu_10th
```

Out[289]: 12.75

In [297]:
```
mu_10th = np.percentile(medv,10)
print("The 10th percentile is {:.4f}".format(mu_10th))
# {} refers to a variable, and :.4f rounds the number off to 4 decimal plac
print("==================================================================

boot_quantile = np.percentile(boot_samples1 ,10, axis = 0) # apply function
print('First 5 bootstrap sample of 10th percentiles: ', np.round(boot_quant
```

```
The 10th percentile is 12.7500
======================================================================
First 5 bootstrap sample of 10th percentiles:  [13.3 13.1 12.4 12.5 13.3]
```

In [298]:
```
Percentile_SE = np.std(boot_quantile, ddof = 1)
```

In [299]:
```
Percentile_SE
```

Out[299]:
```
0.5039022670616743
```

In [ ]: