Section 4.7

Conceptual

1. $$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \quad \text{— (4.2)}$$ [Logistic ~~representation~~ function representation]

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X} \quad \text{— (4.3)}$$ [logit representation of logistic regression]

To prove: $(4.2) \equiv (4.3)$

~~4.3~~ $$1-p(X) = 1 - \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$ [Using (4.2)]

$$\therefore 1-p(X) = \frac{1+e^{\beta_0+\beta_1 X} - e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} = \frac{1}{1+e^{\beta_0+\beta_1 X}}$$

$$\therefore \frac{p(X)}{1-p(X)} = \left(\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}\right) \div \left(\frac{1}{1+e^{\beta_0+\beta_1 X}}\right)$$

$$\therefore \frac{p(X)}{1-p(X)} = \left(\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}\right)\left(1+e^{\beta_0+\beta_1 X}\right)$$

$$\therefore \frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X} \equiv (4.3)$$

Hence proved (4.3) using (4.2).

5.(a) If the Bayes decision boundary is linear, we expect QDA to perform better on the training set. On the test set, we expect LDA to perform better. This would be the case as QDA is likely to overfit since it has greater flexibility, leading to lower training error, since it used the training data to build the model which would be very close/specific to the training data, however, would not perform well for a test/unseen set since it is too specific to the training data, leading to the LDA having better performance since the decision boundary is linear.

(b) If the Bayes decision boundary is non-linear, we expect or QDA to perform better on the training set as well as the test set as LDA is built to model linear decision boundaries whereas the greater flexibility provided by QDA would better mimic the non- linear boundary on both the training and test set.

(c) In general, as the sample size n increases, we expect the test prediction accuracy of QDA relative to LDA to improve since QDA is a more complex model requiring a larger training set to prevent overfitting and improve performance on test data.

or be unchanged? Why?

(d) The statement 'Even if the Bayes decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible' is false because LDA is made for modelling linear decision boundaries. In this case a QDA model which is more complex than LDA may overfit and probably give a higher test error since the model is specific to the traing data provided and incorporates noise in the model.

8. We should prefer the method giving a lower test error since that (test) is the 'unseen' data and can be considered to mimic real world data. Based on the results provided, the test error for logistic regression is 30%. For k nearest neighbours, the model is having k=1, this would lead to a training error of zero since the data provided to test the model would be the training data and hence the same point provided would be chosen as the nearest neighbour leading to accurate predictions, however in reality this model has overfit. Since the training and test datasets are of equal size, we can conclude that k nearest neighbours would have a test error of 36% (test error+train error=18*2 where training error is zero), leading us to prefer the logistic model over k nearest neighbours.

Applied

13. (a) From the correlation plot, we can observe that the correlation among all attributes besides year and volume is weak, allowing us to conclude that no significant relationship exists between any attributes besides year and volume.

We proceed to plot volumes vs year and volume (since multiple readings exist corresponding to a single year), where we can see that volume increases with year in general until around 2008n after which there is a slight drop.

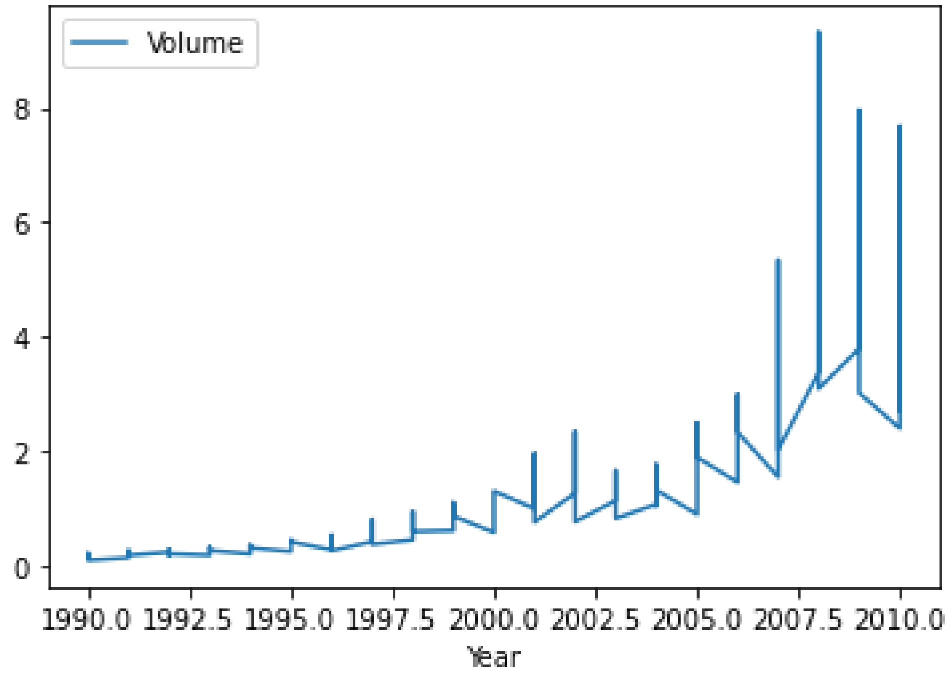
13 (a)

In [22]:
```
#correlation plot to observe correlation among the features
Weekly.corr()
```

Out[22]:

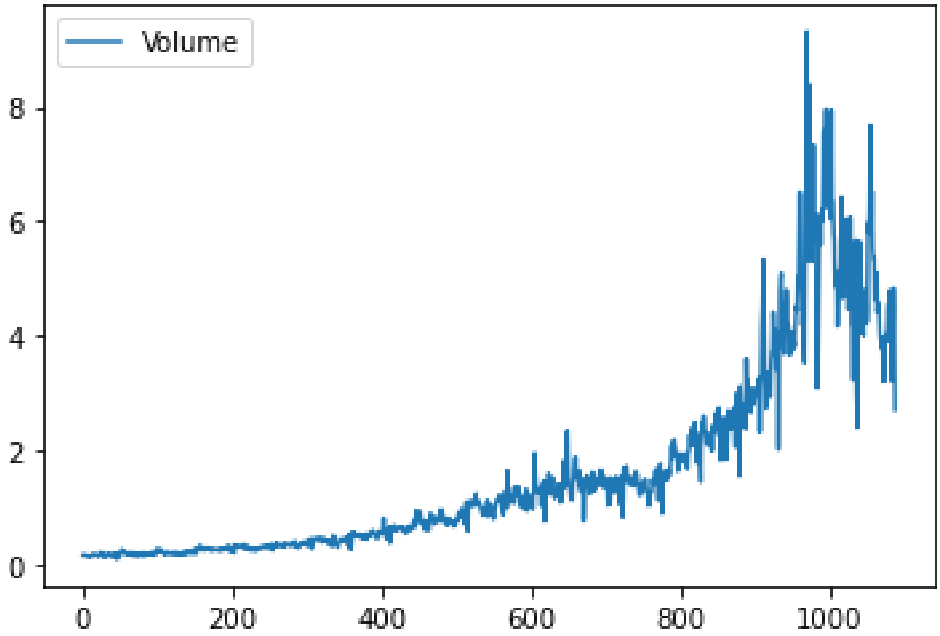
	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today
Year	1.000000	-0.032289	-0.033390	-0.030006	-0.031128	-0.030519	0.841942	-0.032460
Lag1	-0.032289	1.000000	-0.074853	0.058636	-0.071274	-0.008183	-0.064951	-0.075032
Lag2	-0.033390	-0.074853	1.000000	-0.075721	0.058382	-0.072499	-0.085513	0.059167
Lag3	-0.030006	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

In [24]:
```
Weekly.plot(y='Volume',x='Year');
```



In [25]:
```
Weekly.plot(y='Volume')
```

Out[25]: <AxesSubplot:>



(b) From the summary plot, we can conclude that only lag 2 is statistically significant since the other predictors (lag1, lag3, lag4, lag5 and volume) have p values greater than 0.05 allowing us to accept the null hypothesis for them (coefficient corresponding to the predictor in the model being 0, i.e. no relation to the output).

13 (b)

In [27]:
```
predictors = Weekly.columns.drop(['Today', 'Direction', 'Year'])
design = MS(predictors)
X = design.fit_transform(Weekly)
y = Weekly.Direction == 'Up'
glm = sm.GLM(y,
             X,
             family=sm.families.Binomial())
results = glm.fit()
summarize(results)
```

Out[27]:

	coef	std err	z	P>|z|
intercept	0.2669	0.086	3.106	0.002
Lag1	-0.0413	0.026	-1.563	0.118
Lag2	0.0584	0.027	2.175	0.030
Lag3	-0.0161	0.027	-0.602	0.547
Lag4	-0.0278	0.026	-1.050	0.294
Lag5	-0.0145	0.026	-0.549	0.583
Volume	-0.0227	0.037	-0.616	0.538

(c) From the confusion matrix corresponding to the logistic regression (model trained and evaluated), we can see that the model accuracy or the percent of times that the model makes correct predictions is 56.1, however when we calculate the percent of correct predictions on the basis of the labels (count of correct predictions of label/ count of label), we notice a stark difference with the model performing considerably better when predicting the label 'up'.

13 (c)

In [30]:
```
#obtaining probability of market going up
probs = results.predict()

#creating the array with labels
labels = np.array(['Down']*1089)
labels[probs>0.5] = "Up"

labels
```

Out[30]:
```
array(['Up', 'Up', 'Up', ..., 'Up', 'Up', 'Up'], dtype='<U4')
```

In [31]:
```
confusion_table(labels, Weekly.Direction)
```

Out[31]:

Truth	Down	Up
Predicted		
Down	54	48
Up	430	557

In [32]:
```
print("overall fraction of correct predictions=",(54+557)/(1089))
```
```
overall fraction of correct predictions= 0.5610651974288338
```

In [36]:
```
print('error rate is',100-100*np.mean(labels == Weekly.Direction))
```
```
error rate is 43.89348025711662
```

In [40]:
```
print('overall fraction of correct predictions for the label "up"=',(557)/(557+48))
```
```
overall fraction of correct predictions for the label "up"= 0.9206611570247933
```

In [41]:
```
print('overall fraction of correct predictions for the label "down"=',(54)/(54+430))
```
```
overall fraction of correct predictions for the label "down"= 0.1115702479338843
```

(d)

13 (d)

In [63]:
```
#using lag2 as predictor
allvars = Weekly.columns.drop(['Year', 'Lag1', 'Lag3','Lag4','Lag5','Volume','Today','Directi
design = MS(allvars)
X = design.fit_transform(Weekly)
y = Weekly.Direction == 'Up'
```

In [65]:
```
train_data=Weekly.loc[Weekly['Year'] <2009]
train_data.tail()
```

Out[65]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
980	2008	12.026	-8.389	-6.198	-3.898	10.491	5.841565	-2.251	Down
981	2008	-2.251	12.026	-8.389	-6.198	-3.898	6.093950	0.418	Up
982	2008	0.418	-2.251	12.026	-8.389	-6.198	5.932454	0.926	Up
983	2008	0.926	0.418	-2.251	12.026	-8.389	5.855972	-1.698	Down
984	2008	-1.698	0.926	0.418	-2.251	12.026	3.087105	6.760	Up

In [66]:
```
test_data=train_data=Weekly.loc[Weekly['Year'] >2008]
test_data.head()
```

Out[66]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
985	2009	6.760	-1.698	0.926	0.418	-2.251	3.793110	-4.448	Down
986	2009	-4.448	6.760	-1.698	0.926	0.418	5.043904	-4.518	Down
987	2009	-4.518	-4.448	6.760	-1.698	0.926	5.948758	-2.137	Down
988	2009	-2.137	-4.518	-4.448	6.760	-1.698	6.129763	-0.730	Down
989	2009	-0.730	-2.137	-4.518	-4.448	6.760	5.602004	5.173	Up

In [68]:
```
train = (Weekly.Year < 2009)
Weekly_train = Weekly.loc[train]
Weekly_test = Weekly.loc[~train]
Weekly_test.shape
```

Out[68]: (104, 9)

In [70]:
```
X_train, X_test = X.loc[train], X.loc[~train]
y_train, y_test = y.loc[train], y.loc[~train]
```

In [71]:
```
D = Weekly.Direction
L_train, L_test = D.loc[train], D.loc[~train]
```

In [72]:
```
model = MS(['Lag2']).fit(Weekly)
X = model.transform(Weekly)
X_train, X_test = X.loc[train], X.loc[~train]
glm_train = sm.GLM(y_train,
                   X_train,
                   family=sm.families.Binomial())
results = glm_train.fit()
```

In [73]:
```
#confusion matrix for test data
probs = results.predict(exog=X_test)
labels = np.array(['Down']*104)
labels[probs >0.5] = 'Up'
confusion_table(labels, L_test)
```

Out[73]:

Truth / Predicted	Down	Up
Down	9	5
Up	34	56

In [141]:
```
print('overall fraction of correct predictions for the held out data= ',(9+56)/(14+34+56))
```

```
overall fraction of correct predictions for the held out data=  0.625
```

(e)

(e)

In [24]:
```
#LDA model
lda = LDA(store_covariance=True)
```

In [25]:
```
X_train, X_test = [M.drop(columns=['intercept'])
                   for M in [X_train, X_test]]
lda.fit(X_train, L_train)
```

Out[25]:
```
LinearDiscriminantAnalysis
LinearDiscriminantAnalysis(store_covariance=True)
```

In [26]:
```
lda.means_
```

Out[26]:
```
array([[-0.03568254],
       [ 0.26036581]])
```

In [27]:
```
lda.classes_
```

Out[27]:
```
array(['Down', 'Up'], dtype='<U4')
```

In [28]:
```
lda.priors_
```

Out[28]:
```
array([0.44771574, 0.55228426])
```

In [29]:
```
lda.scalings_
```

Out[29]:
```
array([[0.44141622]])
```

In [30]:
```
lda_pred = lda.predict(X_test)
```

In [31]:
```
confusion_table(lda_pred, L_test)
```

Out[31]:

Truth / Predicted	Down	Up
Down	9	5
Up	34	56

```
In [32]: print('overall fraction of correct predictions for the held out data= ',(9+56)/(14+34+56))
```

```
overall fraction of correct predictions for the held out data=  0.625
```

(f)

(f)

```
In [35]: qda = QDA(store_covariance=True)
         qda.fit(X_train, L_train)
```

```
Out[35]: QuadraticDiscriminantAnalysis
         QuadraticDiscriminantAnalysis(store_covariance=True)
```

```
In [36]: qda.means_, qda.priors_
```

```
Out[36]: (array([[-0.03568254],
                 [ 0.26036581]]),
          array([0.44771574, 0.55228426]))
```

```
In [37]: qda.covariance_[0]
```

```
Out[37]: array([[4.83781758]])
```

```
In [38]: qda_pred = qda.predict(X_test)
         confusion_table(qda_pred, L_test)
```

Out[38]:

Truth	Down	Up
Predicted		
Down	0	0
Up	43	61

```
In [39]: np.mean(qda_pred == L_test)
```

```
Out[39]: 0.5865384615384616
```

```
In [47]: print('overall fraction of correct predictions for the held out data= ',(61)/(14+34+56))
```

```
overall fraction of correct predictions for the held out data=  0.5865384615384616
```

(g)

(g)

```
In [41]: #needs scaling as distance based
         #scaler = StandardScaler(with_mean=True,
                 #with_std=True,
                # copy=True)
```

```
In [55]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         print(X_train_scaled.shape)
         print(X_test_scaled.shape)
```

```
(985, 1)
(104, 1)
```

```
In [57]: from sklearn import metrics
         knn1 = KNeighborsClassifier(n_neighbors=1)
         knn1_model = knn1.fit(X_train_scaled, y_train)
         y_pred = knn1_model.predict(X_test_scaled)
         score = metrics.accuracy_score(y_test,y_pred)
         #.predict(scaled_test)
         print(score)
         #np.mean(y_test != knn1_pred)

         0.49038461538461536
```

```
In [61]: #False corresponds to 'down' label, the 'up' label is represented by true
         confusion_table(y_pred, y_test)

Out[61]:
```

Truth	False	True
Predicted		
False	22	32
True	21	29

```
In [62]:
         confusion_table( y_test, y_pred)

Out[62]:
```

Truth	False	True
Predicted		
False	22	21
True	32	29

```
In [60]: metrics.confusion_matrix(y_test, y_pred)
         #print("Confusion Matrix:")
         #print(result)

Out[60]: array([[22, 21],
                [32, 29]], dtype=int64)
```

```
In [68]: print('overall fraction of correct predictions for the held out data= ',(22+29)/(14+34+56))

         overall fraction of correct predictions for the held out data=  0.49038461538461536
```

(i) Logistic regression and LDA dive the best results on the data with an accuracy of 62.5%.

(j) Among the 2 interaction models, the second one built using LDA performs better with an accuracy of around 58.65%, while trying different values of k, k=13 results in the highest accuracy of around 58.65%

(j)

In [66]:
```
#import statsmodels.api as sm
X_interaction1 = MS(['Lag1',
  'Year','Lag2',
    ('Lag1', 'Lag2'),('Lag2', 'Year'),('Lag2', 'Year')]).fit_transform(Weekly)
print(X_interaction1)
#summarize(model_interaction1.fit())
X_interaction1_train=X_interaction1.loc[X_interaction1['Year'] <2009]
X_interaction1_train.tail()
X_interaction1_test=X_interaction1.loc[X_interaction1['Year'] >2008]
```

```
      intercept   Lag1  Year   Lag2  Lag1:Lag2  Lag2:Year  Lag2:Year
0           1.0  0.816  1990  1.572   1.282752    3128.28    3128.28
1           1.0 -0.270  1990  0.816  -0.220320    1623.84    1623.84
2           1.0 -2.576  1990 -0.270   0.695520    -537.30    -537.30
3           1.0  3.514  1990 -2.576  -9.052064   -5126.24   -5126.24
4           1.0  0.712  1990  3.514   2.501968    6992.86    6992.86
...         ...    ...   ...    ...        ...        ...        ...
1084        1.0 -0.861  2010  0.043  -0.037023      86.43      86.43
1085        1.0  2.969  2010 -0.861  -2.556309   -1730.61   -1730.61
1086        1.0  1.281  2010  2.969   3.803289    5967.69    5967.69
1087        1.0  0.283  2010  1.281   0.362523    2574.81    2574.81
1088        1.0  1.034  2010  0.283   0.292622     568.83     568.83

[1089 rows x 7 columns]
```

In [67]:
```
qda_interaction = QDA(store_covariance=True)
qda_interaction.fit(X_interaction1_train, L_train)
qda_pred_i = qda_interaction.predict(X_interaction1_test)
confusion_table(qda_pred_i, L_test)
```

```
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:935: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:960: RuntimeWarning: divide by zero encountered in
power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:960: RuntimeWarning: invalid value encountered in m
ultiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:963: RuntimeWarning: divide by zero encountered in
log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
```

Out[67]:

Truth	Down	Up
Predicted		
Down	43	61
Up	0	0

In [68]:
```
print('overall fraction of correct predictions for the held out data= ',(43)/(14+34+56))
```

```
overall fraction of correct predictions for the held out data=  0.41346153846153844
```

In [69]:
```
result = metrics.classification_report(L_test, qda_pred_i)
print("Classification Report:",result)
```

```
Classification Report:               precision    recall  f1-score   support

        Down       0.41      1.00      0.59        43
          Up       0.00      0.00      0.00        61

    accuracy                           0.41       104
   macro avg       0.21      0.50      0.29       104
weighted avg       0.17      0.41      0.24       104
```

```
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ishaj\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [70]:
```
#import statsmodels.api as sm
X_interaction2 = MS(['Lag1','Year',
  'Lag4','Lag2',
    ('Lag1', 'Lag2'),('Lag2', 'Lag4'),('Lag2', 'Lag4')]).fit_transform(Weekly)
print(X_interaction2)
#summarize(model_interaction1.fit())
X_interaction2_train=X_interaction2.loc[X_interaction2['Year'] <2009]
X_interaction2_train.tail()
X_interaction2_test=X_interaction2.loc[X_interaction2['Year'] >2008]
```

```
      intercept   Lag1  Year   Lag4   Lag2  Lag1:Lag2  Lag2:Lag4  Lag2:Lag4
0           1.0  0.816  1990 -0.229  1.572   1.282752  -0.359988  -0.359988
1           1.0 -0.270  1990 -3.936  0.816  -0.220320  -3.211776  -3.211776
2           1.0 -2.576  1990  1.572 -0.270   0.695520  -0.424440  -0.424440
3           1.0  3.514  1990  0.816 -2.576  -9.052064  -2.102016  -2.102016
4           1.0  0.712  1990 -0.270  3.514   2.501968  -0.948780  -0.948780
...         ...    ...   ...    ...    ...        ...        ...        ...
1084        1.0 -0.861  2010  3.599  0.043  -0.037023   0.154757   0.154757
1085        1.0  2.969  2010 -2.173 -0.861  -2.556309   1.870953   1.870953
1086        1.0  1.281  2010  0.043  2.969   3.803289   0.127667   0.127667
1087        1.0  0.283  2010 -0.861  1.281   0.362523  -1.102941  -1.102941
1088        1.0  1.034  2010  2.969  0.283   0.292622   0.840227   0.840227

[1089 rows x 8 columns]
```

In [71]:
```
#LDA model
lda_interaction = LDA(store_covariance=True)
lda_interaction.fit(X_interaction2_train, L_train)
lda_pred_i = lda_interaction.predict(X_interaction2_test)
confusion_table(lda_pred_i, L_test)
```

Out[71]:

Truth	Down	Up
Predicted		
Down	21	21
Up	22	40

In [72]:
```
print('overall fraction of correct predictions for the held out data= ',(21+40)/(14+34+56))
```

```
overall fraction of correct predictions for the held out data=  0.5865384615384616
```

In [73]:
```
result = metrics.classification_report(L_test, lda_pred_i)
print("Classification Report:",result)
```

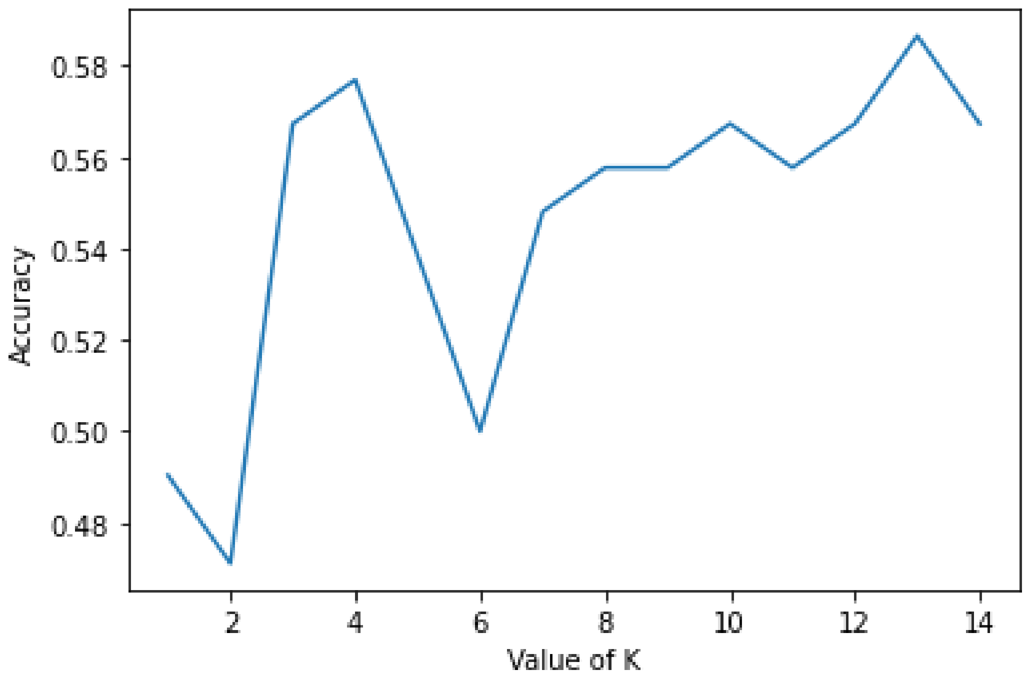
```
Classification Report:               precision    recall  f1-score   support

        Down       0.50      0.49      0.49        43
          Up       0.65      0.66      0.65        61

    accuracy                           0.59       104
   macro avg       0.57      0.57      0.57       104
weighted avg       0.59      0.59      0.59       104
```

In [74]:
```
#import patsy
#y, X = patsy.dmatrices('Direction ~ Lag2 + Lag1 + Year + Lag2:Year', train_data)
#print(Direction,y)

#qda = QDA(store_covariance=True)
#qda.fit(X, y)

#print(y)
```

In [75]:
```
range_k = range(1,15)

scores_list = []
for k in range_k:
   classifier = KNeighborsClassifier(n_neighbors=k)
   classifier.fit(X_train_scaled, y_train)
   y_pred = classifier.predict(X_test_scaled)

   scores_list.append(metrics.accuracy_score(y_test,y_pred))

print (scores_list)

%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(range_k,scores_list)
plt.xlabel("Value of K")
plt.ylabel("Accuracy")

#k=13 results in the highest accuracy of around 58.65%
```

```
[0.49038461538461536, 0.47115384615384615, 0.5673076923076923, 0.5769230769230769, 0.5384615384615384, 0.5, 0.5480769230769231, 0.5576923076923077, 0.5576923076923077, 0.5673076923076923, 0.5576923076923077, 0.5673076923076923, 0.5865384615384616, 0.56730
76923076923]
```

Out[75]: `Text(0, 0.5, 'Accuracy')`



14.(a)

14

In [67]:
```
Auto = load_data('Auto')
Auto.tail(5)
```

Out[67]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
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

(a)

In [68]:
```
median_mileage=Auto['mpg'].median()
print(median_mileage)
```

```
22.75
```

In [69]:
```
Auto['mpg01'] = [1 if x>median_mileage else 0 for x in Auto['mpg']]
Auto.tail(5)
```

Out[69]:

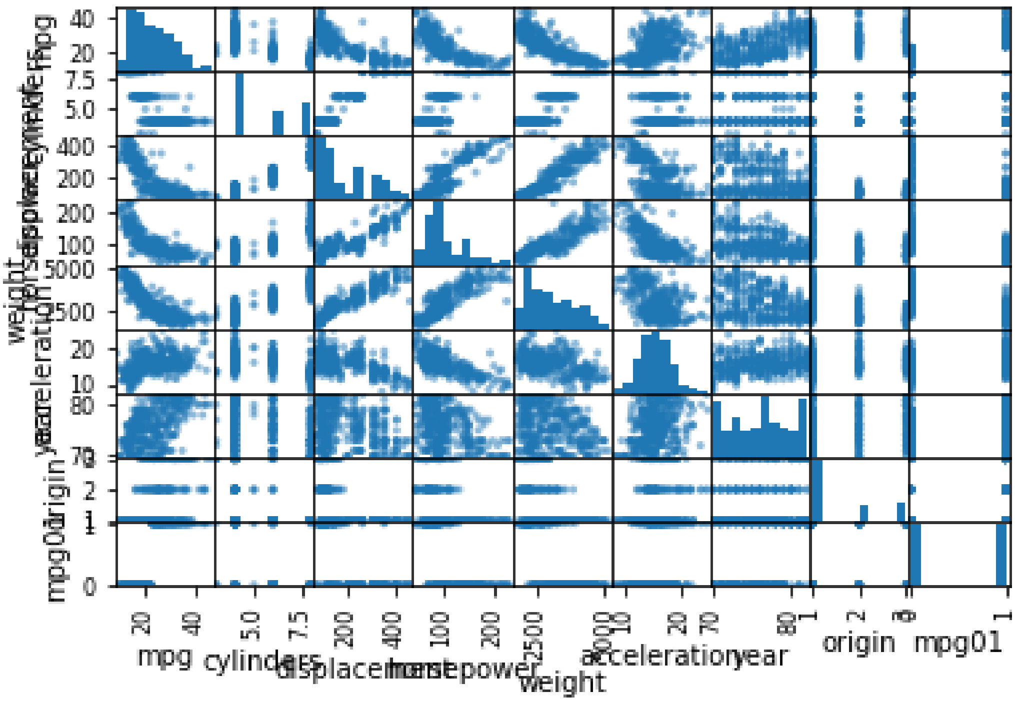
	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg01
387	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	1
388	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	1
389	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	1
390	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	1
391	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	1

(b) Based on the plots below, 'acceleration', 'weight', 'displacement' and 'horsepower' seem most likely to be useful in predicting mpg01.

(b)

In [70]:
```
pd.plotting.scatter_matrix(Auto)
```

Out[70]:
```
array([[<AxesSubplot:xlabel='mpg', ylabel='mpg'>,
```

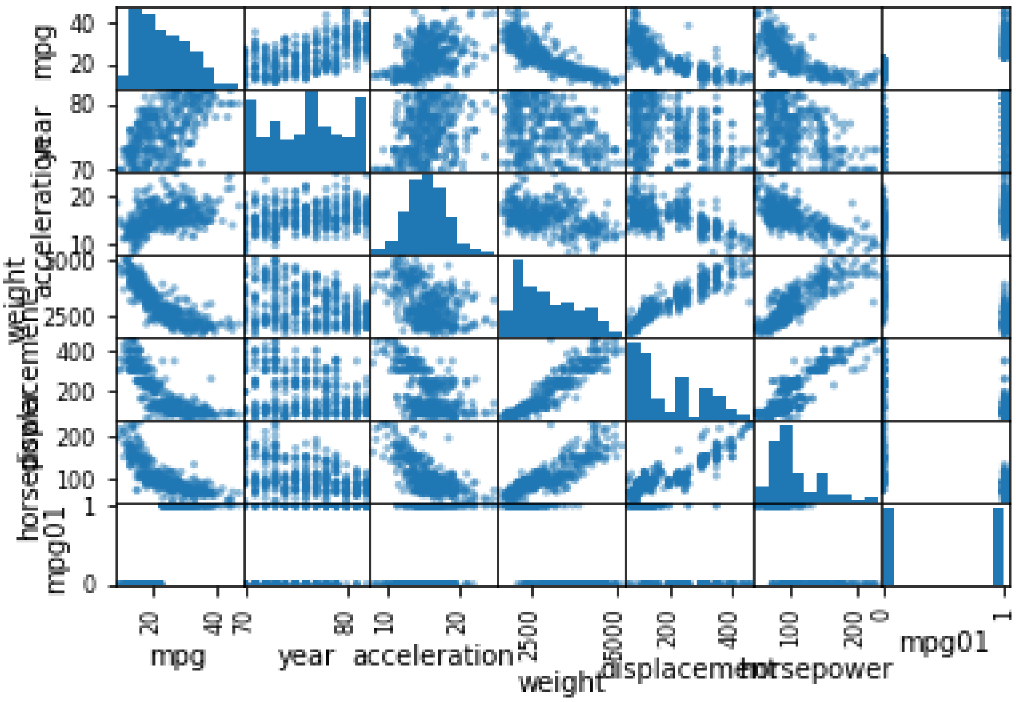


In [71]:
```
#dropping name as it is a qualitative variable
#dropping cylinder and origin (as they are categorical variables with a small number of categories) from the sc
small_auto_df=Auto[['mpg','year', 'acceleration','weight','displacement','horsepower','mpg01']].copy()
small_auto_df.head(5)
```

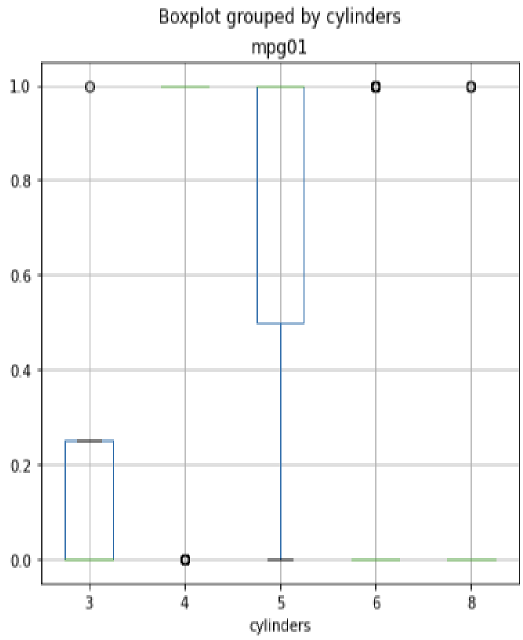
Out[71]:

	mpg	year	acceleration	weight	displacement	horsepower	mpg01
0	18.0	70	12.0	3504	307.0	130	0
1	15.0	70	11.5	3693	350.0	165	0
2	18.0	70	11.0	3436	318.0	150	0
3	16.0	70	12.0	3433	304.0	150	0
4	17.0	70	10.5	3449	302.0	140	0

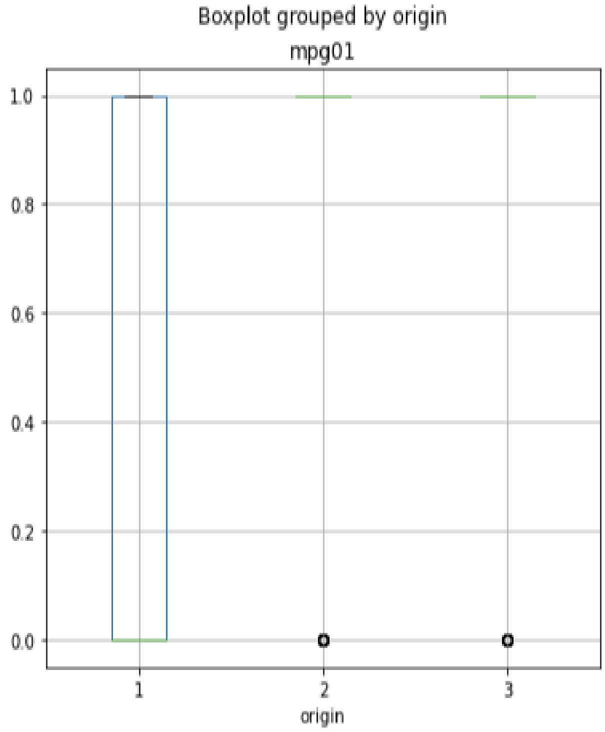
In [72]:
```
pd.plotting.scatter_matrix(small_auto_df)
```



In [73]:
```
#import subplots from matplotlib.pyplot
#plotting mpg vs cylinders as a boxplot and origin as the number of categories is small
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 6))
Auto.boxplot('mpg01', by='cylinders', ax=ax);
```



```
In [74]: import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 6))
Auto.boxplot('mpg01', by='origin', ax=ax);
```



(c)

(c)

```
In [75]: #creating test set with 30 percent of the dataset
from sklearn.model_selection import train_test_split
#not using name
#splitting into features and labels, not using mpg as a predictor as it can be considere
X = Auto[[ 'acceleration','weight','displacement','horsepower']]
y = Auto['mpg01']

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)
```

(d)

(d)

```
In [81]: #LDA model
         lda = LDA(store_covariance=True)

In [82]: lda.fit(X_train,y_train)

Out[82]: ▾ LinearDiscriminantAnalysis
         LinearDiscriminantAnalysis(store_covariance=True)

In [83]: lda_pred = lda.predict(X_test)

In [84]: confusion_table(lda_pred, y_test)

Out[84]:
```

Truth	0	1
Predicted		
0	44	6
1	10	58

```
In [85]: print("overall fraction of correct predictions using LDA=",(44+58)/(44+10+6+58))

         overall fraction of correct predictions using LDA= 0.864406779661017

In [86]: result = metrics.classification_report(y_test, lda_pred)
         print("Classification Report:",result)

         Classification Report:               precision    recall  f1-score   support

                    0       0.88      0.81      0.85        54
                    1       0.85      0.91      0.88        64

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

The test error of the LDA model is around 100*(1-0.8644), i.e.,13.56%.

(e)

(e)

```
In [87]: qda = QDA(store_covariance=True)
         qda.fit(X_train, y_train)
```

```
Out[87]: QuadraticDiscriminantAnalysis
         QuadraticDiscriminantAnalysis(store_covariance=True)
```

```
In [88]: qda_pred = qda.predict(X_test)
         confusion_table(qda_pred, y_test)
```

Out[88]:

Truth	0	1
Predicted		
0	45	8
1	9	56

```
In [89]: print("overall fraction of correct predictions using QDA=",(45+56)/(44+10+6+58))

overall fraction of correct predictions using QDA= 0.8559322033898306
```

```
In [90]: result = metrics.classification_report(y_test, qda_pred)
         print("Classification Report:",result)

Classification Report:               precision    recall  f1-score   support

           0       0.85      0.83      0.84        54
           1       0.86      0.88      0.87        64

    accuracy                           0.86       118
   macro avg       0.86      0.85      0.85       118
weighted avg       0.86      0.86      0.86       118
```

The test error of the QDA model obtained is around 100*(1-0.8559), i.e.,14.41%.

(f)

(f)

In [91]:
```
glm_train = sm.GLM(y_train,
                   X_train,
                   family=sm.families.Binomial())
results = glm_train.fit()
```

In [92]:
```
#confusion matrix for test data
probs = results.predict(exog=X_test)
labels = np.array([0]*118)
labels[probs >0.5] = 1
confusion_table(labels, y_test)
```

Out[92]:

Truth	0	1
Predicted		
0	44	7
1	10	57

In [93]:
```
print("overall fraction of correct predictions using logistic regression=",(44+57)/(44+10+6+58))
```

```
overall fraction of correct predictions using logistic regression= 0.8559322033898306
```

In [94]:
```
result = metrics.classification_report(y_test, labels)
print("Classification Report:",result)
```

```
Classification Report:               precision    recall  f1-score   support

           0       0.86      0.81      0.84        54
           1       0.85      0.89      0.87        64

    accuracy                           0.86       118
   macro avg       0.86      0.85      0.85       118
weighted avg       0.86      0.86      0.86       118
```

The test error of the logistic regression model obtained is around 100*(1-0.8559), i.e.,14.41%.

(h)

(h)

In [95]:
```
#scaling data as kNN is distance based

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(X_train_scaled.shape)
print(X_test_scaled.shape)
```

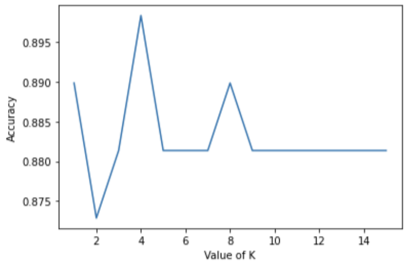
```
(274, 4)
(118, 4)
```

In [102]:
```
range_k = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
test_errors=[]
scores_list = []
for k in range_k:
    classifier = KNeighborsClassifier(n_neighbors=k)
    classifier.fit(X_train_scaled, y_train)
    y_pred = classifier.predict(X_test_scaled)
    scores_list.append(metrics.accuracy_score(y_test,y_pred))
    print('test error for k=',k,'is',100*(1-metrics.accuracy_score(y_test,y_pred)))
print (scores_list)

%matplotlib inline
#import matplotlib.pyplot as plt
plt.plot(range_k,scores_list)
plt.xlabel("Value of K")
plt.ylabel("Accuracy")
```

```
test error for k= 1 is 11.016949152542377
test error for k= 2 is 12.711864406779661
test error for k= 3 is 11.864406779661019
test error for k= 4 is 10.169491525423723
test error for k= 5 is 11.864406779661019
test error for k= 6 is 11.864406779661019
test error for k= 7 is 11.864406779661019
test error for k= 8 is 11.016949152542377
test error for k= 9 is 11.864406779661019
test error for k= 10 is 11.864406779661019
test error for k= 11 is 11.864406779661019
test error for k= 12 is 11.864406779661019
test error for k= 13 is 11.864406779661019
test error for k= 14 is 11.864406779661019
test error for k= 15 is 11.864406779661019
```

```
test error for k= 15 is 11.864406779661019
[0.8898305084745762, 0.8728813559322034, 0.8813559322033898, 0.8983050847457628, 0.8813559322033898, 0.8813559322033898, 0.8813559322033898, 0.8898305084745762, 0.8813559322033898, 0.8813559322033898, 0.8813559322033898, 0.8813559322033898, 0.8813559322033898, 0.8813559322033898, 0.8813559322033898]
```

Out[102]: Text(0, 0.5, 'Accuracy')



In [100]:

```
#k=4 seems to give the best result

classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train_scaled, y_train)
y_pred4 = classifier.predict(X_test_scaled)

result = metrics.confusion_matrix(y_test, y_pred4)
print("Confusion Matrix:")
print(result)

result1 = metrics.classification_report(y_test, y_pred4)
print("Classification Report:",result1)
```

```
Confusion Matrix:
[[47  7]
 [ 5 59]]
Classification Report:               precision    recall  f1-score   support

           0       0.90      0.87      0.89        54
           1       0.89      0.92      0.91        64

    accuracy                           0.90       118
   macro avg       0.90      0.90      0.90       118
weighted avg       0.90      0.90      0.90       118
```

In [101]: `confusion_table(y_pred4, y_test)`

Out[101]:

Truth	0	1
Predicted		
0	47	5
1	7	59

K=4 seems to perform best among the values of k tried on this dataset.

Section 5.4

Conceptual

1. (5.6) states:

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

where, $\sigma^2_X = Var(X)$, $\sigma^2_Y = Var(Y)$, and $\sigma_{XY} = Cov(X,Y)$

Here, α is computed by minimizing $Var\{\alpha X + (1-\alpha)Y\}$

$\cancel{\sigma^2 = \frac{\Sigma(X-\mu)^2}{N}}$ ~~[Formula for variance]~~

~~We can obtain the~~

To get the α which corresponds to ~~the minimum~~ the ~~minimal~~ minimal value of $Var[\alpha X + (1-\alpha)Y]$

$$\frac{\partial}{\partial\alpha}\left[Var\{\alpha X + (1-\alpha)Y\}\right] = 0 \quad (1)$$

[~~Set differential~~ equate differential of term to be minimized as zero with respect to α]

$$\sigma^2(aX + bY) = a^2\sigma^2(X) + b^2\sigma^2(Y) + 2ab\,\sigma_{XY}$$

$$\therefore Var\{\alpha X + (1-\alpha)Y\} = \alpha^2 Var(X) + (1-\alpha)^2 Var(Y) + 2\alpha(1-\alpha)Cov(X,Y)$$

Substituting the above in 1:

$$\cancel{\frac{\partial}{\partial\alpha}\left[Var(\alpha X)\ \frac{\partial}{\partial\alpha}[(1-\alpha)^2 Var(Y)]\right] = 2Var(Y)\frac{\partial}{\partial\alpha}(1-\alpha)}$$

$$\frac{\partial}{\partial\alpha}\left[\alpha^2 Var(X) + (1-\alpha)^2 Var(Y) + 2\alpha Cov(X,Y) - 2\alpha^2 Cov(X,Y)\right] = 0$$

$\because$ $Var(X)$, $Var(Y)$ and $Cov(X,Y)$ are constants with respect to α

$$\therefore 2\alpha Var(X) - 2(1-\alpha)Var(Y) + 2Cov(X,Y) - 4\alpha Cov(X,Y) = 0$$

$$\cancel{\partial\alpha\left[2Var(X) - 2\alpha Var(X) + 2Var(Y)\right]}$$

$$\therefore 2\alpha Var(X) - 2Var(Y) + 2\alpha Var(Y) + 2Cov(X,Y) - 4\alpha Cov(X,Y) = 0$$

$$\boxed{\frac{\partial}{\partial\alpha}\left[(1-\alpha)^2 Var(Y)\right] = 2(1-\alpha)Var(Y)(-1) = Var(Y)\left[2(\alpha-1)\right]}$$

Isha Jain

$$\therefore \alpha\left[2Var(X)+2Var(Y)-4Cov(X,Y)\right] - 2Var(Y) + 2Cov(X,Y) = 0$$

$$\therefore \alpha\left[4Cov(X,Y) - 2Var(Y) - 2Var(X)\right] = 2Cov(X,Y) - 2Var(Y)$$

On dividing throughout by 2 and simplifying

$$\cancel{\alpha = \frac{Cov(X,Y) - Var(Y)}{Var(X)+Var(Y)\ 2Cov(X,Y)}}$$

$$\alpha\left[Var(X)+Var(Y)-2Cov(X,Y)\right] = Var(Y) - Cov(X,Y)$$

$$\therefore \alpha = \frac{Var(Y) - Cov(X,Y)}{Var(X)+Var(Y)-2Cov(X,Y)}$$

$$\therefore \alpha = \frac{\sigma^2_Y - \sigma_{XY}}{\sigma^2_X+\sigma^2_Y-2\sigma_{XY}}$$

Hence proved.