CRYPTOCURRENCY PRICE MOVEMENT PREDICTIONS

Importing the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from math import exp
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

Importing the dataset

In [2]:
```
Crypto=pd.read_csv('Crypto.csv')
```

Exploratory Data Analysis

1) Checking the shape of the data

In [3]:
```
Crypto.shape
```

Out[3]:
```
(17199, 19)
```

2) Preview of the how the dataset is

In [4]:
```
Crypto.head()
```

Out[4]:

	Date	CoinName	Open	High	Low	Close	Adj Close	Volume	High-Low	rt-1	rt-2	rt-3	rt-4	rt-5	MA(Last 5 days)	log returns (last 3)	log returns (last 5)	last 5-last 3	Up/Down
0	09-11-2017	ADA	0.025160	0.035060	0.025006	0.032053	0.032053	18716200.0	0.010054	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	1
1	10-11-2017	ADA	0.032219	0.033348	0.026451	0.027119	0.027119	6766780.0	0.006897	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	-1
2	11-11-2017	ADA	0.026891	0.029659	0.025684	0.027437	0.027437	5532220.0	0.003975	-0.167156	0.000000	0.000000	0.0	0.0	0.0	-0.167156	-0.167156	0.0	1
3	12-11-2017	ADA	0.027480	0.027952	0.022591	0.023977	0.023977	7280250.0	0.005361	0.011658	-0.167156	0.000000	0.0	0.0	0.0	-0.155498	-0.155498	0.0	-1
4	13-11-2017	ADA	0.024364	0.026300	0.023495	0.025808	0.025808	4419440.0	0.002805	-0.134797	0.011658	-0.167156	0.0	0.0	0.0	-0.290296	-0.290296	0.0	1

3) Checking for missing values

In [5]:
```
Crypto.isna().sum()
```

Out[5]:
```
Date                     0
CoinName                 0
Open                     0
High                     0
Low                      0
Close                    0
Adj Close                0
Volume                   0
High-Low                 0
rt-1                     0
rt-2                     0
rt-3                     0
rt-4                     0
rt-5                     0
MA(Last 5 days)          0
log returns (last 3)     0
log returns (last 5 )    0
last 5- last 3           0
Up/Down                  0
dtype: int64
```

4) Checking the data types of the columns

In [6]:
```
Crypto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17199 entries, 0 to 17198
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   17199 non-null  object
 1   CoinName               17199 non-null  object
 2   Open                   17199 non-null  float64
 3   High                   17199 non-null  float64
 4   Low                    17199 non-null  float64
 5   Close                  17199 non-null  float64
 6   Adj Close              17199 non-null  float64
 7   Volume                 17199 non-null  float64
 8   High-Low               17199 non-null  float64
 9   rt-1                   17199 non-null  float64
 10  rt-2                   17199 non-null  float64
 11  rt-3                   17199 non-null  float64
 12  rt-4                   17199 non-null  float64
 13  rt-5                   17199 non-null  float64
 14  MA(Last 5 days)        17199 non-null  float64
 15  log returns (last 3)   17199 non-null  float64
 16  log returns (last 5 )  17199 non-null  float64
 17  last 5- last 3         17199 non-null  float64
 18  Up/Down                17199 non-null  int64
dtypes: float64(16), int64(1), object(2)
memory usage: 2.5+ MB
```

5) Removing the columns that are not neccessary for prediction

In [8]:
```
Crypto.drop(['Adj Close','Date','CoinName'],axis=1,inplace=True)
```

6) Renaming the columns

In [9]:
```
Crypto.columns=['Open', 'High', 'Low', 'Close', 'Volume',
       'HighLow', 'rt_1', 'rt_2', 'rt_3', 'rt_4', 'rt_5', 'MA',
```

```
        'lr3', 'lr5', 'lr5_lr3',
        'Movement']
```

7) Looking at the dataframe after cleaning the data

In [11]:
```
Crypto.head()
```

Out[11]:

	Open	High	Low	Close	Volume	HighLow	rt_1	rt_2	rt_3	rt_4	rt_5	MA	lr3	lr5	lr5_lr3	Movement
0	0.025160	0.035060	0.025006	0.032053	18716200.0	0.010054	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	1
1	0.032219	0.033348	0.026451	0.027119	6766780.0	0.006897	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	-1
2	0.026891	0.029659	0.025684	0.027437	5532220.0	0.003975	-0.167156	0.000000	0.000000	0.0	0.0	0.0	-0.167156	-0.167156	0.0	1
3	0.027480	0.027952	0.022591	0.023977	7280250.0	0.005361	0.011658	-0.167156	0.000000	0.0	0.0	0.0	-0.155498	-0.155498	0.0	-1
4	0.024364	0.026300	0.023495	0.025808	4419440.0	0.002805	-0.134797	0.011658	-0.167156	0.0	0.0	0.0	-0.290296	-0.290296	0.0	1

8) Checking whether the data is balanced

In [12]:
```
Crypto.Movement.value_counts()
```

Out[12]:
```
 1    8705
-1    8494
Name: Movement, dtype: int64
```

Now we have cleaned the data successfully,so we can move on to building the model

PREREQUISITES

1) Separating the Movement Column from the features

In [16]:
```
x = Crypto.drop('Movement', axis = 1)
y = Crypto['Movement']
```

Looking at the features and movement separately

In [17]:
```
x.head()
```

Out[17]:

	Open	High	Low	Close	Volume	HighLow	rt_1	rt_2	rt_3	rt_4	rt_5	MA	lr3	lr5	lr5_lr3
0	0.025160	0.035060	0.025006	0.032053	18716200.0	0.010054	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0
1	0.032219	0.033348	0.026451	0.027119	6766780.0	0.006897	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0
2	0.026891	0.029659	0.025684	0.027437	5532220.0	0.003975	-0.167156	0.000000	0.000000	0.0	0.0	0.0	-0.167156	-0.167156	0.0
3	0.027480	0.027952	0.022591	0.023977	7280250.0	0.005361	0.011658	-0.167156	0.000000	0.0	0.0	0.0	-0.155498	-0.155498	0.0
4	0.024364	0.026300	0.023495	0.025808	4419440.0	0.002805	-0.134797	0.011658	-0.167156	0.0	0.0	0.0	-0.290296	-0.290296	0.0

In [18]:
```
y.head()
```

Out[18]:
```
0    1
1   -1
2    1
3   -1
4    1
Name: Movement, dtype: int64
```

2) Splitting the data into training and test set

In [19]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

3) Scaling the data

In [20]:
```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

Converting it back into a dataframe`

In [21]:
```
cols=x.columns
x_train = pd.DataFrame(x_train, columns=cols)
x_test = pd.DataFrame(x_test, columns=cols)
```

Now we are done with the prerequistes, now we can build different models

LOGISTIC REGRESSION MODEL

1) Importing the neccessary libraries

In [22]:
```
from sklearn.linear_model import LogisticRegression
```

2) Fitting the Logistic Regression algorithm into the training set

In [23]:
```
LRmodel = LogisticRegression()
LRmodel.fit(x_train, y_train)
```

Out[23]:
```
LogisticRegression()
```

3) Model Evaluation

In [24]:
```
x_test_prediction = LRmodel.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction, y_test)
print('Accuracy score on Test Data : ', test_data_accuracy)
confusion_matrix(y_test, x_test_prediction )
```

```
Accuracy score on Test Data :  0.5897674418604651
```

Out[24]:
```
array([[ 998, 1166],
       [ 598, 1538]], dtype=int64)
```

RANDOM FOREST CLASSIFIER

1) Importing libraries and fitting the Random forest classifier model into the training set

In [25]:
```
from sklearn.ensemble import RandomForestClassifier
rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)
rfc_100.fit(x_train,y_train)
```

Out[25]:
```
RandomForestClassifier(random_state=0)
```

2) Model Evaluation

In [26]:
```
y_pred_100 = rfc_100.predict(x_test)
print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))
confusion_matrix(y_test, y_pred_100 )
```

```
Model accuracy score with 100 decision-trees : 0.5777
```

Out[26]:
```
array([[1237,  927],
       [ 889, 1247]], dtype=int64)
```

3) Finding the significant features

In [27]:
```
clf = RandomForestClassifier(n_estimators=100, random_state=0)
clf.fit(x_train, y_train)
feature_scores = pd.Series(clf.feature_importances_, index= x_train.columns).sort_values(ascending=False)
feature_scores
```

Out[27]:
```
rt_1       0.085097
rt_2       0.077634
rt_3       0.076474
rt_4       0.076274
Volume     0.076075
lr3        0.075333
rt_5       0.074048
lr5_lr3    0.072861
lr5        0.071731
HighLow    0.061600
Close      0.054096
Open       0.051956
MA         0.051281
Low        0.047808
High       0.047733
dtype: float64
```

Here we can see all the features are significant

K-NEAREST NEIGHOURS

1) Importing libraries and fitting the K-nearest neighours algorithm into the training set

In [25]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

Out[25]:
```
KNeighborsClassifier()
```

Finding the optimal k for the given model

In [26]:
```
from sklearn import neighbors
from sklearn.metrics import f1_score
f1_list=[]
k_list=[]
for k in range(1,15):
    clf=neighbors.KNeighborsClassifier(n_neighbors=k,n_jobs=-1)
    clf.fit(x_train,y_train)
    pred=clf.predict(x_test)
    f=f1_score(y_test,pred,average='macro')
    f1_list.append(f)
    k_list.append(k)
best_f1_score=max(f1_list)
```

```
best_k=k_list[f1_list.index(best_f1_score)]
print("Optimum K value=",best_k)
```

```
Optimum K value= 11
```

Building the model with the optimal k

In [27]:
```
classifier= KNeighborsClassifier(n_neighbors=11, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
print("Accuracy score=",classifier.score(x_test,y_test))
confusion_matrix(y_test, y_pred)
```

```
Accuracy score= 0.5502325581395349
```

Out[27]:
```
array([[1162, 1002],
       [ 932, 1204]], dtype=int64)
```

DECISION TREES

1) Importing libraries and fitting the decision tree classifier algorithm into the training set

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

Out[28]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

2) Model Evaluation

In [29]:
```
y_pred= classifier.predict(x_test)
print("Accuracy score of the model=",accuracy_score(y_test, y_pred))
confusion_matrix(y_test, y_pred)
```

```
Accuracy score of the model= 0.5225581395348837
```

Out[29]:
```
array([[1134, 1030],
       [1023, 1113]], dtype=int64)
```

Here 1134+1113=2247 predictions are correct and 1023+1030=2053 predictions are incorrect

BERNOULLI NAIVE BAYES

1) Checking for Multicollinearity`

In [30]:
```
Crypto.corr()
```

Out[30]:

	Open	High	Low	Close	Volume	HighLow	rt_1	rt_2	rt_3	rt_4	rt_5	MA	lr3	lr5	lr5_lr3	Movement
Open	1.000000	0.999679	0.999401	0.999197	0.720594	0.836400	-0.001110	-0.001239	-0.001076	-0.001042	-0.000844	0.999062	-0.001963	-0.002276	-0.001342	-0.001949
High	0.999679	1.000000	0.999342	0.999642	0.723121	0.842318	-0.001374	-0.001315	-0.001143	-0.001059	-0.000769	0.998840	-0.002196	-0.002426	-0.001301	0.001952
Low	0.999401	0.999342	1.000000	0.999584	0.715292	0.822220	-0.000668	-0.000845	-0.000906	-0.000765	-0.000773	0.998305	-0.001386	-0.001696	-0.001095	0.002674
Close	0.999197	0.999642	0.999584	1.000000	0.719679	0.833099	-0.001257	-0.001107	-0.000967	-0.000861	-0.000697	0.998266	-0.001909	-0.002095	-0.001109	0.006144
Volume	0.720594	0.723121	0.715292	0.719679	1.000000	0.718396	0.014413	0.013095	0.011762	0.012650	0.016486	0.721582	0.022503	0.029315	0.020739	0.010049
HighLow	0.836400	0.842318	0.822220	0.833099	0.718396	1.000000	-0.011637	-0.008085	-0.004476	-0.005247	-0.000584	0.839525	-0.013866	-0.012869	-0.004150	-0.009103
rt_1	-0.001110	-0.001374	-0.000668	-0.001257	0.014413	-0.011637	1.000000	-0.013433	0.049638	0.046742	0.033440	-0.006850	0.593828	0.478503	0.057074	-0.071049
rt_2	-0.001239	-0.001315	-0.000845	-0.001107	0.013095	-0.008085	-0.013433	1.000000	-0.013477	0.049696	0.046769	-0.005388	0.557601	0.458384	0.068665	0.015296
rt_3	-0.001076	-0.001143	-0.000906	-0.000967	0.011762	-0.004476	0.049638	-0.013477	1.000000	-0.013252	0.049643	-0.003922	0.593694	0.459639	0.025902	0.013311
rt_4	-0.001042	-0.001059	-0.000765	-0.000861	0.012650	-0.005247	0.046742	0.049696	-0.013252	1.000000	-0.013148	-0.002570	0.047671	0.458501	0.702461	-0.001767
rt_5	-0.000844	-0.000769	-0.000773	-0.000697	0.016486	-0.000584	0.033440	0.046769	0.049643	-0.013148	1.000000	-0.001053	0.074408	0.478477	0.702424	-0.011241
MA	0.999062	0.998840	0.998305	0.998266	0.721582	0.839525	-0.006850	-0.005388	-0.003922	-0.002570	-0.001053	1.000000	-0.009260	-0.008479	-0.002579	-0.001414
lr3	-0.001963	-0.002196	-0.001386	-0.001909	0.022503	-0.013866	0.593828	0.557601	0.593694	0.047671	0.074408	-0.009260	1.000000	0.800246	0.086896	-0.024325
lr5	-0.002276	-0.002426	-0.001696	-0.002095	0.029315	-0.012869	0.478503	0.458384	0.459639	0.458501	0.478477	-0.008479	0.800246	1.000000	0.666942	-0.023767
lr5_lr3	-0.001342	-0.001301	-0.001095	-0.001109	0.020739	-0.004150	0.057074	0.068665	0.025902	0.702461	0.702424	-0.002579	0.086896	0.666942	1.000000	-0.009259
Movement	-0.001949	0.001952	0.002674	0.006144	0.010049	-0.009103	-0.071049	0.015296	0.013311	-0.001767	-0.011241	-0.001414	-0.024325	-0.023767	-0.009259	1.000000

Here the features High, Low, Close, Volume, HighLow are highly correlated with Open and rt_4,rt_5 and lr5 are highly correlated with lr5_lr3. This reduces the acccuracy of the model.Hence we drop all the feautures except one of them from each

2) Dropping the highly correlated features

In [31]:

```
Crypto.drop(['High', 'Low', 'Close', 'HighLow','Open','MA','rt_4','rt_5','lr5'],axis=1, inplace=True)
```

Now Checking again for correlation

In [32]:

```
Crypto.corr()
```

Out[32]:

	Volume	rt_1	rt_2	rt_3	lr3	lr5_lr3	Movement
Volume	1.000000	0.014413	0.013095	0.011762	0.022503	0.020739	0.010049
rt_1	0.014413	1.000000	-0.013433	0.049638	0.593828	0.057074	-0.071049
rt_2	0.013095	-0.013433	1.000000	-0.013477	0.557601	0.068665	0.015296
rt_3	0.011762	0.049638	-0.013477	1.000000	0.593694	0.025902	0.013311
lr3	0.022503	0.593828	0.557601	0.593694	1.000000	0.086896	-0.024325
lr5_lr3	0.020739	0.057074	0.068665	0.025902	0.086896	1.000000	-0.009259
Movement	0.010049	-0.071049	0.015296	0.013311	-0.024325	-0.009259	1.000000

Now we can see that the features of the model are less correlated

3) Importing the neccessary libraries

```
from sklearn.datasets import make_classification
```

```
In [33]: from sklearn.datasets import make_classification
from sklearn.naive_bayes import BernoulliNB
```

4) Separating and splitting the new dataframe

```
In [34]: X = Crypto.drop('Movement', axis = 1)
Y = Crypto['Movement']
```

5) Building and fitting the bernoulli naive bayes algorithm

```
In [35]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20)
bnb = BernoulliNB(binarize=0.0)
bnb.fit(X_train, Y_train)
```

```
Out[35]: BernoulliNB()
```

6) Model Accuracy

```
In [36]: Y_pred= bnb.predict(X_test)
print('Accuracy of the model =' ,bnb.score(X_test, Y_test))
confusion_matrix(Y_test, Y_pred)
```

```
Accuracy of the model = 0.5287790697674418
Out[36]: array([[895, 814],
       [807, 924]], dtype=int64)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js