Stock Trade Signal – KWANG HEUM YEON

April 26, 2022

1 Stock price - Decision Trees and Random Forests

1.1 Getting Started

I am crazy about predicting stock prices. To figure out the best way to minimize the loss in our choice, analyzing the movement of the giants such as investment banks, fund managers, and tycoons are very required. A stock price can be manipulated but not trade volume. That is, volume can be compared to a log file containing all the historical facts. Again, only if we can go back in time, we can possibly change the amount of the past trade volume. For these reasons, I gathered the price and volume for a day and calculated subsets from those two variables in order to find insight from the dataset. Here is my belief:

> Trade Signal insight = Volume Speed + Volume Speed Acceleration + Volume Change per Price Change

1.2 Dataset

This data comes from our Github repository. We gathered the real-time stock price with its volume by using API-yfinance and then get subsets. US stock market operates for 390 minutes from 9 30 AM to 4 00 PM, the data contains the values checked approximately three times every single minute (every 20 seconds). The price gathered here is an adjusted close price which applied an equity change, stock split as an example. A more detailed introduction of variables follows under the preview of the dataset.

- Data source: (1) yfinance (2) Student's Github

(1) https://pypi.org/project/yfinance/

(2) https://raw.githubusercontent.com/cpasean/Projects/main/GOOGL%202022-04-25.csv

[5]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt

# Pretty display for notebooks
%matplotlib inline

import warnings   # ignore warnings
warnings.simplefilter(action='ignore', category=FutureWarning) # filter warning
```

```
warnings.filterwarnings("ignore")  # runtime warning

# Load the dataset
df = pd.read_csv('https://raw.githubusercontent.com/cpasean/Projects/main/
 ↪GOOGL%202022-04-25.csv', index_col = 0).reset_index(drop = True)
df.head()
```

[5]:

	Date	Minutepassed	Adj Close	Volume	VolumeSpeed	\
0	2022-04-22	-2.766667	2392.709961	2841400.0	-1.033236e+06	
1	2022-04-22	-2.750000	2392.709961	2841400.0	-1.033236e+06	
2	2022-04-22	-2.416667	2392.709961	2841400.0	-1.175752e+06	
3	2022-04-22	-2.166667	2392.709961	2841400.0	-1.311415e+06	
4	2022-04-22	-1.916667	2392.709961	2841400.0	-1.482470e+06	

	VolumeSpeed_vs_mean	Volume_change	Price_change	\
0	-227.360900	0.000001	0.000001	
1	-228.747247	0.000001	0.000001	
2	-258.721024	0.000001	0.000001	
3	-288.573450	0.000001	0.000001	
4	-329.074987	0.000001	0.000001	

	Volume_speed_acceleration	VolumeC_per_PriceC
0	0.000000	1.0
1	0.000000	1.0
2	-427546.081505	1.0
3	-542654.641910	1.0
4	-684216.722408	1.0

Introduction of each variable: - **Date**: date

- **Minutepassed**: minute elapsed
- **Adj Close**: adjusted stock price
- **Volume**: current volume
- **VolumeSpeed**: current volume divided by minutepassed
- **VolumeSpeed_vs_mean**: current volume speed divided by 100 days average volume speed at the moment
- **Volume_change**: difference with the volume in the previous line
- **Price_change**: difference with the price in the previous line
- **Volume_speed_acceleration**: difference with the volume speed change in the previous line
- **VolumeC_per_PriceC**: volume change divided by price change

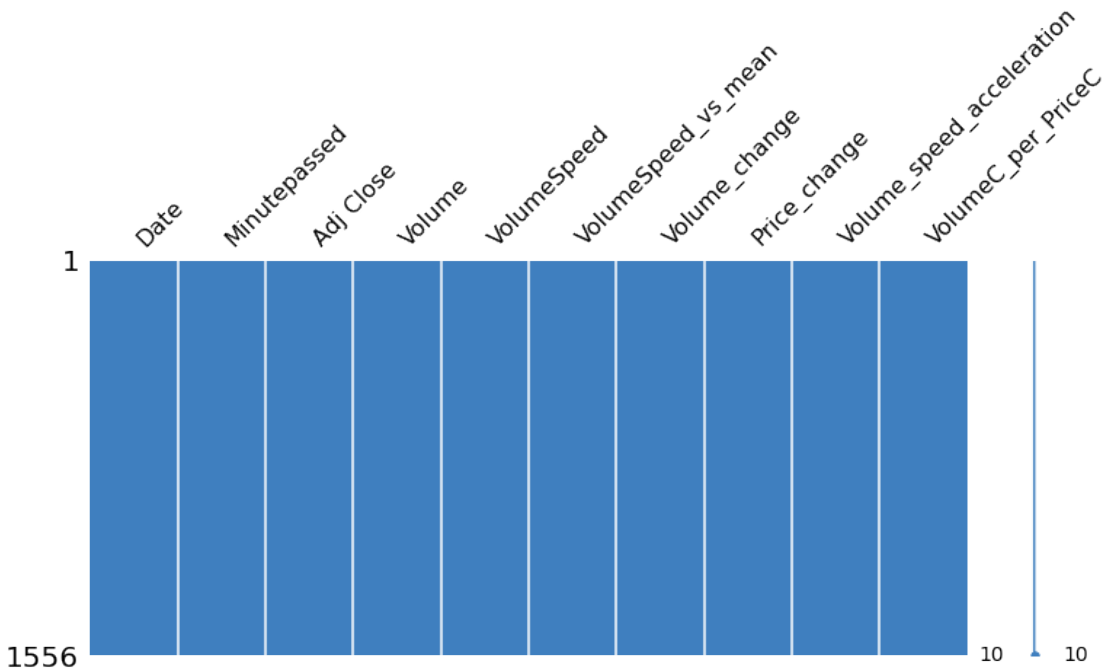
1.3 Data Cleansing

[2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1556 entries, 0 to 1555
Data columns (total 10 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        1556 non-null   object
 1   Minutepassed                1556 non-null   float64
 2   Adj Close                   1556 non-null   float64
 3   Volume                      1556 non-null   float64
 4   VolumeSpeed                 1556 non-null   float64
 5   VolumeSpeed_vs_mean         1556 non-null   float64
 6   Volume_change               1556 non-null   float64
 7   Price_change                1556 non-null   float64
 8   Volume_speed_acceleration   1556 non-null   float64
 9   VolumeC_per_PriceC          1556 non-null   float64
dtypes: float64(9), object(1)
memory usage: 121.7+ KB
```

[3]:
```
# check null values

import missingno as msno
msno.matrix(df, figsize = (12,5), color=(0.25, 0.5, 0.75));

print(f"Number of Null values: {df.isna().sum().sum()}")
```

```
Number of Null values: 0
```



[4]:
```
# check duplicated values

print(f"Number of duplicated values: {df.duplicated().sum()}")
```

```
Number of duplicated values: 0
```

[5]:
```
# check column names

df.columns
```

[5]:
```
Index(['Date', 'Minutepassed', 'Adj Close', 'Volume', 'VolumeSpeed',
       'VolumeSpeed_vs_mean', 'Volume_change', 'Price_change',
       'Volume_speed_acceleration', 'VolumeC_per_PriceC'],
      dtype='object')
```

[6]:
```
# filter out rows before the market open

df = df.query('1 <= Minutepassed <= 390')
df.tail(2)
```

[6]:
```
            Date  Minutepassed    Adj Close     Volume  VolumeSpeed  \
1552  2022-04-25    389.633333  2462.489990  2072369.0  5318.767217
1553  2022-04-25    389.900000  2461.939941  2081351.0  5338.166196

      VolumeSpeed_vs_mean  Volume_change  Price_change  \
```

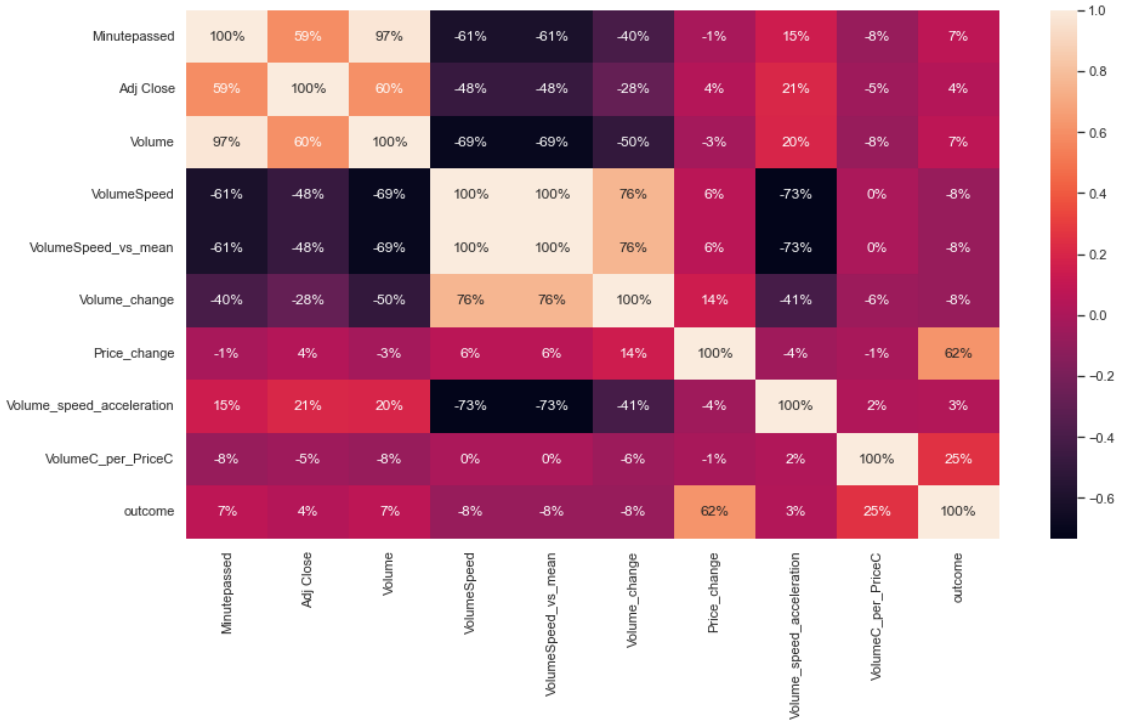
```
1552             1.170769       0.378240      0.131747
1553             1.174899       0.433417     -0.022337

      Volume_speed_acceleration  VolumeC_per_PriceC
1552                  65.049607            2.870958
1553                  72.746173           19.403461
```

[7]:
```
# add a new column for the outcome
# if the Price change is equal to or bigger than zero, add '1' into the␣
 ↪'outcome' column or '0' for else case

df.loc[df['Price_change'] >= 0, 'outcome'] = 1
df.loc[df['Price_change'] < 0, 'outcome'] = 0
df.sample(3)
```

[7]:
```
            Date  Minutepassed    Adj Close     Volume   VolumeSpeed  \
824   2022-04-25    213.716667  2427.850098  1290932.0   6040.534976
1381  2022-04-25    346.700000  2438.310059  1651297.0   4762.898760
33    2022-04-25      5.600000  2410.280029   191326.0  34242.500000

      VolumeSpeed_vs_mean  Volume_change  Price_change  \
824              1.335325       0.061311     -0.026349
1381             1.050693       0.012416      0.000001
33               7.617128       2.275084     -0.044994

      Volume_speed_acceleration  VolumeC_per_PriceC  outcome
824                   -12.894449            2.326851      0.0
1381                  -11.380859        12416.025273      1.0
33                  -3160.078125           50.564570      0.0
```

1.4 Data Exploration

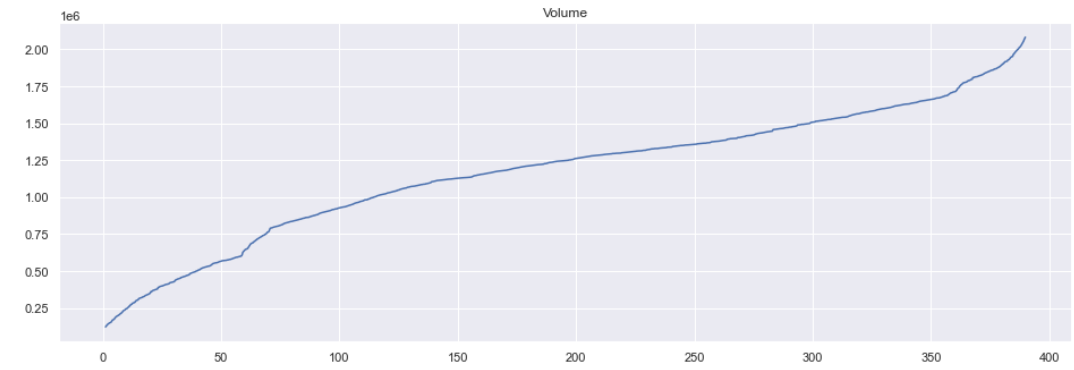
[8]:
```
# Making a correlation plot

sb.set(rc = {'figure.figsize':(15,8)})
sb.heatmap(df.corr(), annot=True, fmt ='.0%');
```

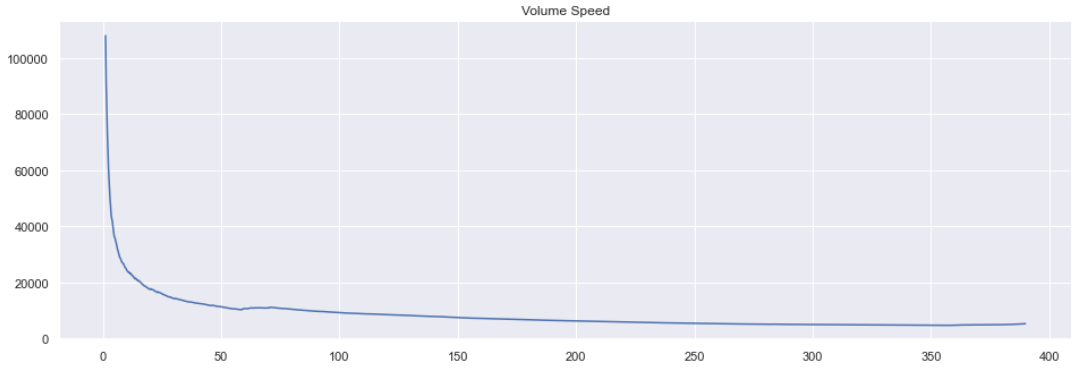


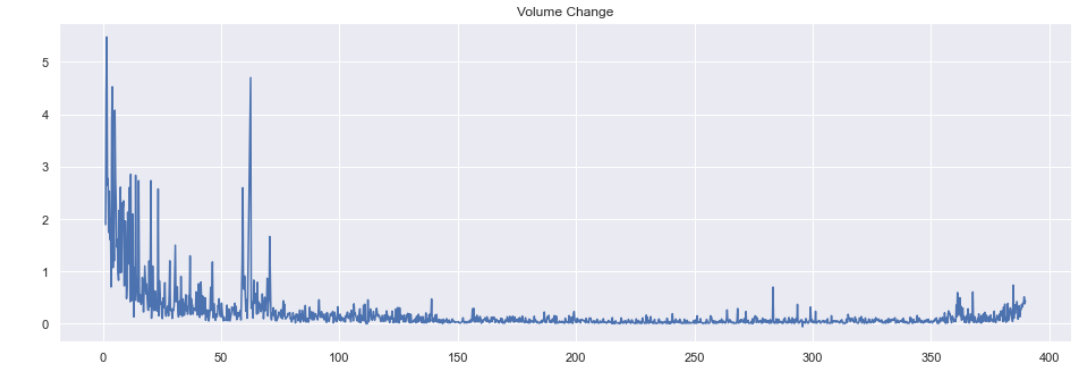
[9]:
```
# line plot shows each time series pattern

plt.figure(figsize = (16,30))

plt.subplot(5,1,1)
plt.errorbar(x = df['Minutepassed'], y = df['Price_change'])
plt.title('Price Change')

plt.subplot(5,1,2)
plt.errorbar(x = df['Minutepassed'], y = df['Volume'])
plt.title('Volume')

plt.subplot(5,1,3)
plt.errorbar(x = df['Minutepassed'], y = df['VolumeSpeed'])
plt.title('Volume Speed')

plt.subplot(5,1,4)
plt.errorbar(x = df['Minutepassed'], y = df['Volume_change'])
plt.title('Volume Change')

plt.subplot(5,1,5)
plt.errorbar(x = df['Minutepassed'], y = df['VolumeC_per_PriceC'])
plt.title('VolumeChange per PriceChange')
```

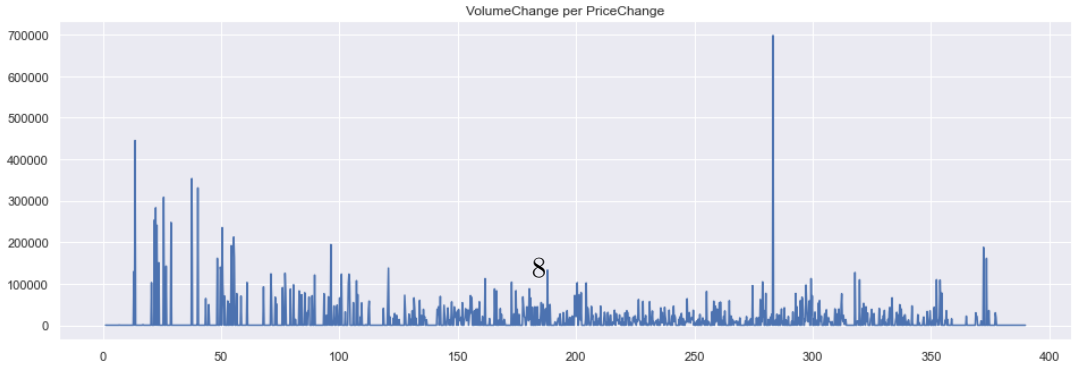
```
plt.show()
```


Price Change
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
50
100
150
200
250
300
350
400


Volume
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
50
100
150
200
250
300
350
400


Volume Speed
100000
80000
60000
40000
20000
0
0
50
100
150
200
250
300
350
400


Volume Change
5
4
3
2
1
0
0
50
100
150
200
250
300
350
400


VolumeChange per PriceChange
700000
600000
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350
400

1.5 Preprocessing the data

[10]:
```
# filter out unwanted columns

df = df[['Adj Close', 'Volume', 'VolumeSpeed',
       'VolumeSpeed_vs_mean', 'Volume_change',
       'Volume_speed_acceleration', 'VolumeC_per_PriceC',
       'outcome']]
df.head(2)
```

[10]:

	Adj Close	Volume	VolumeSpeed	VolumeSpeed_vs_mean	Volume_change	\
16	2381.379883	122356.0	107961.176471	24.082405	1.892009	
17	2386.685059	127710.0	91221.428571	20.299314	4.375756	

	Volume_speed_acceleration	VolumeC_per_PriceC	outcome
16	-114709.530543	150.180880	0.0
17	-66958.991597	19.641832	1.0

[11]:
```
# Remove 'Minutepassed' and 'outcome' from the dataset and store it into␣
↪features

outcomes = df.outcome
features = df.drop('outcome', axis = 1)
```

1.6 Training the model

[12]:
```
# Split data into training and testing sets

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, outcomes,␣
↪test_size=0.3, random_state=42)
```

[13]:
```
# # Feature Scaling

# from sklearn.preprocessing import StandardScaler

# sc = StandardScaler()
# X_train = sc.fit_transform(X_train)
# X_test = sc.transform(X_test)
```

Standardization was not effective, so we do not apply the process

1.7 Training the Decision Tree Classifier

[14]:
```
# Import the classifier from sklearn
from sklearn.tree import DecisionTreeClassifier

# Define the classifier, and fit it to the data
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

[14]:
```
DecisionTreeClassifier()
```

1.8 Perform 5 fold cross validation

[15]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model, X_train, y_train, cv = 5)
cv_scores
```

[15]:
```
array([0.68981481, 0.71627907, 0.73023256, 0.6744186 , 0.61860465])
```

[16]:
```
np.average(cv_scores)
```

[16]:
```
0.685869939707149
```

[17]:
```
# Getting the Parameters of the Decision Tree

model.get_params()
```

[17]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

1.9 Testing the model

[18]:
```
# Making predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

```
# Calculate the accuracy
from sklearn.metrics import accuracy_score
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print('The training accuracy is', train_accuracy)
print('The test accuracy is', test_accuracy)
```

```
The training accuracy is 1.0
The test accuracy is 0.6796536796536796
```

1.10 Generating the confusion matrix

[19]:
```
#Generating the confusion matrix using scikit-learn's confusion matrix method
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_test_pred)
```

[19]:
```
array([[ 74,  78],
       [ 70, 240]], dtype=int64)
```

[20]:
```
#Generating the confusion matrix as a dataframe.
pd.DataFrame(confusion_matrix(y_test, y_test_pred), columns = ['Actual Positive␣
↪(Up)', 'Acutal Negative (Down)'], index = [ 'Predicted Positive (Up)',␣
↪'Predicted Negative (Down)']          )
```

[20]:

	Actual Positive (Up)	Acutal Negative (Down)
Predicted Positive (Up)	74	78
Predicted Negative (Down)	70	240

1.11 Classification Report

[21]:
```
#Generate Classification Report
from sklearn.metrics import classification_report

print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.51      0.49      0.50       152
         1.0       0.75      0.77      0.76       310

    accuracy                           0.68       462
   macro avg       0.63      0.63      0.63       462
weighted avg       0.68      0.68      0.68       462
```

1.12 Training the Random Forest Classifier

[22]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10, criterion='gini',random_state=1)
rf.fit(X_train, y_train.values.ravel())
```

[22]:
```
RandomForestClassifier(n_estimators=10, random_state=1)
```

1.13 Perform 5 fold cross validation

[23]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(rf, X_train, y_train, cv = 5)
cv_scores
```

[23]:
```
array([0.66203704, 0.70697674, 0.72093023, 0.70697674, 0.6372093 ])
```

[24]:
```
np.average(cv_scores)
```

[24]:
```
0.6868260120585703
```

1.14 Testing the model

[25]:
```
y_train_pred_rf = rf.predict(X_train)
y_test_pred_rf = rf.predict(X_test)

# Calculate the accuracy
from sklearn.metrics import accuracy_score
train_accuracy = accuracy_score(y_train, y_train_pred_rf)
test_accuracy = accuracy_score(y_test, y_test_pred_rf)
print('The training accuracy is', train_accuracy)
print('The test accuracy is', test_accuracy)
```

```
The training accuracy is 0.9897769516728625
The test accuracy is 0.683982683982684
```

1.15 Generating the confusion matrix

[26]:
```
confusion_matrix(y_test, y_test_pred_rf)
```

[26]:
```
array([[ 84,  68],
       [ 78, 232]], dtype=int64)
```

[27]:
```
#Generating the confusion matrix as a dataframe.
pd.DataFrame(confusion_matrix(y_test, y_test_pred_rf), columns = ['Actual␣
↪Positive (Up)', 'Acutal Negative (Down)'], index = [ 'Predicted Positive␣
↪(Up)', 'Predicted Negative (Down)']          )
```

[27]:

	Actual Positive (Up)	Acutal Negative (Down)
Predicted Positive (Up)	84	68
Predicted Negative (Down)	78	232

1.16 Classification Report

[28]:
```
#Generate Classification Report
from sklearn.metrics import classification_report

print(classification_report(y_test, y_test_pred_rf))
```

```
              precision    recall  f1-score   support

         0.0       0.52      0.55      0.54       152
         1.0       0.77      0.75      0.76       310

    accuracy                           0.68       462
   macro avg       0.65      0.65      0.65       462
weighted avg       0.69      0.68      0.69       462
```

1.17 Converting the prediction to a dataframe

[29]:
```
#converting the prediction to a dataframe

y_pred_df = pd.DataFrame(y_test_pred_rf, columns = ['predicted price up or␣
 ↪down'])
```

[30]:
```
#dataframe of predicted values
y_pred_df.head(2)
```

[30]:

	predicted price up or down
0	0.0
1	1.0

[31]:
```
# creating a merged dataframe of test data and the predictions

df = pd.concat([X_test.reset_index(drop=True), y_test.reset_index(drop=True),␣
 ↪y_pred_df.reset_index(drop=True)], axis=1)
df.head(2)
```

[31]:

	Adj Close	Volume	VolumeSpeed	VolumeSpeed_vs_mean	Volume_change	\
0	2419.520020	1057835.0	8326.863440	1.843177	0.033003	
1	2411.980957	363349.0	17508.028846	3.890776	0.499526	

	Volume_speed_acceleration	VolumeC_per_PriceC	outcome	\
0	-49.510040	2.159204	0.0	
1	-341.412596	2.770468	1.0	

```
   predicted price up or down
0                         0.0
1                         1.0
```

[32]:
```
# We can now compare our predictions with the actual data

df.head()
```

[32]:
```
     Adj Close     Volume   VolumeSpeed  VolumeSpeed_vs_mean  Volume_change  \
0  2419.520020  1057835.0   8326.863440             1.843177       0.033003
1  2411.980957   363349.0  17508.028846             3.890776       0.499526
2  2452.000000  1905986.0   5006.310905             1.102854       0.317376
3  2429.429932  1203611.0   6799.421900             1.503829       0.029503
4  2440.010010  1695290.0   4742.069930             1.045950       0.242492

   Volume_speed_acceleration  VolumeC_per_PriceC  outcome  \
0                 -49.510040            2.159204      0.0
1                -341.412596            2.770468      1.0
2                  50.237487            3.738058      1.0
3                 -30.937323            0.491241      0.0
4                  34.975298            2.035762      0.0

   predicted price up or down
0                         0.0
1                         1.0
2                         1.0
3                         0.0
4                         1.0
```

1.18 Reference

- cross value score by sklearn:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html