ST1 Assignment 9 (Capstone Programming Project) By William Blucher and Kim Grove

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True
```

```
%cd /content/drive/MyDrive/Colab_Project
```

```
/content/drive/MyDrive/Colab_Project
```

```
!ls
```

```
DataForML.pkl  final_LinearRegression.pkl  NFLX.csv
```

This project is based on Netflix stock prices for the past five years. The data used for this project is on the Kaggle repository. (https://www.kaggle.com/datasets/jainilcoder/netflix-stock-price-prediction/data)

- This data set contains (xyz) amount of data
- Our project is to develop a machine learning model that can predict the next stock price based on the previous days data.
- For solving this problem we (... fill wih process)

Step 1: Reading the data Reading the data with python

```
import warnings
warnings.filterwarnings("ignore")
```

```
import pandas as pd
import numpy as np

# This is simply used to access and shape the data so that it is easier for us to use later on.
NFLXData = pd.read_csv("/content/drive/MyDrive/Colab_Project/NFLX.csv", encoding='latin')
print("Shape before deleting duplicates:", NFLXData.shape)

# We remove any duplicates that may be within the data to ensure that no errors will occur later on.
NFLXData=NFLXData.drop_duplicates()
print("Shape after deleting duplicate views:", NFLXData.shape)

# Next we check that the data is reflected the way we want.
NFLXData.head(10)
```

```
Shape before deleting duplicates: (1009, 7)
Shape after deleting duplicate views: (1009, 7)
```

	Date	Open	High	Low	Close	Adj Close	Vc
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	1189
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	1259
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	898
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	930
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	1690
5	2018-02-12	252.139999	259.149994	249.000000	257.950012	257.950012	853
6	2018-02-13	257.290009	261.410004	254.699997	258.269989	258.269989	685

Next steps: Generate code with NFLXData View recommended plots

A) Key Observations Regarding Data

data ⋮ •••

```
#Step 23:
Save the
movel as a
serialised
file that
can be
stored and
accessed
```

There is a total of 1,009 rows, representing days and 7 categories of data for each day. The following are the descriptions of these categories:

- Date - Date stock open and closed (explain why each is continuous or not)
- Open - Stock price at market open
- High - high point of stock price
- Low - low point of stock price
- Close - stock price at market close
- Adj Close - adjusted close price
- Volume - amount of available stocks

Step 2 : Problem Statement Definition

- Creating a prediction model to predict the price(Close) of the stocks.
- Target Variable: Close Predictors/Features: Open, High, Low, Adj Close and Volume.

Step 3: Choosing the appropriate ML/AI Algorithm for Data Analysis.

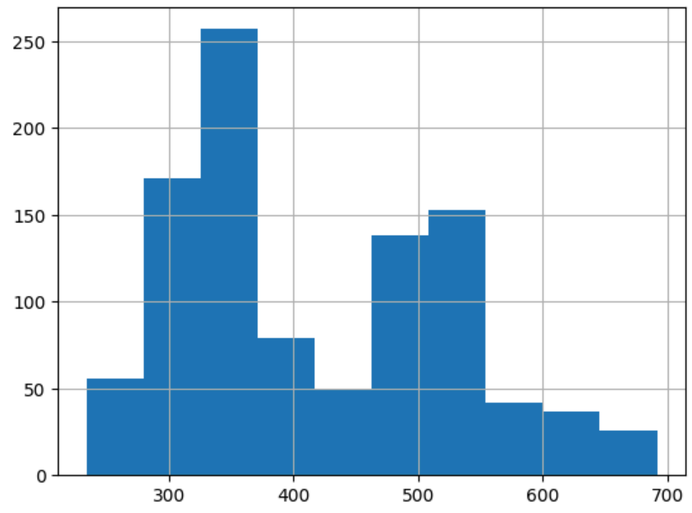
- Considering the problem statement, we need to create a supervised ML Regression model, as the target variable (and the other variables) are **continuous**. This hypothesis is further supported by the results further down.

Step 4: Looking at the class distribution (Target variable distribution to check if the data is balanced or skewed.)

- This is used to check if the data is balanced or skewed (if the data is skewed too heavily, the accuracy of the ML algorithm may be negatively impacted.)
- A bell curve would be the ideal result.
- The bell curve is ideally required to ensure that the Regression Model will work with the highest accuracy value, however slightly skewed data should still work effectively.
- If the data is too skewed the machine learning algorithm will struggle when trying to learn all possible scenarios for the data.

```
%matplotlib inline
# This is used as the data we have chosen (close) is continuous.
# The results of this, as stated above, will help us estimate the accuracy of our ML algorithm.
NFLXData["Close"].hist()
```

```
<Axes: >
```



Observations from Step 4

So based on this, the data distribution is fairly skewed. However, noting that this data set is based around stocks, and stocks are typically a volatile data set that is subject to influence from outside effectors (such as politics or standing of the company) this is to be expected and we can proceed regardless.

Step 5: Basic Exploratory Data analysis

```
# From this, we look at a sample of the rows of data.
NFLXData.head()
```

	Date	Open	High	Low	Close	Adj Close	Vc
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	1189
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	1259
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	898
	2018-						

Next steps: Generate code with NFLXData View recommended plots

```
# Using tail and head (previous block) we are able to get a 'snapshot' of what the entire sample is like.
NFLXData.tail()
```

	Date	Open	High	Low	Close	Adj Close	
1004	2022-01-31	401.970001	427.700012	398.200012	427.140015	427.140015	2
1005	2022-02-01	432.959991	458.480011	425.540009	457.130005	457.130005	2
1006	2022-02-02	448.250000	451.980011	426.480011	429.480011	429.480011	1
	2022-						

```
# This step allows us to inspect.
# Using this we are able to identify the Data Type, the columns, and the number of empty cells.
# In this instance there is no empty cells.
# Additionally we can view what columns need to be removed (Qualitative data will not work for ML algorithms and thus needs to be r
NFLXData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1009 non-null   object
 1   Open       1009 non-null   float64
 2   High       1009 non-null   float64
 3   Low        1009 non-null   float64
 4   Close      1009 non-null   float64
 5   Adj Close  1009 non-null   float64
 6   Volume     1009 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 55.3+ KB
```

```
# Remove the data that was identified as qualitative in the previous step.
NFLXData = NFLXData.drop("Date", axis="columns")
```

```
NFLXData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       1009 non-null   float64
 1   High       1009 non-null   float64
 2   Low        1009 non-null   float64
 3   Close      1009 non-null   float64
 4   Adj Close  1009 non-null   float64
 5   Volume     1009 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 47.4 KB
```

```
# This step is another way to gain an understanding of the data within the set.
NFLXData.describe(include='all')
```

	Open	High	Low	Close	Adj Close	
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1.
mean	419.059673	425.320703	412.374044	419.000733	419.000733	7.
std	108.537532	109.262960	107.555867	108.289999	108.289999	5.
min	233.919998	250.649994	231.229996	233.880005	233.880005	1.
25%	331.489990	336.299988	326.000000	331.619995	331.619995	4.
50%	377.769989	383.010010	370.880005	378.670013	378.670013	5.
75%	509.130005	515.630005	502.529999	509.079987	509.079987	9.
max	692.349976	700.989990	686.090027	691.690002	691.690002	5.

```
# This allows us to identify how many duplicates appear amongst each respective row.
# Any columns that are roughly 20 or less may be categories, not contiuous.
NFLXData.nunique()
```

```
Open          976
High          983
Low           989
Close         988
Adj Close     988
Volume       1005
dtype: int64
```

All the data values are > 20, therefore all the values are continuous and not categorical.

Observations from Step 5 - Basic Data Exploratory Analysis

- based on the exploration above we discovered the following
- The Open column is continuos - Selected for now
- The Adj Close column is continuous - Selected for now (Will be reviewd later as results are identical to close so far.
- High - Continuous and Selected
- Low - Continuous and Selected
- Close - Continuous, selected as target variable
- Volume - Contiuous, selected for now.

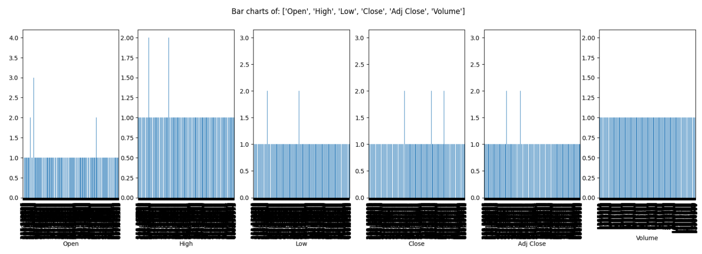
Step 7: Removal of bad columns

There are no bad columns left to remove as of this stage, so none are removed.

Step 8: Visual Exploratory Data Analysis

- Visualising the data with a bar plot
- As previously mentioned, without categorical data, this step will not produce correct bar charts, however for the sake of exhaustion it will be done anyway.

```
# This step is used to display any data that is categorical.
# For the sake of this data set, this step further confirms that none of the columns we currently have are categorical.
def PlotBarCharts(inpData, colsToPlot):
  %matplotlib inline

  import matplotlib.pyplot as plt

  fig, subPlot = plt.subplots(nrows=1, ncols=len(colsToPlot), figsize=(20,5))
  fig.suptitle("Bar charts of: " + str(colsToPlot))

  for colName, plotNumber in zip(colsToPlot, range(len(colsToPlot))):
    inpData.groupby(colName).size().plot(kind='bar', ax=subPlot[plotNumber])


PlotBarCharts(inpData=NFLXData, colsToPlot=['Open', 'High', 'Low', 'Close', 'Adj Close', "Volume"])
```



Observations

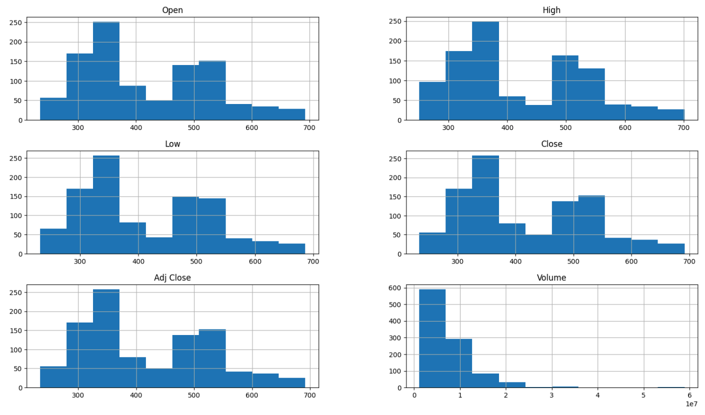
As predicted, bar charts are not usable as only works with categorical data, of which there is none.

Step 9: Visualising the distribution

Visualise the data with histograms.

```
# Displaying each of the data columns as histograms allows us to identify any outliers that may be present.
NFLXData.hist(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], figsize=(18,10))
```

```
array([[<Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>],
       [<Axes: title={'center': 'Adj Close'}>,
        <Axes: title={'center': 'Volume'}>]], dtype=object)
```



From this step, it can be observed that within 'Volume" there is an outlier beyond roughly 3.5x10^7.

As this column will not be used as a predictor later on in the project, this fact can be ignored.

No other outliers are observed at this time.

Step 10: Outlier Analysis

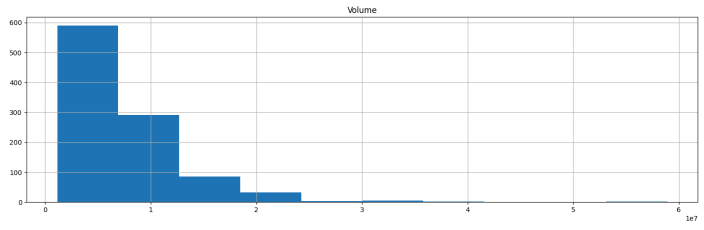
Outliers are as seen below for volume, as stated this is just to represent that we understand the process, however we will not be removing this data as it will not be used later in the models.

```
# This allows us to view the data, and identify where the outlier may appear.

NFLXData['Volume'][NFLXData['Volume']<(4*(10**7))].sort_values(ascending=True)
```

```
728      1144000
982      1287200
883      1367800
896      1595500
979      1621100
          ...
999     32346000
177     32610900
744     32637500
49      33866500
428     38258900
Name: Volume, Length: 1007, dtype: int64
```

```
# This allows us to single out the volume histogram and further view the outlier.

NFLXData.hist(['Volume'], figsize=(18,5))
```

```
    array([[<Axes: title={'center': 'Volume'}>]], dtype=object)
```



Step 12: (11 has been skipped as mentioned but number pattern has been followed for ease of reading) Missing Value Analysis

Check if any missing values have been introduced into the data.

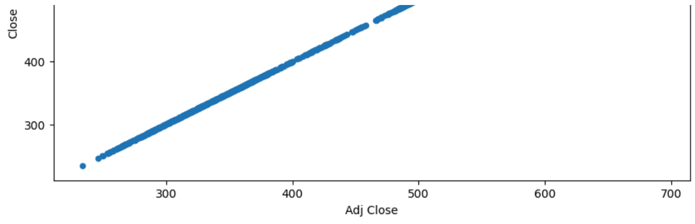
```
# Next, once again we double check to ensure that there are no null values that could skew the results later on.

NFLXData.isnull().sum()
```

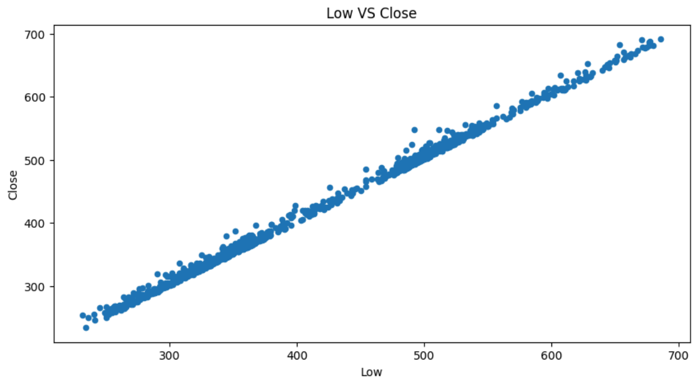
```
    Open         0
    High         0
    Low          0
    Close        0
    Adj Close    0
    Volume       0
    dtype: int64
```

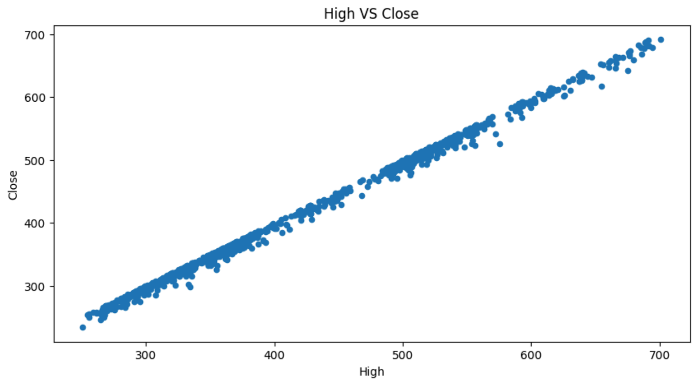
Step 13: Feature selection - Scatter Plot and Correlation matrix.

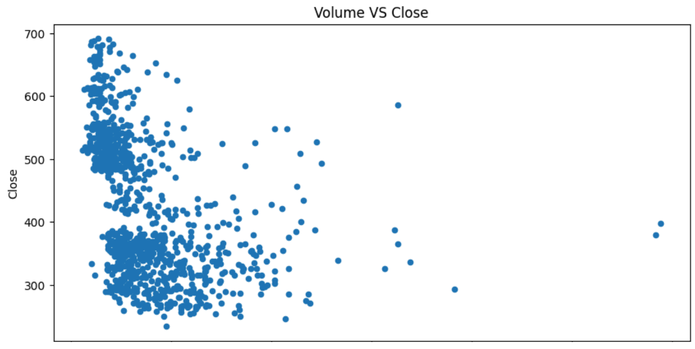
Checking the correlation of the data to identify the columns of interest.

```
# Using scatter grams, we plot the current predictor columns against our selected 'Close' column
# This is used to visualise any positive or negative trends within the data.

ContinuousCols=['Open', 'Adj Close', 'Low', 'High', 'Volume']

for predictor in ContinuousCols:
  NFLXData.plot.scatter(x=predictor, y="Close", figsize=(10,5), title=predictor+" VS "+ 'Close')
```


Close
400
300
300
400
500
600
700
Adj Close


Low VS Close
700
600
500
400
300
Close
300
400
500
600
700
Low


High VS Close
700
600
500
400
300
Close
300
400
500
600
700
High


Volume VS Close
700
600
500
400
300
Close

Scatter Charts interpretation

Upon reviewing the scatter plots, all data sets except volume have a strong positive correlation. Volume seems to have no identifiable correlation with our target variablel.

Finally, Adj Close ones again appears to be an exact copy of Close.

Step 14: Feature selection with Correlation Value

Using the corr() function to identify the correlations of the data.

```
# Again, we use a corrolation function to check the correlations of the data.
# Greater than 0.5, or less than -0.5 usually indicates a positive or negative correlation.

ContinuousCols=['Close', 'Open', 'Adj Close', 'Low', 'High', 'Volume']

CorrelationData=NFLXData[ContinuousCols].corr()
CorrelationData
```

```
           Close      Open  Adj Close       Low      High    Volume
Close   1.000000  0.996812   1.000000  0.998544  0.998551 -0.413362
Open    0.996812  1.000000   0.996812  0.998508  0.998605 -0.415838
 Adj
Close   1.000000  0.996812   1.000000  0.998544  0.998551 -0.413362
Low     0.998544  0.998508   0.998544  1.000000  0.998203 -0.432116
High    0.998551  0.998605   0.998551  0.998203  1.000000 -0.400699
Volume -0.413362 -0.415838  -0.413362 -0.432116 -0.400699  1.000000
```

Next steps: Generate code with CorrelationData View recommended plots

From the data above, we can see that open, low and high all have strong positive correlations with our focused 'Close'.

Additionally, we can identify that the 'ADJ Close' column has a correlation percentage of 100%, therefore meaning that it is, for this data set, functionally the same row. As such, it should be removed IOT not skew the accuracy of the ML model.

```
# This allows us to filter specifically for 'Close' and its correlations, specifically showing those that are a positive correlatic
CorrelationData['Close'][abs(CorrelationData['Close']) > 0.5]
```

```
    Close        1.000000
    Open         0.996812
    Adj Close    1.000000
    Low          0.998544
    High         0.998551
    Name: Close, dtype: float64
```

Again this further shows that the 'Adj Close' column is an 100% match with our selected 'Close' column, and should therefore be discarded.

Additionally, from this we can further deduce that the volume column does not have a strong enough correlation, whether positive or negative, to be considered from this data set.

Selecting Final Predictors/Features for building Machine Learning/AI model.

Step 15 and 16 are skipped as no categorical data.

```
# As we do not have categorical data, we are able to simply select the columns that we have good correlation.

SelectedColumns = ['Open', 'Low', 'High']

DataForML = NFLXData[SelectedColumns]
DataForML.head()
```

	Open	Low	High
0	262.000000	250.029999	267.899994
1	247.699997	245.000000	266.700012
2	266.579987	264.329987	272.450012
3	267.079987	250.000000	267.619995
4	253.850006	236.110001	255.800003

Next steps: Generate code with DataForML View recommended plots

```
# This saves the final data subset so that we can reference it later.

DataForML.to_pickle('DataForML.pkl')
```

Step 17: Data Pre-processing for Machine Learning Model Building or Model Development

Converting the nominal variable to numeric using get_dummies()

```
# This treats all the nominal variables using dummy variables.
DataForML_Numeric = pd.get_dummies(DataForML)

# Include the target variable in the data.
DataForML_Numeric['Close'] = NFLXData['Close']

# Print a sample of the first five rows
DataForML_Numeric.head()
```

	Open	Low	High	Close
0	262.000000	250.029999	267.899994	254.259995
1	247.699997	245.000000	266.700012	265.720001
2	266.579987	264.329987	272.450012	264.559998
3	267.079987	250.000000	267.619995	250.100006
4	253.850006	236.110001	255.800003	249.470001

Next steps: Generate code with DataForML_Numeric View recommended plots

Step 18: Machine Learning Model Development

Splitting the data into training and testing sample

```
DataForML_Numeric.columns
```

```
Index(['Open', 'Low', 'High', 'Close'], dtype='object')
```

```
# Seperates the Target variable from the Predictor variables.
TargetVariable = "Close"
Predictors =  ['Open', 'Low', 'High']

X = DataForML_Numeric[Predictors].values
y = DataForML_Numeric[TargetVariable].values

# Divide the data sets into training and testing data sets.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=428)
```

Step 18: Standardisation/Normalisation of data

```
# Standardisation of data
# This is stated as optional, however for the sake of exhaustion it is needed for KNN and nueral networks.
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Chose MinMaxScaler over StandardScaler as the data doesnt have varying scales.
PredictorScaler = MinMaxScaler()

# Stored for use later.
PredictorScalerFit = PredictorScaler.fit(X)

# Generating the standardised values of X
X = PredictorScalerFit.transform(X)

# Once again splitting the data into test and train.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# This step shows us how much data is in each set.
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(706, 3)
(706,)
(303, 3)
(303,)
```

Step 20: Multiple Linear Regression Algorithm

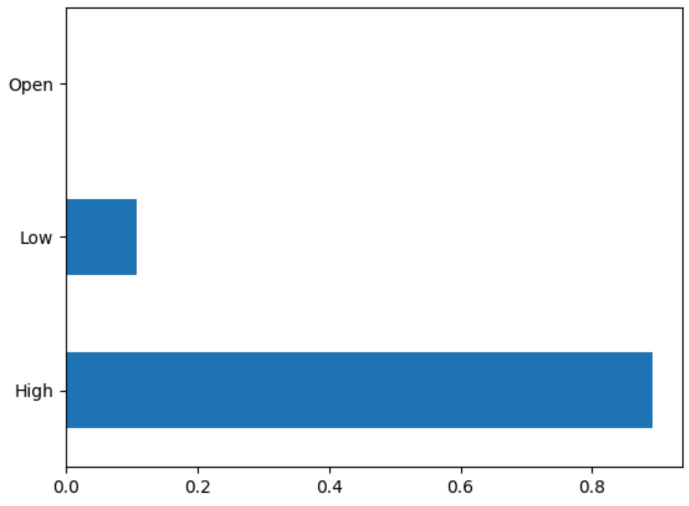
```
# Multiple Linear Regression
# Import
from sklearn.linear_model import LinearRegression
RegModel = LinearRegression()

# Print the parameters of the linear regression.
print(RegModel)

# Create the model using the training data.
LREG=RegModel.fit(X_train,y_train)
prediction=LREG.predict(X_test)

from sklearn import metrics
# Measurement of how well the data fits with the test.
print('R2 Value:',metrics.r2_score(y_train, LREG.predict(X_train)))

#Print the title of the output
print('\n########## Model Validation and Accuracy Calculations ##########')

# Creating the sample data values that will be printed
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing five of the predicted values
print(TestingDataResults.head())

# Calculates the error for each of the rows
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Close']-TestingDataResults['PredictedClose']))/TestingDataResults['Close'])


MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # If negative, indicative of an outlier present in the data.
print('Median Accuracy on test data:', MedianAccuracy)

# Custom function to ensure accuracy.
# Ensures there are no zeros in data when using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    return(100-MAPE)

# Using the make_scorer of sklearn.metrics to have a custom MAPE scoring calculation.
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Getting crossvalidation from sklearn
from sklearn.model_selection import cross_val_score

# Running a 10-fold cross validation to produce more accuracy values for comparison.
# Finally passes the data of x and y as K-fold splits the data and authomatically chooses train/test.
# This step prints the main values we care about.
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
LinearRegression()
R2 Value: 0.9986731542285744

########## Model Validation and Accuracy Calculations ##########
       Open       Low      High       Close  PredictedClose
0  0.577471  0.587917  0.576875  509.640015           508.0
1  0.592304  0.561601  0.572101  494.730011           493.0
2  0.596449  0.589962  0.585891  500.859985           507.0
3  0.330279  0.322451  0.303482  380.070007           381.0
4  0.195188  0.182474  0.166985  315.100006           319.0
Mean Accuracy on test data: 99.3029073385015
Median Accuracy on test data: 99.47124720149543

Accuracy values for 10-fold Cross Validation:
 [99.21860151 99.21910542 99.19755296 99.4083654  99.47040237 98.9092798
 99.20776329 99.40785386 99.62149833 99.22984505]

Final Average Accuracy of the model: 99.29
```

Decision Tree Regressor

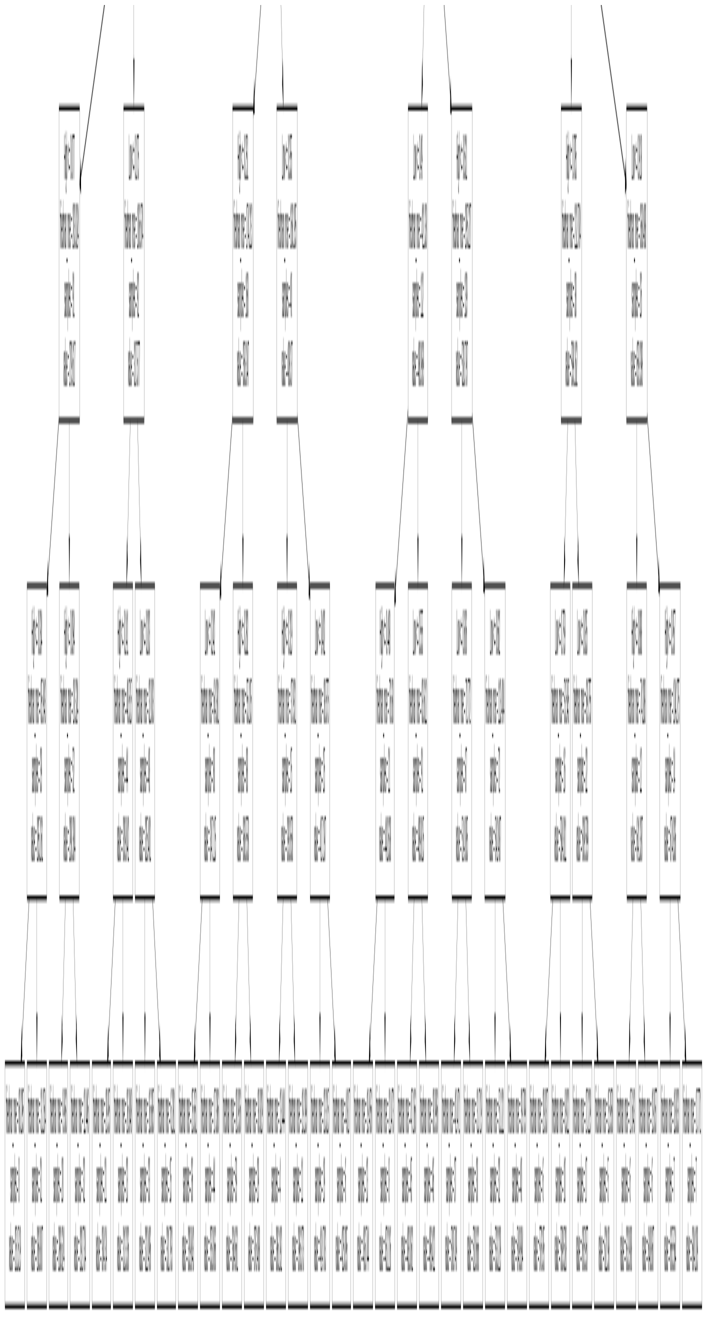
```
# Decision Trees (Bulk Else-If statements)
from sklearn.tree import DecisionTreeRegressor
RegModel = DecisionTreeRegressor(max_depth=5,criterion='friedman_mse')
# A good range of Max_depth is roughly between 2 - 20

# Printing all parameters of the decision tree.
print(RegModel)

# Create the mdodel using the training data.
DT=RegModel.fit(X_train,y_train)
prediction=DT.predict(X_test)

from sklearn import metrics
# Measuring how well the data fits.
print('R2 Value:',metrics.r2_score(y_train, DT.predict(X_train)))

# Using Matplotlib, plot the 10 largest importance pieces.
%matplotlib inline
feature_importances = pd.Series(DT.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

# Print statement model title.
print('\n##### Model Validation and Accuracy Calculations ##########')

# Setting up the variables to be printed.
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing the sample of predicted values.
print(TestingDataResults.head())

# Calculating the individual error for each row.
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Close']-TestingDataResults['PredictedClose']))/TestingDataResults['Close'])

# Calculating mean and median MAPE
MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

# Calculating accuracy as a percentage.
Accuracy =100 - MAPE
MedianAccuracy=100 - MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # If negative, outlier present in data set.
print('Median Accuracy on test data:', MedianAccuracy)

# Custom function to ensure accuracy.
# Also ensure no zeros are present that may skew data.
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    return(100-MAPE)

# Using sklearn.metrics's make_scorer to have a custom MAPE scoring calculation.
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Getting the crossvalidation function from sklearn
from sklearn.model_selection import cross_val_score

# Printing the 10-fold cross validation so to give further values to compare.
# Finally, pass the data of the x and y as K-fold splits the data and authomatically decides test or train.
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=5)
R2 Value: 0.9976613969339876

##### Model Validation and Accuracy Calculations ##########
       Open       Low      High       Close  PredictedClose
0  0.577471  0.587917  0.576875  509.640015           509.0
1  0.592304  0.561601  0.572101  494.730011           496.0
2  0.596449  0.589962  0.585891  500.859985           509.0
3  0.330279  0.322451  0.303482  380.070007           386.0
4  0.195188  0.182474  0.166985  315.100006           322.0
Mean Accuracy on test data: 98.56869716791199
Median Accuracy on test data: 98.8354057430968

Accuracy values for 10-fold Cross Validation:
 [98.40322063 98.50561057 98.48597259 98.84648187 98.33407053 97.24234762
 97.95457369 98.81150986 98.98729668 95.69681323]

Final Average Accuracy of the model: 98.13
```



Visualising the Decision tree

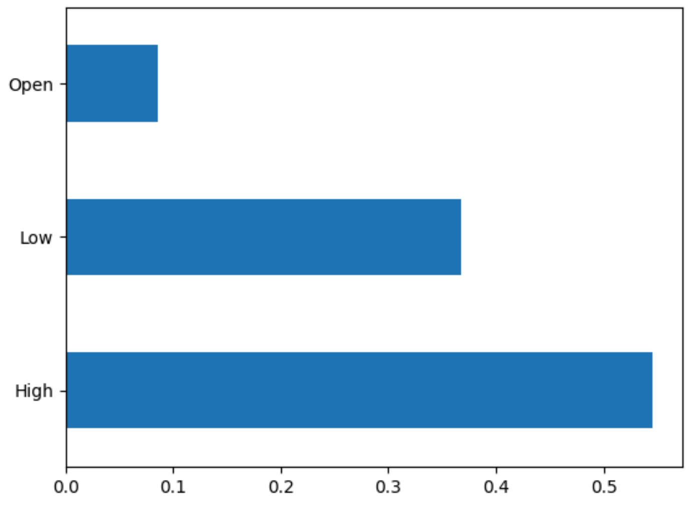
```
# Load libraries used
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT (graph description language) data
dot_data = tree.export_graphviz(RegModel, out_file=None,
                                feature_names=Predictors, class_names=TargetVariable)

# printing the rules
#print(dot_data)

# Draws the graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Loads the graph as an image
Image(graph.create_png(), width=2000,height=2000)
```



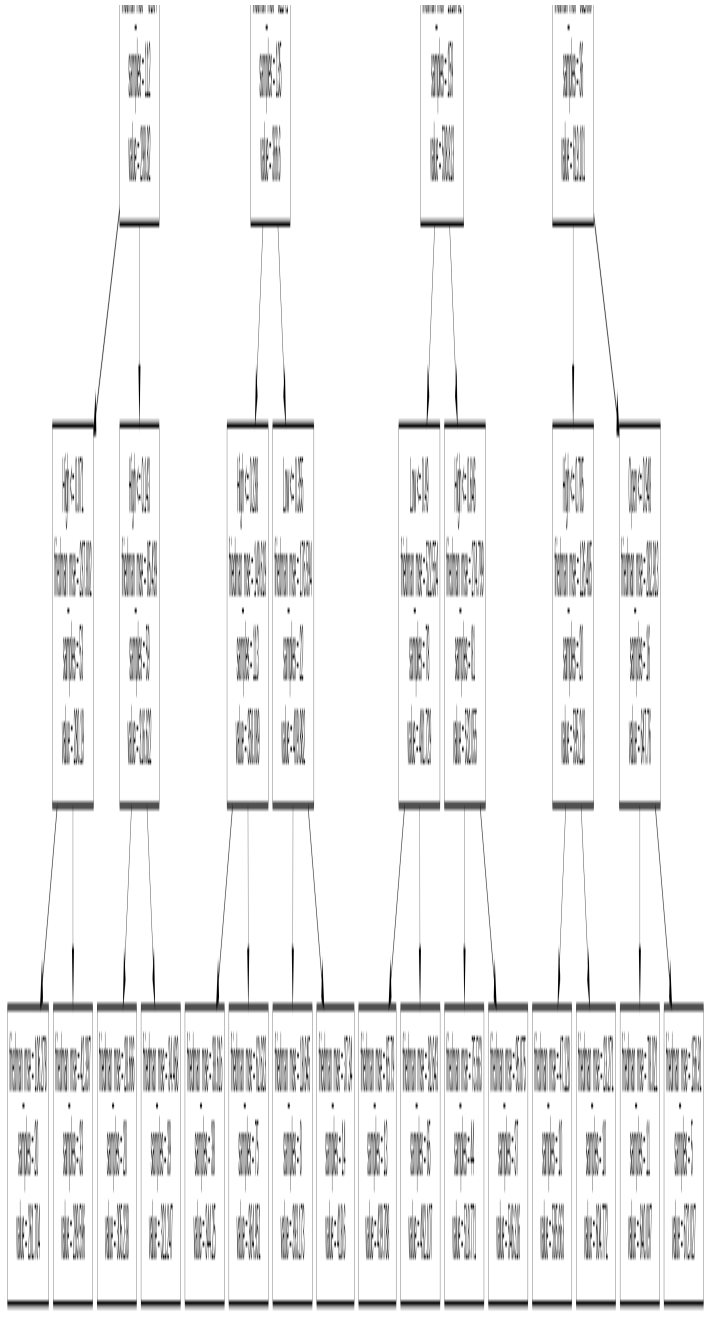
Random Forest Regressor

```
# Random Forest
from sklearn.ensemble import RandomForestRegressor
RegModel = RandomForestRegressor(max_depth=4, n_estimators=400,criterion='friedman_mse')
# A good range for the max_depth is generatlly 2-10 and the should be between 100-1000

# Printing all the parameters
print(RegModel)

# Create the model using the training data
RF=RegModel.fit(X_train,y_train)
prediction=RF.predict(X_test)

from sklearn import metrics
# Measurment of how well the data fits within the model.
print('R2 Value:',metrics.r2_score(y_train, RF.predict(X_train)))

# Plot of the 10 largest importance on a bar graph
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')


print('\n########## Model Validation and Accuracy Calculations ##########')

# Creating the prediction value variables
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing a sample of the prediction value variables
print(TestingDataResults.head())

# Calculating the APE percentage
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Close']-TestingDataResults['PredictedClose']))/TestingDataResults['Close'])

#Create the mean and median APE
MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

#Using the MAPE to calculate the accuracy percentage of the model
Accuracy = 100 - MAPE
MedianAccuracy= 100 - MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # This value results as negative if outliers are present.
print('Median Accuracy on test data:', MedianAccuracy)


# Making a custom function to calculate the accuracy score using the original and predicted.
# Make sure no zeros are present within.
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    return(100-MAPE)

# A custom score calculation using the make_scorer from sklearn
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Getting the cross validation value from sklearn
from sklearn.model_selection import cross_val_score

# Printing the 10-fold cross validation so to give further values to compare.
# Finally, pass the data of the x and y as K-fold splits the data and authomatically decides test or train.
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
RandomForestRegressor(criterion='friedman_mse', max_depth=4, n_estimators=4
R2 Value: 0.9971395316156282

########## Model Validation and Accuracy Calculations ##########
       Open       Low      High       Close  PredictedClose
0  0.577471  0.587917  0.576875  509.640015           500.0
1  0.592304  0.561601  0.572101  494.730011           496.0
2  0.596449  0.589962  0.585891  500.859985           502.0
3  0.330279  0.322451  0.303482  380.070007           384.0
4  0.195188  0.182474  0.166985  315.100006           322.0
Mean Accuracy on test data: 98.74477854666928
Median Accuracy on test data: 98.99831266406315

Accuracy values for 10-fold Cross Validation:
 [98.4554302  98.42310631 98.37079947 98.82139029 98.60895763 97.43986628
 97.72854863 98.56576993 99.02174248 94.64640591]

Final Average Accuracy of the model: 98.01
```



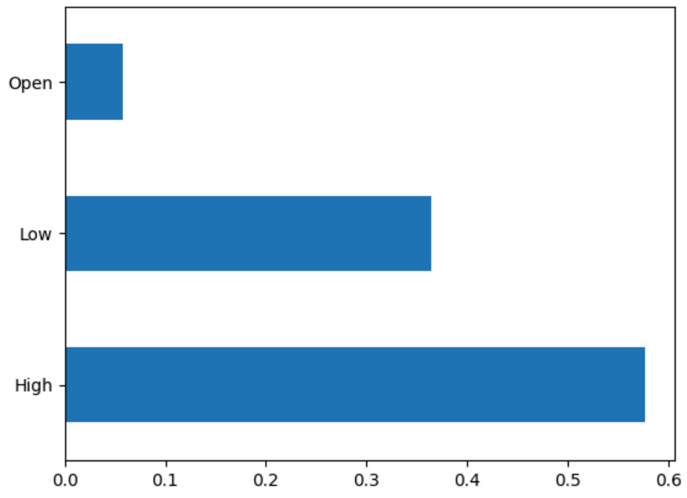
```
# Plotting the single Decision Tree from our Random Forest
# Load libraries used
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT (graph description language) data for the our 6th Decision Tree in Random Forest
dot_data = tree.export_graphviz(RegModel.estimators_[5] , out_file=None, feature_names=Predictors, class_names=TargetVariable)

# load the graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Create the png of the graph
Image(graph.create_png(), width=2000,height=2000)
```



Step 21: AdaBoost Algorithm

```
# Adaboost (Boost of multiple decision trees)
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

# The 6 level decision tree is chosen as the "weak learner"
DTR=DecisionTreeRegressor(max_depth=3)
RegModel = AdaBoostRegressor(n_estimators=100, base_estimator=DTR ,learning_rate=0.04)

# Printing the parameters of our Adaboost
print(RegModel)

# Creating the model using our training data
AB=RegModel.fit(X_train,y_train)
prediction=AB.predict(X_test)

from sklearn import metrics
# Measuring the quality of fit with training data.
print('R2 Value:',metrics.r2_score(y_train, AB.predict(X_train)))

# Using matplotlib to plot the 10 most highest importance columns.
%matplotlib inline
feature_importances = pd.Series(AB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')


print('\n########## Model Validation and Accuracy Calculations ##########')

# Creating the variable to store our predicted values.
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing a sample of the prediction values.
print(TestingDataResults.head())

# Calculation to calculate the error for each row.
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Close']-TestingDataResults['PredictedClose']))/TestingDataResults['Close'])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy = 100 - MAPE
MedianAccuracy= 100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Negative value indicative of outlier in some cases
print('Median Accuracy on test data:', MedianAccuracy)


# Making a custom function to calculate the accuracy score using the original and predicted.
# Make sure no zeros are present within.
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    return(100-MAPE)

# calculation using sklearn's make_scorer to make a custom scorer for accuracy.
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running a 10 fold cross validation of the data.
# Passing full data of X and y because the K-fold will split the data into test or train for us.
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

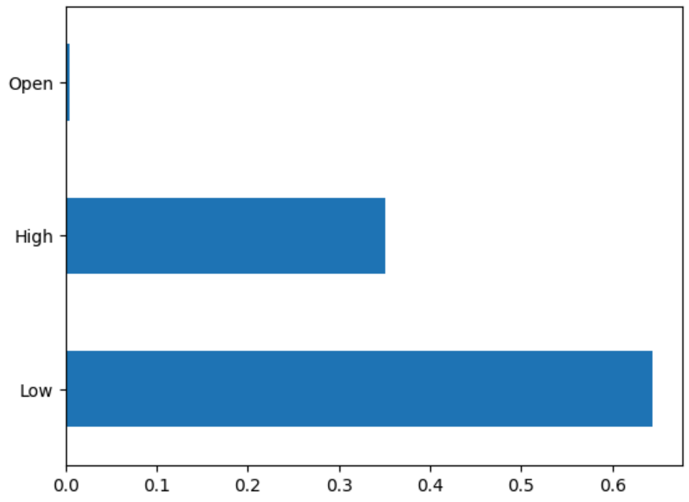
```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3),
                  learning_rate=0.04, n_estimators=100)
R2 Value: 0.9894519230021294

########## Model Validation and Accuracy Calculations ##########
       Open       Low      High       Close  PredictedClose
0  0.577471  0.587917  0.576875  509.640015           503.0
1  0.592304  0.561601  0.572101  494.730011           493.0
2  0.596449  0.589962  0.585891  500.859985           505.0
3  0.330279  0.322451  0.303482  380.070007           388.0
4  0.195188  0.182474  0.166985  315.100006           317.0
Mean Accuracy on test data: 97.74835457799438
Median Accuracy on test data: 98.04831337136687

Accuracy values for 10-fold Cross Validation:
 [97.26972335 97.39495947 96.75205006 97.17342265 96.75582601 94.63849664
 98.54092564 97.09591207 98.16570955 93.57097683]

Final Average Accuracy of the model: 96.74
```



XGBoost Regressor

```
# Xtreme Gradient Boosting (XGBoost)
from xgboost import XGBRegressor
RegModel=XGBRegressor(max_depth=2,
                      learning_rate=0.1,
                      n_estimators=1000,
                      objective='reg:linear',
                      booster='gbtree')

# Print all XGBoost parameters
print(RegModel)

# Creating the model using out training data.
XGB=RegModel.fit(X_train,y_train)
prediction=XGB.predict(X_test)

from sklearn import metrics
# Measuring how well our training data works within the model.
print('R2 Value:',metrics.r2_score(y_train, XGB.predict(X_train)))

# Plotting the ten columns with the highest feature importance value.
%matplotlib inline
feature_importances = pd.Series(XGB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

print('\n########## Model Validation and Accuracy Calculations ##########')

# Creating a variable to store the values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing a sample of the prediction values
print(TestingDataResults.head())

# Making a calulation of the error.
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Close']-TestingDataResults['PredictedClose']))/TestingDataResults['Close'])

# Calculating the mean and median APE using the previous result.
MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

#Calculating the accuracy as a percentage using the MAPE
Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # This can be negative if an outlier is present.
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None
             enable_categorical=False, eval_metric=None, feature_types=None
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=2, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, objective='reg:linear', ...)
R2 Value: 0.9995434001336165

########## Model Validation and Accuracy Calculations ##########
       Open       Low      High       Close  PredictedClose
0  0.577471  0.587917  0.576875  509.640015           508.0
1  0.592304  0.561601  0.572101  494.730011           493.0
2  0.596449  0.589962  0.585891  500.859985           510.0
3  0.330279  0.322451  0.303482  380.070007           386.0
4  0.195188  0.182474  0.166985  315.100006           314.0
Mean Accuracy on test data: 99.07659283944815
Median Accuracy on test data: 99.34123847167325

Accuracy values for 10-fold Cross Validation:
 [98.92753273 98.92258095 98.9963492  99.31086522 99.14092335 98.04559908
 98.86405587 99.12300896 99.47586872 95.58969741]

Final Average Accuracy of the model: 98.64
```



```
#Plotting a single Decision tree out of XGBoost
from xgboost import plot_tree
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20, 8))
plot_tree(XGB, num_trees=10, ax=ax)
```