SOLENT UNIVERSITY

BSc (Hons) [Year Three]

Academic Year 2022-2023

Nastaran Sharifi Sadr

Data Science (COM 624)

Predicting Brain Stroke

Report

Step 1:

Problem definition: SOLFINTECH wants to implement an intelligence stock trader platform for more than 50 million of their subscribers. This platform helps users to understanding:

1- Anticipating stock/equities prices on a **daily/ a week or max of quarterly basis**.
2- Then customers can i**dentify stocks with the potential of buying low and selling high with specified interval.**
3- The user can ask the system, which stocks will give them specified amount of profit at a specified interval.

The system inputs are:

- time interval (a day/ a week/ max of quarterly basis)
- profit
- stock of interest

The system outputs are:

The system shows tomorrow's price for special ticker or crypto. Also, system say that tomorrows' price whether goes up or not. There is other extra information for the user that can help the user to know more about the stock that they wish to buy, for example, how many times the stock's price went up in the last 2 days or last 5 days.

value proposition: The system will help the users to make decisions to know which stock or a group of stocks will give them more profit if they buy them and, will help the user to know if the price goes up or down.

success metrics: The successful metric is having higher precision_score. Precision_score is s a metric that quantifies the number of correct positive predictions by model. The goal is having a success metrics more than %55.

Step 2:

The method of collecting data, data collection technique:

The data in this project, is gathered from **YAHOO FINANCE** website using Yahoo finance API. The data is a **real time** data in the range **of 25 years before the current day till the current day**. The data is collected from **51 cryptocurrencies and tickers.**

Image below shows list of names.

```
# A list for all Cryptos
name_list=['BTC-GBP','ETH-GBP','USDT-GBP','BNB-GBP','USDC-GBP','XRP-GBP',
            'DOGE-GBP','ADA-GBP','MATIC-GBP','DOT-GBP','DAI-GBP','LTC-GBP',
            'SOL-GBP','TRX-GBP','HEX-GBP','UNI7083-GBP','AVAX-GBP','LINK-GBP'
            ,'ATOM-GBP','XMR-GBP','CVNA','GME','RXDX','BYND','SAVA','AMC','MOMO','RENT','TSLA','AI',
            'BILI','MYSZ','EXPR','NUWE','ASML','ATOS','ZIM','CRM','RIG','GSL','AVCT',
            'MULN','BTI','GOCO','MBLY','VRM','MTB','EVGO','LULU','LVS','AAPL']
```

These steps have been done for gathering all data:

1. Writing construct-download_url function to downloading data for each ticker/cryptos individually
2. Each ticker/crypto's name will be send to construct-download_url function and a separate csv file will be created for each one of them.
3. Then all the csv files will be saved in the current path of the application.

The construct-download_url function has been used to retrieving the data from yahoo finance. This function gets, thicker/crypto's name, time and interval as its arguments. As in yahoo finance API is needed to convert the time to seconds, it first converts the time. Then, it places all given data to yahoo finance API. Finally, it returns the downloaded data after calling the API.

```
def construct_download_url(
    ticker,
    period1,
    period2,
    interval='monthly'
):
    """
    :period1 & period2: 'yyyy-mm-dd'
    :interval: {daily; weekly, monthly}
    """
    def convert_to_seconds(period):
        datetime_value = datetime.strptime(period, '%Y-%m-%d')
        total_seconds = int(time.mktime(datetime_value.timetuple())) + 86400
        return total_seconds
    try:
        interval_reference = {'daily': '1d', 'weekly': '1wk', 'monthly': '1mo'}
        _interval = interval_reference.get(interval)
        if _interval is None:
            print('interval code is incorrect')
            return
        p1 = convert_to_seconds(period1)
        p2 = convert_to_seconds(period2)
        url = f'https://query1.finance.yahoo.com/v7/finance/download/{ticker}?period1={p1}&period2={p2}&interval={_interval}&filter=history'

        return url
    except Exception as e:
        print(e)
        return
```

Then the user will be asked to add a ticker or a cryptocurrency's name. The Read_CSV function reads the csv file with the same name that user added and saves that to my_data variable.

```
#getting user's choosen ticker or crypto
global file_name
file_name=input("enter your input")
print(file_name)
my_data=read_csv(f"C:\\university\\second semeaster\\machine learning\\ML-final-project\\{file_name}.csv",
 delimiter=',')
```

Here is the first 5 rows and the last five rows of the data set for special product that has been added by user(In this case is AAPL). Here is some information about each column:

- Date is the date of the current day
- Open is the price the stock started with that in the day
- High is the highest price of the day for the stock
- Low is the lowest price of the day
- Close is the closing price of the day
- Volume is the number of time that stock been shared

The important column that we can use to predict is close price. So, we will use the close price to predict the price.

```
1
my_data.tail(5)
```

	Date	Open	High	Low	Close	Adj Close	Volume
3152	2023-01-05	110.510002	111.750000	107.160004	110.339996	110.339996	157986300
3153	2023-01-06	103.000000	114.389999	101.809998	113.059998	113.059998	220575900
3154	2023-01-09	118.959999	123.519997	117.110001	119.769997	119.769997	190284000
3155	2023-01-10	121.070000	122.760002	114.919998	118.849998	118.849998	167356900
3156	2023-01-11	122.089996	125.940002	120.510002	123.199203	123.199203	73384688

5 rows × 7 columns Open in new tab

```
my_data.head(5)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	1.266667	1.666667	1.169333	1.592667	1.592667	281494500
1	2010-06-30	1.719333	2.028000	1.553333	1.588667	1.588667	257806500
2	2010-07-01	1.666667	1.728000	1.351333	1.464000	1.464000	123282000
3	2010-07-02	1.533333	1.540000	1.247333	1.280000	1.280000	77097000
4	2010-07-06	1.333333	1.333333	1.055333	1.074000	1.074000	103003500

5 rows × 7 columns Open in new tab

Then it shows the dimensions of the data and the index's range.

```
print(my_data.shape)
```

```
(3156, 9)
```

```
print(my_data.index)
```

```
Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
            ...
            3146, 3147, 3148, 3149, 3150, 3151, 3152, 3153, 3154, 3155],
           dtype='int64', length=3156)
```

The next step is checking the data types. It is visible that the data type of Date is object. In order to working on timeseries data is needed to change their type to datetime64.

```
print("Data types are:")
print(my_data.dtypes)
```

```
Data types are:
Date            object
Open           float64
High           float64
Low            float64
Close          float64
Adj Close      float64
Volume           int64
dtype: object
```

Here is data types after changing to data’s type.

```
my_data['Date']=pd.to_datetime(my_data['Date'])

print(my_data.dtypes)
```

```
Date          datetime64[ns]
Open                 float64
High                 float64
Low                  float64
Close                float64
Adj Close            float64
Volume                 int64
dtype: object
```

With using describe function, we can get some information about each column.

```python
print("Data describe in csv fil:")
print(my_data.describe())
```

```
count   3157.000000  3157.000000  3157.000000  3157.000000  3157.000000
mean      58.984488    60.306736    57.520747    58.932712    58.932712
std       95.589129    97.785209    93.105920    95.457672    95.457672
min        1.076000     1.108667     0.998667     1.053333     1.053333
25%        9.000000     9.196667     8.821333     9.030000     9.030000
50%       16.261999    16.506666    15.985333    16.273333    16.273333
75%       24.833332    25.162666    24.288668    24.749332    24.749332
max      411.470001   414.496674   405.666656   409.970001   409.970001

             Volume
```

Now, it's time to think about, how we can predict tomorrows' price and how we can say if tomorrows' price goes up or down. The idea for this problem is that we can add 2 columns in the dataset. One column is for tomorrow which for each day we shift close price of the day after on it, then for each day we know tomorrow's price. Another column is Target which shows if the price goes up or down. With comparing today's tomorrow column and Close price for the next day we can guess the values of this column.

```python
#adding tomorrow column to the datasets
#Tomorrow is includes the day after close prive
my_data["Tomorrow"]=my_data["Close"].shift(-1)
#Adding another column for Tagrget. if tomorrow's price is higher than close price then it set 1 in the target
my_data["Target"]=(my_data["Tomorrow"]>my_data["Close"]).astype(int)
my_data.head(10)
```

	Date	Open	High	Low	Close	Adj Close	Volume	Tomorrow	Target
0	2010-06-29	1.266667	1.666667	1.169333	1.592667	1.592667	281494500	1.588667	0
1	2010-06-30	1.719333	2.028000	1.553333	1.588667	1.588667	257806500	1.464000	0
2	2010-07-01	1.666667	1.728000	1.351333	1.464000	1.464000	123282000	1.280000	0
3	2010-07-02	1.533333	1.540000	1.247333	1.280000	1.280000	77097000	1.074000	0
4	2010-07-06	1.333333	1.333333	1.055333	1.074000	1.074000	103003500	1.053333	0
5	2010-07-07	1.093333	1.108667	0.998667	1.053333	1.053333	103825500	1.164000	1
6	2010-07-08	1.076000	1.168000	1.038000	1.164000	1.164000	115671000	1.160000	0
7	2010-07-09	1.172000	1.193333	1.103333	1.160000	1.160000	60759000	1.136667	0

10 rows × 9 columns Open in new tab

Now it's the time for finding missing values. In this case (AAPL) has only one missing column. So, how we can deal with missing values. The first way that been checked on the data to handling missing data was using SimpleImputer. But this method wasn't a good idea for the dataset as it replaced the missing values with the mean of that column and in our dataset, we have timeseries that can not have mean. On the other hand, the number of missing values compare to 25 years data it's nothing and wouldn't affect to the result much. So, decided to drop the rows with missing values.

```
#Finding missing data

#printing rows containing empty variables
my_empty_data=my_data[my_data.isna().any(axis=1)]
print("#####Missing data #######")
print(my_empty_data)

my_data=my_data.dropna()
```

```
#####Missing data #######
           Date        Open        High         Low       Close   Adj Close  \
3156 2023-01-11  122.089996  125.940002  120.510002  123.199203  123.199203

        Volume  Tomorrow  Target
3156  73384688       NaN       0
```


```
#Finding missing data

#printing rows containing empty variables
my_empty_data=my_data[my_data.isna().any(axis=1)]
print("#####Missing data #######")
print(my_empty_data)

my_data=my_data.dropna()
```

```
#####Missing data #######
           Date        Open        High         Low       Close   Adj Close  \
3156 2023-01-11  122.089996  125.940002  120.510002  123.199203  123.199203

        Volume  Tomorrow  Target
3156  73384688       NaN       0
```

```
#recheck the missing values

#printing rows containing empty variables
my_empty_data=my_data[my_data.isna().any(axis=1)]
print("#####Missing data #######")
print(my_empty_data)
```

```
#####Missing data #######
Empty DataFrame
Columns: [Date, Open, High, Low, Close, Adj Close, Volume, Tomorrow, Target]
Index: []
```

```
#Finding dublicated rows
```

Let’s find duplicated rows too. There is not any duplicated row for this case(AAPL).

```
#Finding dublicated rows

print(my_data.duplicated().sum())
```

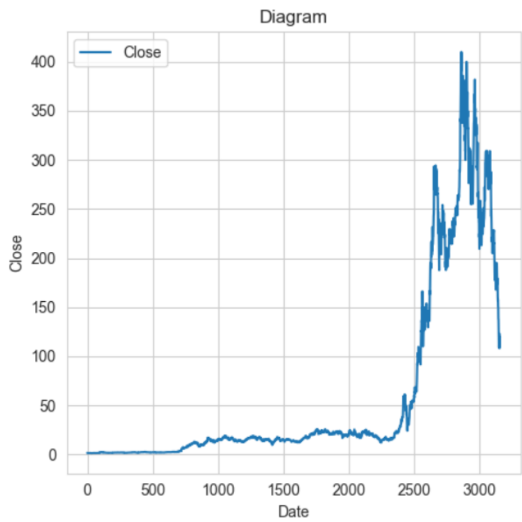
```
0
```

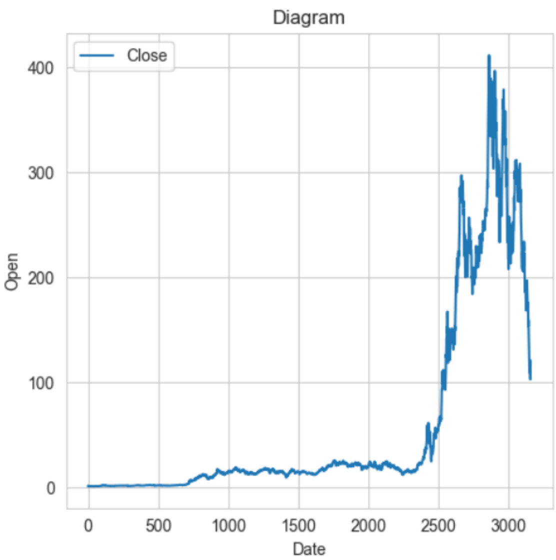
Bivariate and univariate analysis

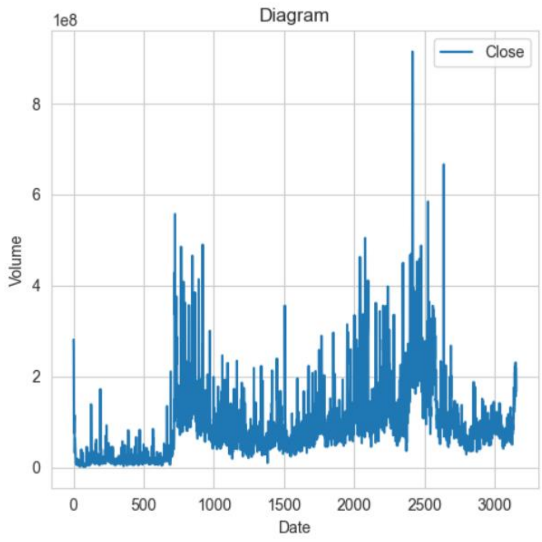
Draw_diagram function gets the data and specific column as its input and then draws a diagram and save it to images folder. These diagrams shows that the price of AAPL went up since it started to work.

```python
#draw a graph function
def draw_diagram(data,column):
    plt.figure(figsize=(5,5))
    plt.title('Diagram')
    plt.xlabel('Date')
    plt.ylabel(f'{column}')
    plt.plot(data[f'{column}'])
    plt.legend(['Close'])
    plt.savefig('images/grapg.png')

draw_diagram(my_data,'Close')
draw_diagram(my_data,'Open')
draw_diagram(my_data,'Volume')
```



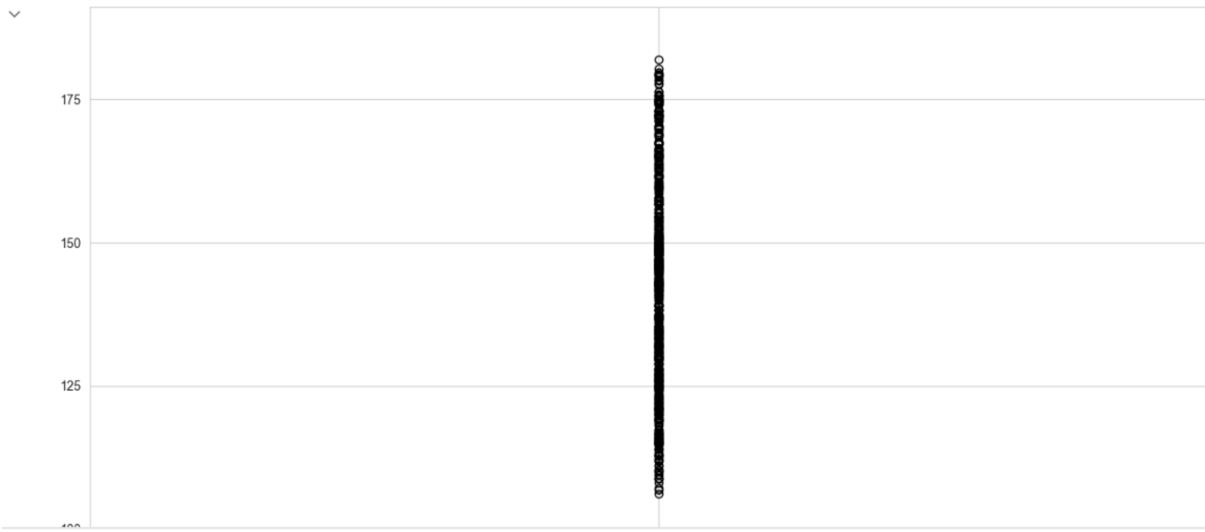




To checking outliers, Box plot is a great tool. This shows there outliers for the AAPL's close price.

```
#boxplot for to find outliers
my_data.boxplot(column=["Close"])
```

```
<AxesSubplot: >
```



Here is median of each column and the max value of Close column.

```
print("Median :")
print(my_data.median())

```

```
Median :
Open          9.233214e+00
High          9.299286e+00
Low           9.118214e+00
Close         9.238214e+00
Adj Close     7.874819e+00
Volume        3.077592e+08
Tomorrow      9.238571e+00
Target        1.000000e+00
dtype: float64
```

```
C:\Users\nastaran\AppData\Local\T
DataFrame.median with numeric_onl
  print(my_data.median())
```

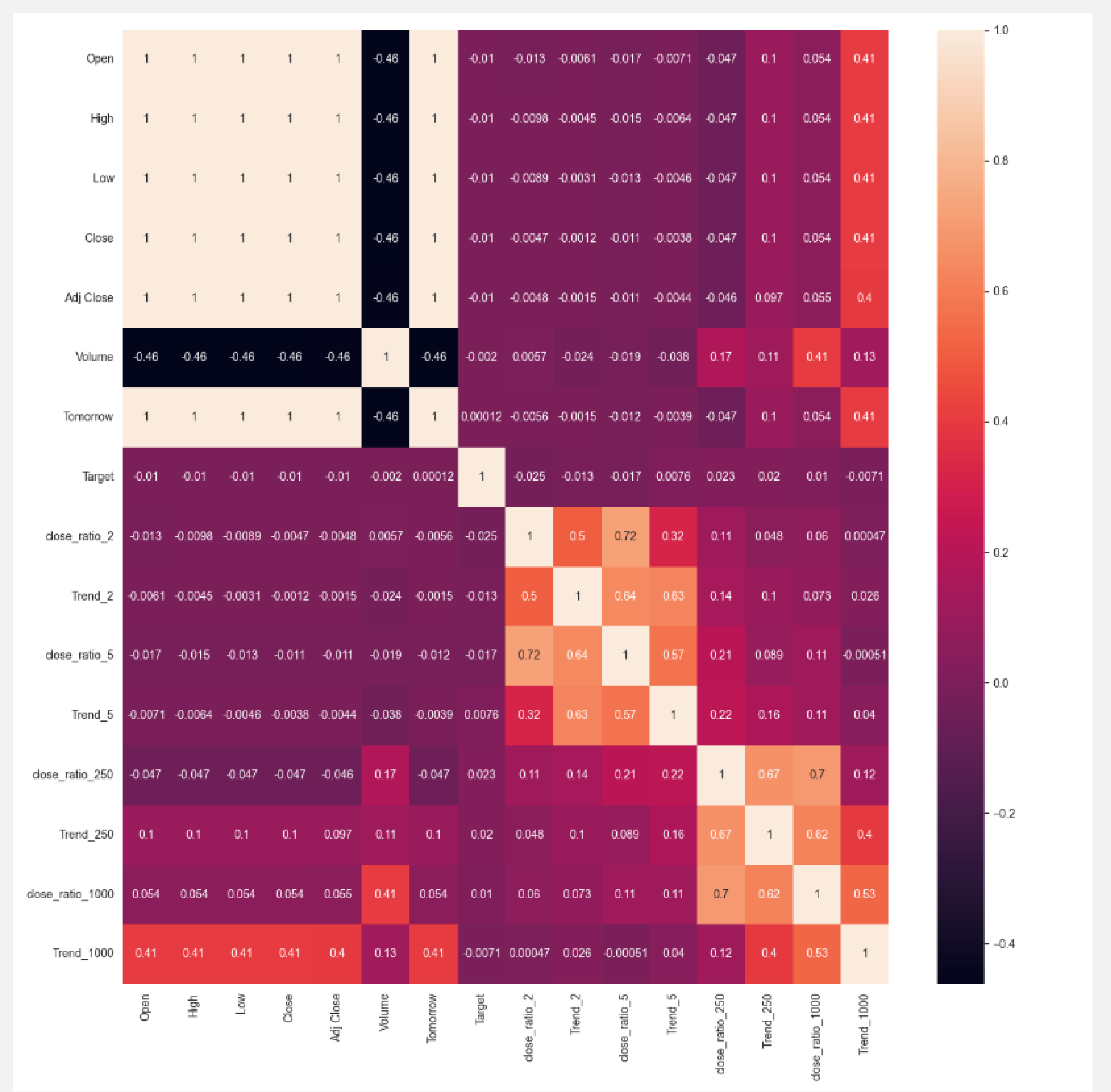
```
#getting max  for each cryptos

my_data['Close'].max()
```

```
182.009995
```

Now it's time to find the correlation between each variable. The method for finding correlation matrix is Pearson method. It is visible that there is not any relationship between tomorrow's price and other variables.

```
#Create a Correlation Matrix using Pandas for cryptos
corrMatrix=my_data.corr(method='pearson')
sn.heatmap(corrMatrix, annot=True)
plt.rcParams["figure.figsize"] = [13,13]
plt.rcParams["figure.autolayout"] = True
plt.show()
#As it  is vissible there is not any linear correlation
```



Open
High
Low
Close
Adj Close
Volume
Tomorrow
Target
close_ratio_2
Trend_2
close_ratio_5
Trend_5
close_ratio_250
Trend_250
close_ratio_1000
Trend_1000

```
correMatrix_tom=my_data.corr()[['Tomorrow']]
correMatrix_tom.style.background_gradient(cmap='YlOrRd').set_precision(2)
#between Tomorrow and close there is strongly possitive relationship
```

```
C:\Users\nastaran\AppData\Local\Temp\ipykernel_26752\3020489423.py:1: FutureWarning: Th
```

	Tomorrow
Open	1.00
High	1.00
Low	1.00
Close	1.00
Adj Close	1.00
Volume	-0.46
Tomorrow	1.00
Target	0.00
close_ratio_2	-0.01
Trend_2	-0.00
close_ratio_5	-0.01
Trend_5	-0.00
close_ratio_250	-0.05
Trend_250	0.10
close_ratio_1000	0.05
Trend_1000	0.41

Skew’s values shows that the target is normally distributed or for high, open, close distribution is highly skewed.

```
#checking for skewness in cryptos data
print(my_data.skew())
```

```
Open              1.826602
High              1.828383
Low               1.824288
Close             1.826324
Adj Close         1.865373
Volume            2.091255
Tomorrow          1.824520
Target           -0.095771
close_ratio_2    -0.186130
Trend_2          -0.066600
close_ratio_5    -0.179693
```

Step3:Demonstrate a good understanding of the datasets. Justify the univariate and multivariate analysis with visualisation. Show good understanding of the information derived from the analysis.

Applying model

The model that decided to predict the value is RandomeForestClassifier. RandomForestClassifier is a supervised learning algorithm that has n decision trees, and the data will be train in different sunsets and will use the average of each subset to predict the value. The reason that I decided to use this algorithm is RandomForestClassifier can be overfit less than other algorithms and resistance to overfitting. Also, this algorithm can take non-linear relationships and as there is not any linear relationship between target and close price, so, this algorithm is the best for my dataset. The parameters for RandomForestClassifier function are:

1. N_estimators: The number of individual decision tress.
2. Min_sample_split: The minimum number of samples needs for split.
3. Random_state
4. Max_depth

```
model=RandomForestClassifier(n_estimators=100,min_samples_split=100,random_state=1)
```

RandomForestClassifier doesn't need scaling before applying the model as it works with trees.

The first approach:

It's been decided to use all the data in our dataset except the last 100 data to predicting the last 100 rows. It means we train the last 100 rows of data from other part of dataset. The reason is that we are working on timeseries dataset and can not use future data to predict the past. Also, the predictors variable is all labels except Target. And as it's been mentioned before the precision_score will be used to evaluate the algorithm. This measure tells the percentage of those days that algorithm tells us the price goes up and the price went up.

The first approach's prediction_score is %47.

***as the application is too heavy to run this part been commented out.

```
model=RandomForestClassifier(n_estimators=100,min_samples_split=100,random_state=1)
#all data except the last 100
predictors=["Close","Volume","Open","High","Low"]

#spiliting train and test
#all data except the last 100 rows
train=my_data.iloc[:-100]
#the last 100 rows
test=my_data.iloc[-100:]
#print(train)
#print(test)

X_train=train[predictors]
y_train=train["Target"]
X_test=test[predictors]
y_test=test["Target"]
model.fit(X_train,y_train)
#now let's predict the model
y_pred_train=model.predict(X_train)
y_pred=model.predict(X_test)
y_pred=pd.Series(y_pred, index=test.index)
ps=precision_score(y_test,y_pred)
print("precision score: {:.2f}".format((ps)))
```

16

```
precision score: 0.47

<AxesSubplot: >
```

The Second approach:

For the second approach I added two functions. Predict function that gets train, test, predictors and the model and returns a series. Also, there is another function that calls backtest that calls predict function to predict the next n+1 years' price. For example, it gets 10 years data and predict prices for the next 11 years. Start is 2500 as each year has 250 days that stock works, so it means 10years. This approach score is %52. We had improvements, but we will try to make it better.

***as the application is too heavy to run this part been commented out.

```python
model=RandomForestClassifier(n_estimators=100,min_samples_split=100,random_state=1)

#this function predict the result
predictors=["Close","Volume","Open","High","Low"]
def predict(train,test,predictors,model):
        model.fit(train[predictors],train["Target"])
        #now let's predict the model
        preds=model.predict(test[predictors])
        preds=pd.Series(preds, index=test.index,name="predictions")
        combined=pd.concat([test["Target"],preds],axis=1)
        return combined
#2500 mean 10 years as each trading year is 250 days
#it train the model with 10 years data
#it trains the model yearly(step=250)
def backtest(data,model, predictors, start=2500,step=250):
    al_predictions=[]
    #loop through each year's data
    for i in range(start,data.shape[0],step):
        #till current year
        train=data.iloc[0:i].copy()
        #current year plus one year
        #for example 10 it gets 10 years data and then predict model for the next 11 years
        test=data.iloc[i:(i+step)].copy()
        predictions=predict(train,test,predictors,model)
        al_predictions.append(predictions)
    return pd.concat(al_predictions)
```

```
0.5232
```

Model Tuning

In In model tunning, we tune the hyperparameters. In this case the hyperparameters are n_estimators, min_sample_split and random_state and max_depth. Grid search method has been chosen to tune the model. A dictionary considered to add some sample data for the hyperparameters. Then it will suggest the hyperparameters for the model.

Unfortunately, this part of the code is heavy, and I couldn’t get any answer after waiting a long time as the dataset is big. So, I decided to tune the hyperparameters manually later.

***as the application is too heavy to run this part been commented out.

```
#declare a dictionary of hyperparameter and values
classifier_hypara=dict()
classifier_hypara['max_depth']=[2,3,4,8,10]
classifier_hypara['min_samples_split']=[2,4,6,8,9]
classifier_hypara['n_estimators']=[10,20,50,100,110]
classifier_hypara['criterion']=['gini','entropy']

X=my_data[predictors]
y=my_data["Target"]
#perform a gridsearch and fit the grid
classifier_grid=GridSearchCV(my_model,classifier_hypara, scoring='accuracy', n_jobs=-1, cv=kfolds_split)
classifier_grid_fit=classifier_grid.fit(X,y)

#compute the array containing the 10 folds and calculate the cros validation mean score
CV_scores=-cross_val_score(classifier_grid_fit,X_train, y_train, cv=kfolds_split)
print("\nCross Val mean: {:.3f} (std: {:.3f})".format(CV_scores.mean()*-1,CV_scores.std()),end="\n\n" )

#we cab print teh hyperparameter tuning results
print('Best Hyperparameters: %s' %classifier_grid_fit.best_params_)
print('Best max_depth=', classifier_grid_fit.best_estimator_.get_params()['max_depth'])
print('Best min_samples_aplit =', classifier_grid_fit.best_estimator_.get_params()['min_samples_split'])
print('Best min_samples_leaf =', classifier_grid_fit.best_estimator_.get_params()['min_samples_leaf'])
print('Best criterion', classifier_grid_fit.best_estimator_.get_params()['criterion'])

#print best hyperparameteres
print('\n Suggested Best Hyperparameters: \n', classifier_grid_fit.best_estimator_.get_params())
print('best score: %s {:.3f}\n'.format(classifier_grid_fit.best_score_))
```

The Third approach:

For this approach I have done a bit change on the predict function. Instead of using predict I used predict_probba, then if the probability of price going up is more than 60% then it predicts 1 otherwise it predicts 0.

```
predictors=["Close","Volume","Open","High","Low"]
def predict(train,test,predictors,model):
        model.fit(train[predictors],train["Target"])
        #now let's predict the model
        #using propability of price going up
        preds=model.predict_proba(test[predictors])[:,1]
        #if it is more than 0.6
        preds[preds>=0.6]=1
        preds[preds<0.6]=0
        preds=pd.Series(preds, index=test.index,name="predictions")
        combined=pd.concat([test["Target"],preds],axis=1)
        return combined
```

This time I added a few new columns to the dataset. Trend2, Trend5, Trend 250 and Trend100. For example, Trend250 tells the number of days that price went up. And, adding ratio column for all of trends column. Ratio2, Ratio 5, Ratio 250 and Ratio1000. For Example, Ratio250 is the mean of last 250 days. We can get today's price through my_data["Tomorrow"].iloc[-2] and tomorrow's price through my_data["Tomorrow"].iloc[-1]. Now, with simple if else statement we check that if today's Target is one means the price goes up and if not means the price doesn't go up.

The accuracy of this algorithms is %60.

```
hor=[2,5,250,1000]
new_pred=[]
for h in hor:
    #calculating avarage for each horizon
    rolling_avr=my_data.rolling(h).mean()
    #adding ratio column
    ratio_column=f"close_ratio_{h}"
    #adding the value of ratio
    my_data[ratio_column]=my_data["Close"]/rolling_avr["Close"]
    #adding another column for trend
    #This shows the numver of column that in the last x days price went up
    trend_column=f"Trend_{h}"
    #it looked at a few days ago and returns sum of target
    my_data[trend_column]=my_data.shift(1).rolling(h).sum()["Target"]
    new_pred+=[ratio_column,trend_column]

my_data=my_data.dropna()


predictions=backtest(my_data,model,new_pred)
predictions["predictions"].value_counts()
print("The accutacy of test:", precision_score(predictions["Target"],predictions["predictions"]))
```

```
###prediction#######
print(my_data.tail(10))
today_price=my_data["Tomorrow"].iloc[-2]
tommorows_price=my_data["Tomorrow"].iloc[-1]
flag=my_data["Target"].iloc[-1]
print("Today's date",today)
print("Tomorrow's price",tommorows_price)
if (flag==1):
    print(f"The Price for {file_name} will go up")
elif (flag==0):
    if (tommorows_price<today_price):
        print(f"The price for {file_name} will be decresed")
    else:
        print(f"The price for {file_name} will be same as today")

```

```
    else:
        print(f"The price for {file_
```

```
> C:\Users\nastaran\AppData\Local\Temp

  The accutacy of test: 0.6
             Date        Open
  6281 2022-12-27  131.380005  131.41
```

Now let's tune our model manually to get the best collections of hyperparameters for AAPL. Different input has been tested on the dataset and saved the accuracy of that in the table. The collection of n_estimators=250, min_samples_split=50 and max_depth=5 gives the best score between them which is the accuracy of %64.

n_estimators	min_samples_split	max_depth	Accuracy
200	40	5	%60
150	40	5	%56
250	40	4	%60
250	50	5	%64
250	30	5	%62
250	50	10	%59

```
C:\Users\nastaran\AppData\Local\Temp\ipykernel_

 The accutacy of test: 0.6428571428571429
             Date        Open        High
 6281 2022-12-27  131.380005  131.410004  128
```

Another option that we can check to see if it can apply improvement in the accuracy is changing the train and test data.

```
#all data except the last 100 rows
#100 working better than 150 and 200
train=my_data.iloc[:-200]
#the last 100 rows
test=my_data.iloc[-200:]
#print(train)
#print(test)
```

```
C:\Users\nastaran\AppData\Local\Temp\ipykern

The accutacy of test: 0.6307692307692307
           Date        Open        High
6281 2022-12-27  131.380005  131.410004  12
```

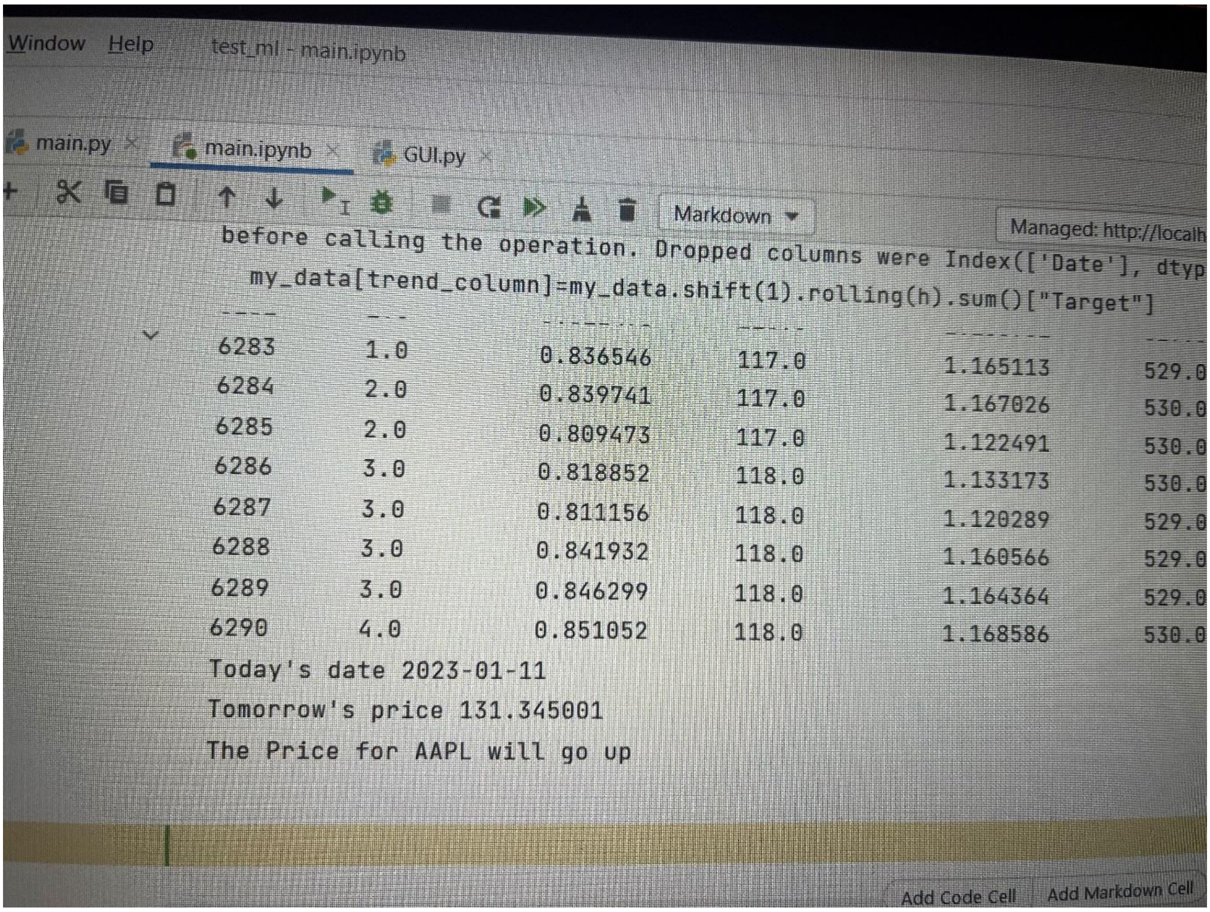
```
#100 working better than 150 and 20
train=my_data.iloc[:-150]
#the last 100 rows
test=my_data.iloc[-150:]
#print(train)
```

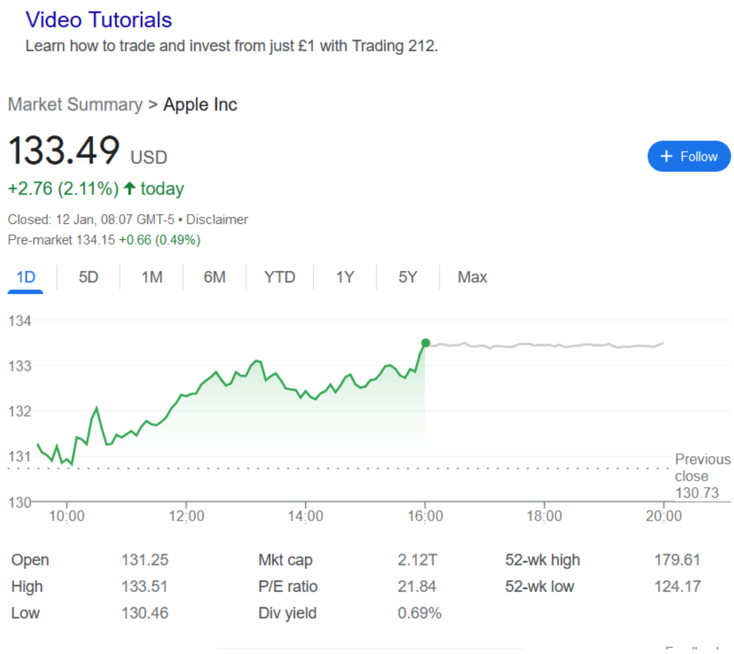
```
The accutacy of test: 0.5555555555555556
           Date        Open        High
6281 2022-12-27  131.380005  131.410004  1
```

It is visible that changing it to 200 is working better.

Prediction In the real world:

To checking the model in the real world, on 11th of January 2023 the model predicted the price for AAPL will be go up to 131.34. Let’s check today’s price for AAPL(12th of January 2023).





The price of AAPL went up and the price is really close to the predicted price in yesterday.