Project Report

ARTIFICIAL INTELLIGENCE (UCS411)

STOCK PRICE PREDICTION

Submitted by:

Shivam Aggarwal (102117034)

Radhika Goel (102117036)

Sarabdeep Singh Raikhi (102117055)

Gurpreet Singh (102117212)

Second Year CSE-II

Submitted to:

Mrs. Navpreet Kaur

Lab Instructor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

ABSTRACT

The principal objective of this project was to learn to use existing data models and increase the accuracy of prediction by combining them. We achieved the said objective by using the Ensemble Methods in machine learning. Ensemble methods usually produce more accurate solutions rather than a single model. We used an existing dataset of **'STOCK PRICE PREDICTION'** for experimentation and analysis for this project. Each team member was able to apply programming skills by integrating coding, optimization, and research for every experiment.

DECLARATION

We, the undersigned, solemnly declare that the project report is based on our own work carried out during the course of our study under the supervision of **Mrs. Navpreet Kaur**.

We assert that the statements made and conclusions drawn are an outcome of our research work. We further certify that-

I. The work contained in the report is original and has been done by us under the general supervision of our supervisor.

II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or the any other University of India or abroad.

III. We have followed the guidelines provided by the university in writing the report.

IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and given their details in the references.

INDEX

<u>INTRODUCTION</u>

Stock markets in general have an air of unpredictability. A stock market, equity market or share market is the aggregation of buyers and sellers (a loose network of economic transactions, not a physical facility or discrete entity) of stocks (also called shares), which represent ownership claims on businesses; these may include securities listed on a public stock exchange, as well as stock that is only traded privately. Examples of the latter include shares of private companies which are sold to investors through equity crowdfunding platforms. Stock exchanges list shares of common equity as well as other security types, e.g. corporate bonds and convertible bonds. Participants in the stock market range from small individual stock investors to larger investors, who can be based anywhere in the world, and may include banks, insurance companies, pension funds and hedge funds. Their buy or sell orders may be executed on their behalf by a stock exchange trader .

Because we're dealing with time series data, we can't just use cross-validation to create predictions for the whole dataset. This will cause leakage where data from the future will be used to predict past prices. This doesn't match with the real world, and will make us think that our algorithm is much better than it actually is.

Instead, we'll split the data sequentially. We'll start off by predicting just the last 100 rows using the other rows.We'll use a random forest classifier to generate our predictions. This is a good "default" model for a lot of applications, because it can pick up nonlinear relationships in the data, and is somewhat robust to overfitting with the right parameters.

This is just to set up our model and ensure that things are working properly. To get an accurate error metric, we need to backtest across our entire price history.

LITERATURE SURVEY

Literature survey is the process in which a complete and comprehensive review is conducted encompassing both the published and unpublished work from other alternative sources of information. This review is conducted in the domains of specific interest to the person or researcher. Further, the results of this process are documented. This entire process comes in aid of the researcher to address the important and relevant aspects of the research that had not been addressed prior to the conduction of this research. Therefore, it can be understood that the conduct of literature survey is necessary for the process of gathering secondary data for the research which might prove to be extremely helpful in the research and also designing the architecture of the project. There can be multiple reasons behind the purpose of conducting literature surveys.

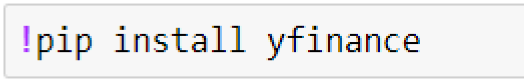
REFERENCES:

1. *"Stock price prediction using LSTM, RNN and CNN-sliding window model - IEEE Conference Publication."* https://ieeexplore.ieee.org/document/8126078 (accessed Dec. 27, 2019).
2. J. Jagwani, M. Gupta, H. Sachdeva, and A. Singhal, *"Stock Price Forecasting Using Data from Yahoo Finance and Analysing Seasonal and Nonseasonal Trend."* in 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, Jun. 2018, pp. 462–467, doi: 10.1109/ICCONS.2018.8663035.
3. S. Sharma and B. Kaushik, *"Quantitative analysis of stock market prediction for accurate investment decisions in future."* Journal of Artificial Intelligence, vol. 11, pp. 48–54, 2018.
4. Y. Lei, K. Zhou, and Y. Liu, *"Multi-Category Events Driven Stock Price Trends Prediction."* in 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), Nanjing, China, Nov. 2018, pp. 497–501, doi: 10.1109/CCIS.2018.8691392.
5. B. Jeevan, E. Naresh, B. P. V. kumar, and P. Kambli, *"Share Price Prediction using Machine Learning Technique."* in 2018 3rd International Conference on Circuits, Control, Communication and Computing (I4C), Bangalore, India, Oct. 2018, pp. 1–4, doi: 10.1109/CIMCA.2018.8739647.
6. M. Usmani, S. H. Adil, K. Raza, and S. S. A. Ali, *"Stock market prediction using machine learning techniques."* in 2016 3rd International Conference on computer and Information Sciences (ICCOINS), 2016, pp. 322–327.

DATASET DOCUMENTATION

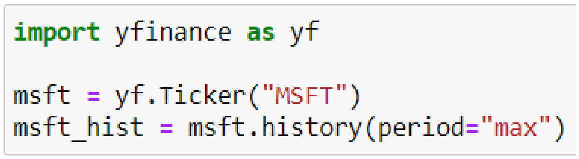
DOWNLOADING THE DATA:

We've used data for a single stock (Microsoft) from when it started trading to the present.



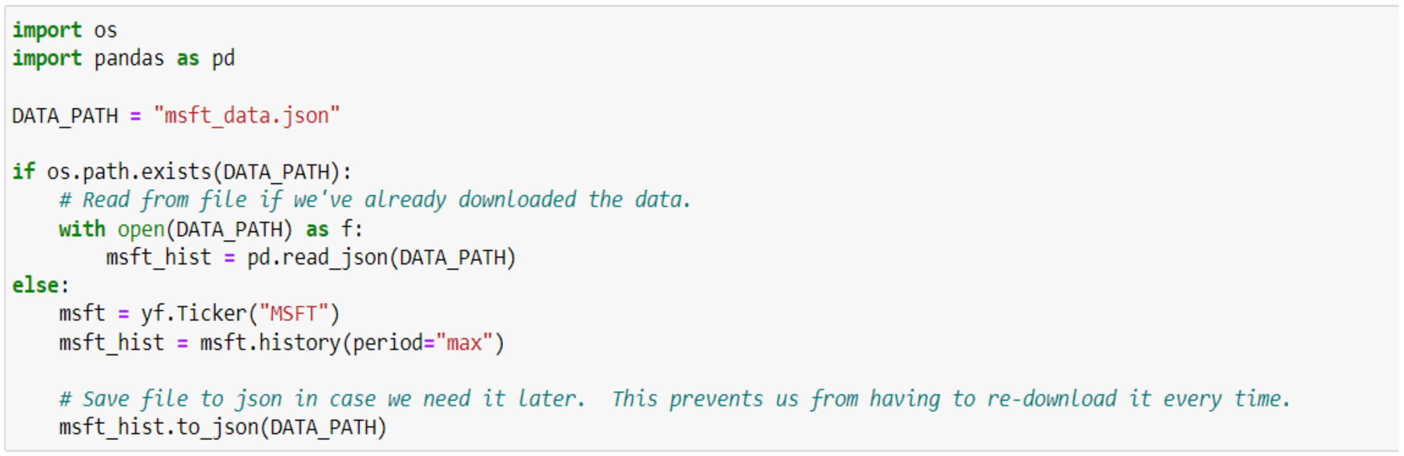
```
!pip install yfinance
```

# downloading the *yfinance* python package



```
import yfinance as yf

msft = yf.Ticker("MSFT")
msft_hist = msft.history(period="max")
```

We've now downloaded the data! In the real world, we want to save the data to disk, so we can access it again if we need it without calling the API over and over. We can do this by checking if we've saved the data before. If we have, we just load the data. Otherwise, we download the data. We need to install **pandas** to do this.



```
import os
import pandas as pd

DATA_PATH = "msft_data.json"

if os.path.exists(DATA_PATH):
    # Read from file if we've already downloaded the data.
    with open(DATA_PATH) as f:
        msft_hist = pd.read_json(DATA_PATH)
else:
    msft = yf.Ticker("MSFT")
    msft_hist = msft.history(period="max")

    # Save file to json in case we need it later.  This prevents us from having to re-download it every time.
    msft_hist.to_json(DATA_PATH)
```

EXPLORING THE DATA:

Next, we take a look at **msft_hist** to see the structure of the data. We've used the .head method on DataFrames to check the first 5 rows of the data.

As we can see below, we have one row of data for each day that Microsoft stock was traded. Here are the columns:

- Open - the price the stock opened at
- High - the highest price during the day
- Low - the lowest price during the day
- Close - the closing price on the trading day
- Volume - how many shares were traded

The row index of the DataFrame is the date the stock was traded. Stock doesn't trade every day (there is no trading on weekends and holidays), so some dates might be missing.

```
msft_hist.head(5)
```

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
1986-03-13 00:00:00-05:00	0.055241	0.063365	0.055241	0.060657	1031788800	0.0	0.0
1986-03-14 00:00:00-05:00	0.060657	0.063907	0.060657	0.062823	308160000	0.0	0.0
1986-03-17 00:00:00-05:00	0.062823	0.064448	0.062823	0.063907	133171200	0.0	0.0
1986-03-18 00:00:00-05:00	0.063907	0.064448	0.061740	0.062281	67766400	0.0	0.0
1986-03-19 00:00:00-05:00	0.062281	0.062823	0.060657	0.061198	47894400	0.0	0.0

PROBLEM DEFINITION & ALGORITHM

PROBLEM STATEMENT:

In this project, we've learned how to predict stock prices using python, pandas, and scikit-learn. Along the way, we've downloaded stock prices, created a machine learning model, and developed a back-testing engine. The model needs to predict tomorrow's closing price using data from today. If the model says that the price will increase, we'll buy stock. If the model says that the price will go down, we won't do anything.

METHOD:

We'll be looking at Microsoft stock, which has the stock symbol MSFT. Here are the steps that we'll follow to make predictions on the price of MSFT stock:

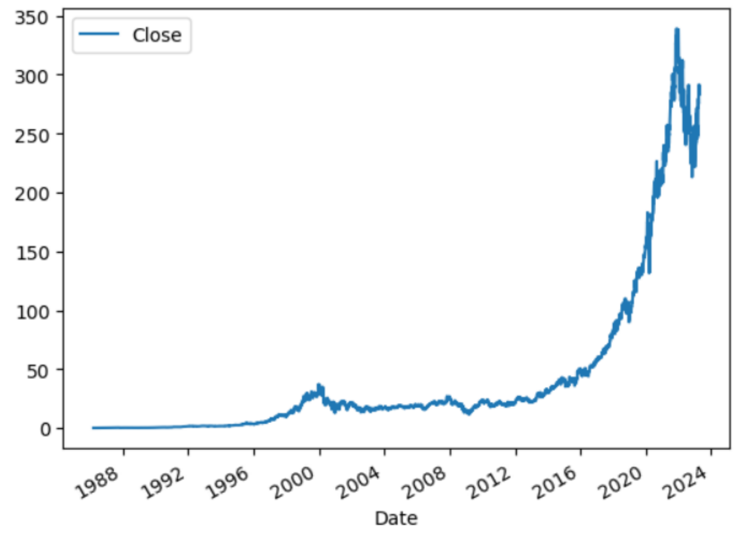
- Download MSFT stock prices from Yahoo finance
- Explore the data
- Setup the dataset to predict future prices using historical prices
- Test a machine learning model
- Set up a back-testing engine
- Improve the accuracy of the model

ALGORITHM:

From the dataset given above, let's plot the data so we can see how the stock price has changed over time. This gives us another overview of the structure of the data. We can use the built-in plot method on DataFrames to do this. We use the **use_index** parameter because we want to use the index as the x-axis values.

```python
# Visualize microsoft stock prices
msft_hist.plot.line(y="Close", use_index=True)
```

```
<AxesSubplot:xlabel='Date'>
```



SETTING UP THE TARGET:

We do this by first copying the Close column to a new DataFrame called data and renaming it to actual_close. This ensures that we can continue to see the value that the stock actually closed at on each day.

Then, we setup our target by doing the following:

- Using the pandas rolling method across every 2 rows of the DataFrame. This will first look at ('1986-03-13', '1986-03-14'), then ('1986-03-14', '1986-03-17'), and so on across the DataFrame.
- We will compare the second row to the first row to see if it is greater. If it is, return 1, otherwise 0.
- We then just look at the Close column.

As you can see above, the Target column now indicates if the price went up or down on the given day. If Target is 1, the price went up. If Target is 0, the price went down.

This is what we want our machine learning model to predict!

```python
# Ensure we know the actual closing price
data = msft_hist[["Close"]]
data = data.rename(columns = {'Close':'Actual_Close'})

# Setup our target.  This identifies if the price went up or down
data["Target"] = msft_hist.rolling(2).apply(lambda x: x.iloc[1] > x.iloc[0])["Close"]
```

```python
data.head()
```

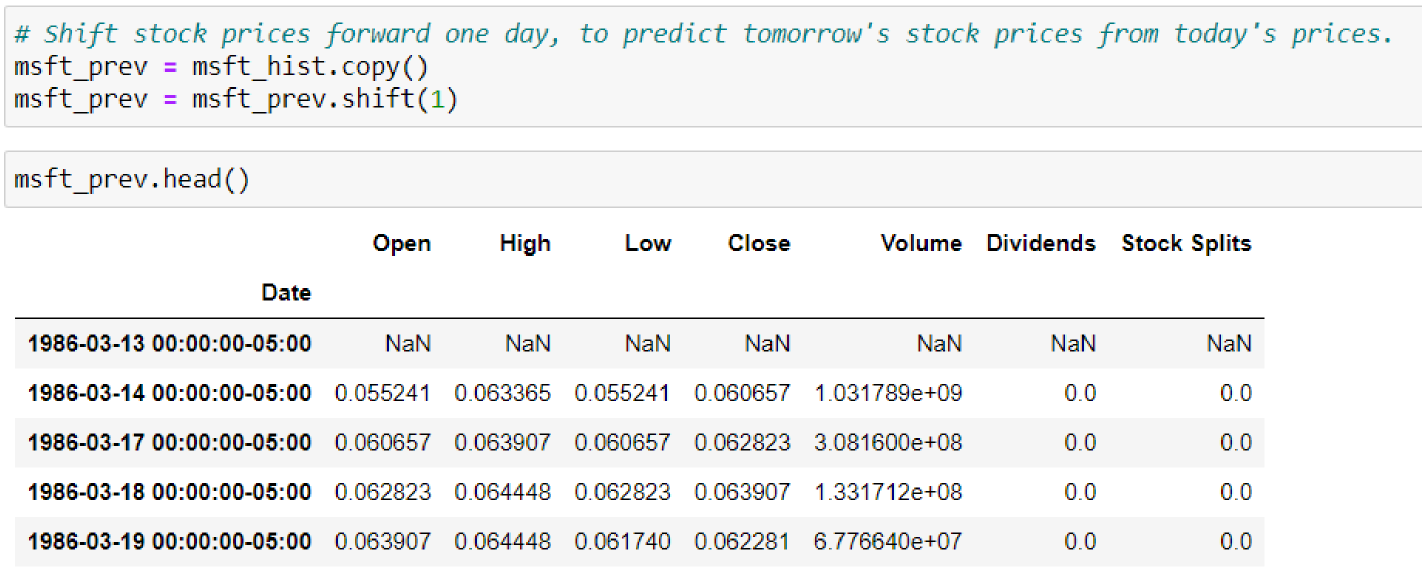
```
                           Actual_Close  Target
                     Date
1986-03-13 00:00:00-05:00      0.060657     NaN
1986-03-14 00:00:00-05:00      0.062823     1.0
1986-03-17 00:00:00-05:00      0.063907     1.0
1986-03-18 00:00:00-05:00      0.062281     0.0
1986-03-19 00:00:00-05:00      0.061198     0.0
```

SHIFTING DATA "FORWARD":

Next, we used the DataFrame **shift** method to move all rows "forward" one trading day.

As you can see, the prices for 1986-03-13 are now associated with 1986-03-14, and every other price is shifted up one row. This is to ensure that we're predicting future prices using past data.

If we didn't do this, we'd be using data from 03-14 to predict prices on 03-14. Instead, we need to use data from 03-13 to predict prices on 03-14. If we don't do this, our model will look amazing when we're testing it, but won't work at all in the real world. In the real world, we don't actually know the price tomorrow, so we can't use it to make our predictions.



```
# Shift stock prices forward one day, to predict tomorrow's stock prices from today's prices.
msft_prev = msft_hist.copy()
msft_prev = msft_prev.shift(1)
```

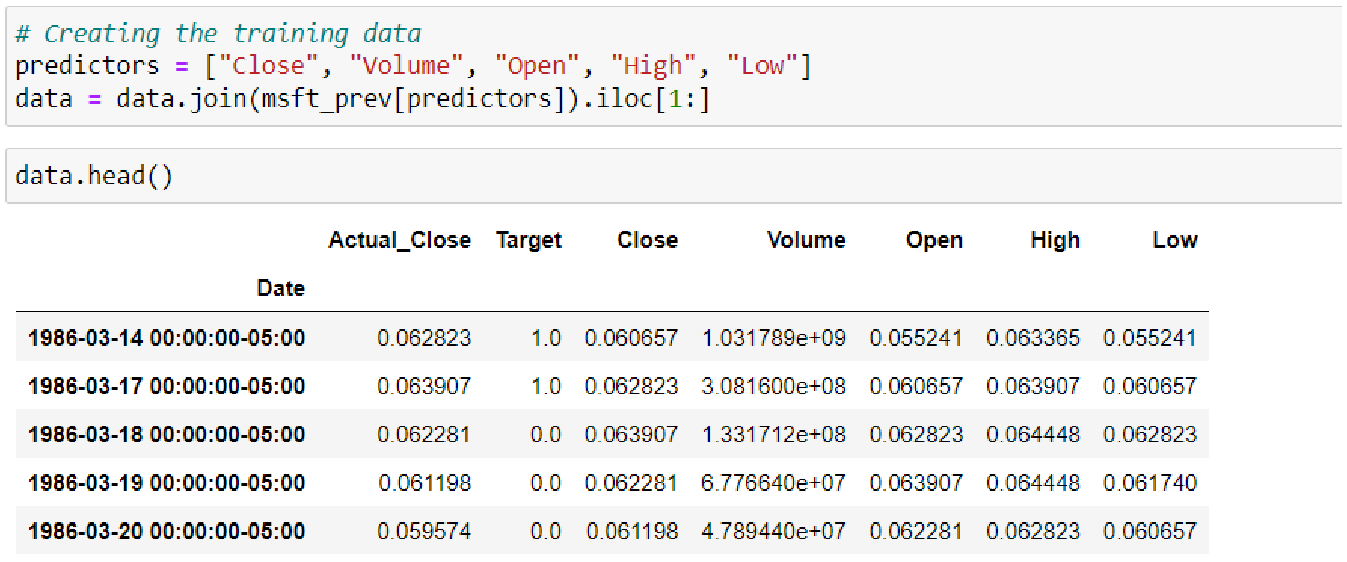
```
msft_prev.head()
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1986-03-13 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1986-03-14 00:00:00-05:00	0.055241	0.063365	0.055241	0.060657	1.031789e+09	0.0	0.0
1986-03-17 00:00:00-05:00	0.060657	0.063907	0.060657	0.062823	3.081600e+08	0.0	0.0
1986-03-18 00:00:00-05:00	0.062823	0.064448	0.062823	0.063907	1.331712e+08	0.0	0.0
1986-03-19 00:00:00-05:00	0.063907	0.064448	0.061740	0.062281	6.776640e+07	0.0	0.0

<u>COMBINING THE DATA:</u>

Next, we need to combine our **Target** with the columns we want to use to predict the target. We've used this with the join method on DataFrames.

Then we predicted the target using data from the previous day. The columns we used to predict our target are : **["Close", "Volume", "Open", "High", "Low"]**.



```
# Creating the training data
predictors = ["Close", "Volume", "Open", "High", "Low"]
data = data.join(msft_prev[predictors]).iloc[1:]
```

```
data.head()
```

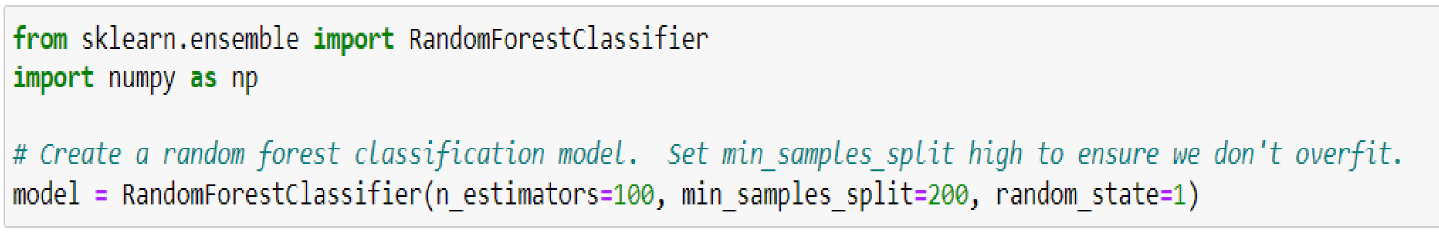
Date	Actual_Close	Target	Close	Volume	Open	High	Low
1986-03-14 00:00:00-05:00	0.062823	1.0	0.060657	1.031789e+09	0.055241	0.063365	0.055241
1986-03-17 00:00:00-05:00	0.063907	1.0	0.062823	3.081600e+08	0.060657	0.063907	0.060657
1986-03-18 00:00:00-05:00	0.062281	0.0	0.063907	1.331712e+08	0.062823	0.064448	0.062823
1986-03-19 00:00:00-05:00	0.061198	0.0	0.062281	6.776640e+07	0.063907	0.064448	0.061740
1986-03-20 00:00:00-05:00	0.059574	0.0	0.061198	4.789440e+07	0.062281	0.062823	0.060657

SETTING UP THE ML MODEL:

We're using **classification algorithm** because our target is binary (0/1). A **1** means the price went up, a **0** means it went down. If we had a different type of target (like the actual price), we might use a regression algorithm.

When we initialized the model, we passed in a few parameters:

- n_estimators - this is the number of individual decision trees that the algorithm should create. A random forest is an ensemble of decision trees that is more robust to overfitting than an individual tree. The more trees, the more robust the algorithm, but fewer trees means it runs faster.
- min_samples_split - this is the minimum number of samples any decision tree should split on. The lower this is, the more prone the trees are to overfitting. Setting it higher also makes it run faster.
- random_state - this is nice to set so that running the algorithm twice over the same data returns the same results.

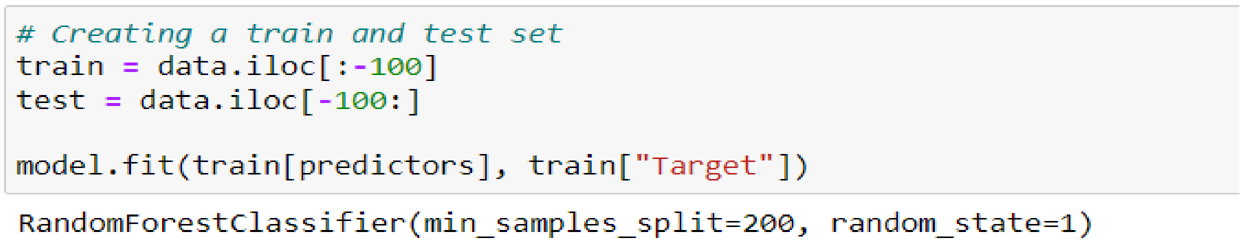


```python
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Create a random forest classification model.  Set min_samples_split high to ensure we don't overfit.
model = RandomForestClassifier(n_estimators=100, min_samples_split=200, random_state=1)
```

TRAINING THE MODEL:

Once we've set up the model, we can train it on the last 100 rows of the dataset. We're using all of the data except the last 100 rows to predict the last 100 rows.

The fit method will train the model using our **predictors** to predict the Target.



```python
# Creating a train and test set
train = data.iloc[:-100]
test = data.iloc[-100:]

model.fit(train[predictors], train["Target"])
```

```
RandomForestClassifier(min_samples_split=200, random_state=1)
```

MEASURING ERROR:

Next, we checked how accurate the model was. Earlier, we mentioned using precision to measure error. We can do this by using the precision_score function from scikit-learn.

Precision will tell us on what % of days that the algorithm said the price would go up it actually went up. Because we want to minimize risk, we want to have a high precision. This means that when we buy stock, we have high confidence that we'll make money.

Below, we did the following to calculate precision:

- Import the scikit-learn precision_score function
- Generate predictions from our model using the predict function. This will give us a 0 or a 1 for each row.
- The predict method returns a numpy array. To make it easier to work with, we've turned it into a pandas Series.
- Finally, we execute the precision_score function, passing in the predictions and the target.

```python
from sklearn.metrics import precision_score

# Evaluate error of predictions
preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test["Target"], preds)
```

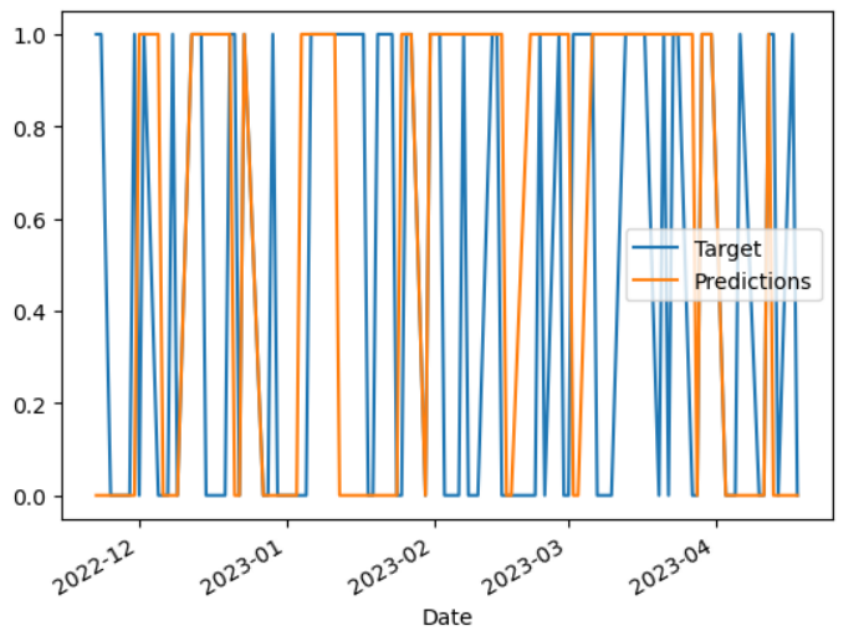
```
0.559322033898305
```

Our model is directionally accurate **55.9%** of the time.We can take a deeper look at the individual predictions and the actuals.We did this by plotting the target against the predictions by first combining the Target column and Predictions column into one DataFrame using the concat function. This function joins two pandas objects. In this case, each object is a series, so we join them on axis=1, so each one becomes a column.

Then, we plot the DataFrame to visualize both columns.

```python
combined = pd.concat({"Target": test["Target"],"Predictions": preds}, axis=1)
combined.plot()
```

```
<AxesSubplot:xlabel='Date'>
```



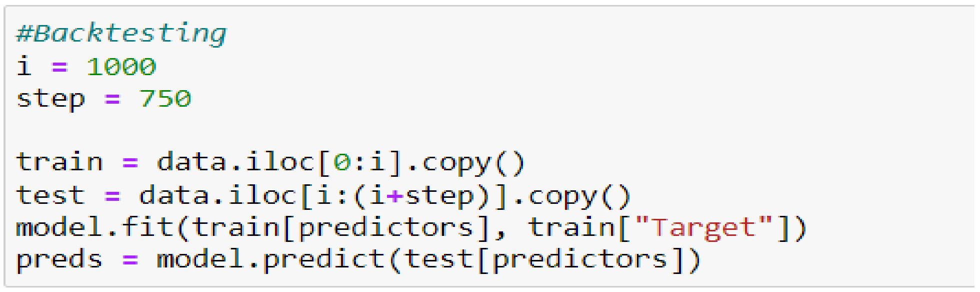
BACKTESTING:

Backtesting ensures that we only use data from before the day that we're predicting. If we use data from after the day we're predicting, the algorithm is unrealistic.

Before we write our full backtesting loop, let's write the code for a single iteration. In the below code:

- We'll take the first 1000 rows of the data as our training set
- We'll take the next 750 rows as our testing set
- We'll fit our machine learning model to the training set
- We'll make predictions on the test set

This is similar to what we did before, but we changed which rows we're looking at:



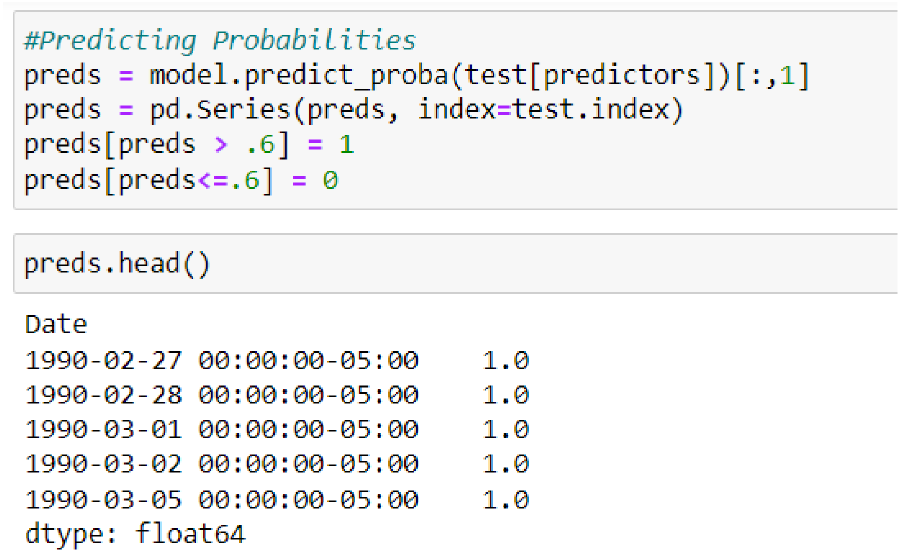
```
#Backtesting
i = 1000
step = 750

train = data.iloc[0:i].copy()
test = data.iloc[i:(i+step)].copy()
model.fit(train[predictors], train["Target"])
preds = model.predict(test[predictors])
```

PREDICTING PROBABILITIES:

In the code below, we:

- Use the predict_proba method to get the probabilities that the price will go up instead of just 0/1.
- Take the second column of the result, to ensure that we only get the probability that the price will go up.
- convert the result from a numpy array to a pandas Series to make it easier to work with.
- If the % chance that the price will go up is >60%, then say the price will go up.

As you can see, preds gives us the prediction of the model on each day.



```python
#Predicting Probabilities
preds = model.predict_proba(test[predictors])[:,1]
preds = pd.Series(preds, index=test.index)
preds[preds > .6] = 1
preds[preds<=.6] = 0
```

```python
preds.head()
```

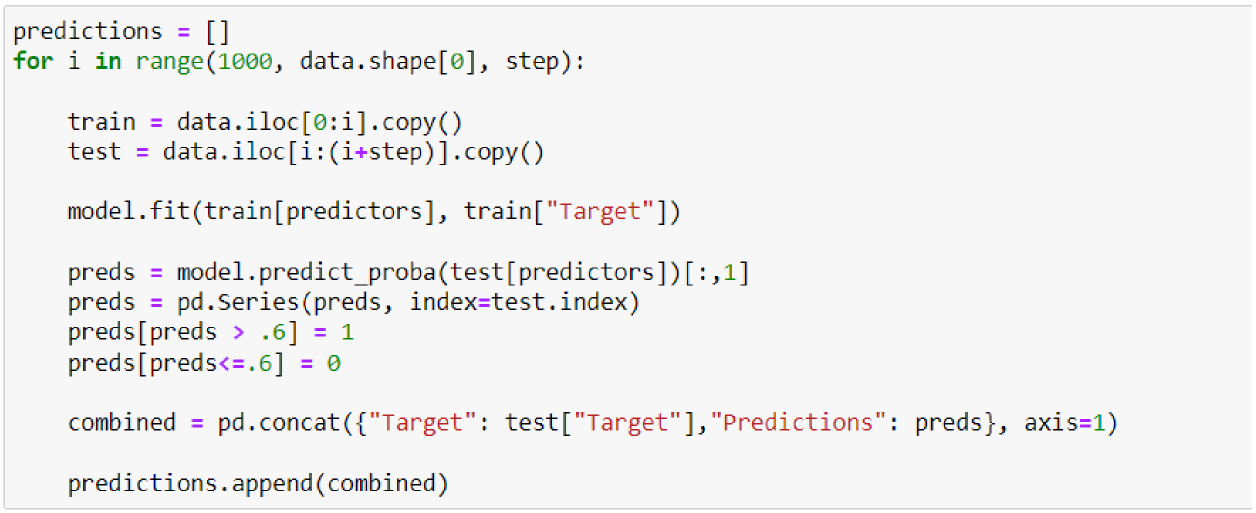
```
Date
1990-02-27 00:00:00-05:00    1.0
1990-02-28 00:00:00-05:00    1.0
1990-03-01 00:00:00-05:00    1.0
1990-03-02 00:00:00-05:00    1.0
1990-03-05 00:00:00-05:00    1.0
dtype: float64
```

PULLING IT INTO A LOOP:

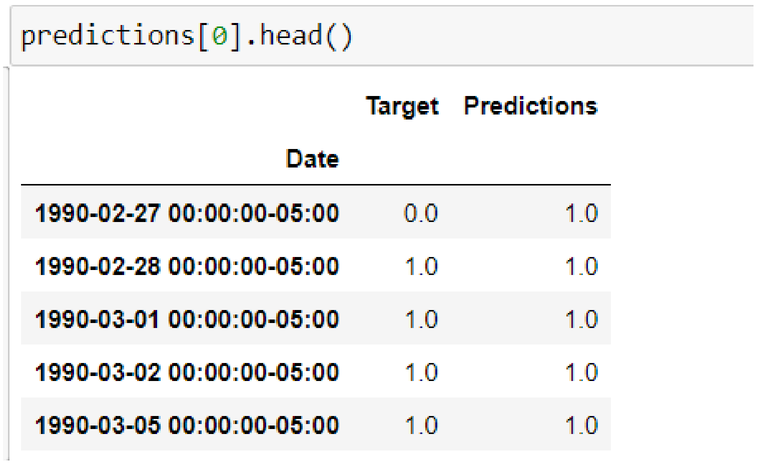
Next, we pull together everything we have done into a loop. This loop will enable us to iterate across the entire dataset, generate predictions, and add them into the predictions list.

In this loop, we:

- Generate predictions on our train and test set
- Combine those predictions with the actual target
- Add all the predictions into one list



```python
predictions = []
for i in range(1000, data.shape[0], step):

    train = data.iloc[0:i].copy()
    test = data.iloc[i:(i+step)].copy()

    model.fit(train[predictors], train["Target"])

    preds = model.predict_proba(test[predictors])[:,1]
    preds = pd.Series(preds, index=test.index)
    preds[preds > .6] = 1
    preds[preds<=.6] = 0

    combined = pd.concat({"Target": test["Target"],"Predictions": preds}, axis=1)

    predictions.append(combined)
```



```python
predictions[0].head()
```

	Target	Predictions
Date		
1990-02-27 00:00:00-05:00	0.0	1.0
1990-02-28 00:00:00-05:00	1.0	1.0
1990-03-01 00:00:00-05:00	1.0	1.0
1990-03-02 00:00:00-05:00	1.0	1.0
1990-03-05 00:00:00-05:00	1.0	1.0

CREATING A BACKTESTING FUNCTION:

In the backtesting function, we:

- Split the training and test data
- Train a model
- Make predictions on the test data using predict_proba
- Combine our predictions with the actual Target, so we can calculate error easily
- Return all of the predictions

This function enables us to create predictions across our whole dataset whenever we want.

```python
def backtest(data, model, predictors, start=1000, step=750):
    predictions = []
    for i in range(start, data.shape[0], step):

        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()

        model.fit(train[predictors], train["Target"])

        preds = model.predict_proba(test[predictors])[:,1]
        preds = pd.Series(preds, index=test.index)
        preds[preds > .6] = 1
        preds[preds<=.6] = 0

        combined = pd.concat({"Target": test["Target"],"Predictions": preds}, axis=1)
        predictions.append(combined)

    return pd.concat(predictions)
```

<u>RUNNING THE FUNCTION:</u>

```python
#Running Backtesting Function
predictions = backtest(data, model, predictors)
```

Next, we used the **value_counts** method to identify how many times the algorithm predicted the price would go up versus down.

```python
predictions["Predictions"].value_counts()
```

```
0.0    7634
1.0     715
Name: Predictions, dtype: int64
```

```python
predictions["Target"].value_counts()
```

```
1.0    4223
0.0    4126
Name: Target, dtype: int64
```

Finally, let's look at precision. We use the same function from before to evaluate this.

It looks like we have a low precision as well. This is because the algorithm has limited predictors to help it determine if the price will go up or down. We've added some more predictors next to help the algorithm make better decisions.

```
precision_score(predictions["Target"], predictions["Predictions"])
```

```
0.5034965034965035
```

ROLLING MEANS:

Rolling means are useful because they can help the algorithm evaluate the current price against the average price this week, quarter, or year. For example, if the price now is higher than the annual price, it could mean that the stock is on an upward trend.

To calculate the rolling averages, we again used the pandas rolling method to find the rolling mean of the Close column for different time horizons.

```
weekly_mean = data.rolling(7).mean()["Close"]
quarterly_mean = data.rolling(90).mean()["Close"]
annual_mean = data.rolling(365).mean()["Close"]
```

We can also tell the algorithm how many days in the last week the price has gone up. We can do this by using the pandas shift and rolling methods:

We shifted the data forward, so we're not incorporating information from the current day into our predictor. If we didn't use shift, then the algorithm would have knowledge of the actual target. Finding the 7-day rolling sum of the target, if the price went up for all 7 days, this would be 7 and if it went up 0 days, this would be 0.

```
weekly_trend = data.shift(1).rolling(7).sum()["Target"]
```

Now, we're ready to add our ratios into our predictor DataFrame.

First, we add the ratios between the weekly, quarterly, and annual means to the close:

```
data["weekly_mean"] = weekly_mean / data["Close"]
data["quarterly_mean"] = quarterly_mean / data["Close"]
data["annual_mean"] = annual_mean / data["Close"]
```

Next, we add in the ratios between different rolling means. This helps the algorithm understand what the weekly trend is relative to the annual trend.

```
data["annual_weekly_mean"] = data["annual_mean"] / data["weekly_mean"]
data["annual_quarterly_mean"] = data["annual_mean"] / data["quarterly_mean"]
```

Next, we add our weekly trend into the predictor DataFrame.

```
data["weekly_trend"] = weekly_trend
```

Then, we add some ratios between intraday open, low, and high prices and the close price. This helps the algorithm understand what the price trend was during the last day. For example, if the high was much higher than the close price, it may mean that the stock was on a downward trend at the end of the day.

```
data["open_close_ratio"] = data["Open"] / data["Close"]
data["high_close_ratio"] = data["High"] / data["Close"]
data["low_close_ratio"] = data["Low"] / data["Close"]
```

Finally, we update our predictors list with all of the new predictors we added. This ensures that we use all of our new predictors when we're training the model.

```
full_predictors = predictors + ["weekly_mean", "quarterly_mean",
"annual_mean", "annual_weekly_mean", "annual_quarterly_mean",
"open_close_ratio", "high_close_ratio", "low_close_ratio"]
```

UPDATING OUR PREDICTIONS:

We then updated our predictions with our new predictors. We had to cut off the first 365 rows because our rolling means had NaN values for the first 365 rows. We want to make sure we have consistent data for every row that we make predictions for.

As we can see, our predictions are more accurate than before!

```
predictions = backtest(data.iloc[365:], model, full_predictors)
```

```
precision_score(predictions["Target"], predictions["Predictions"])
```

```
0.5686900958466453
```

EVALUATING OUR PREDICTIONS:

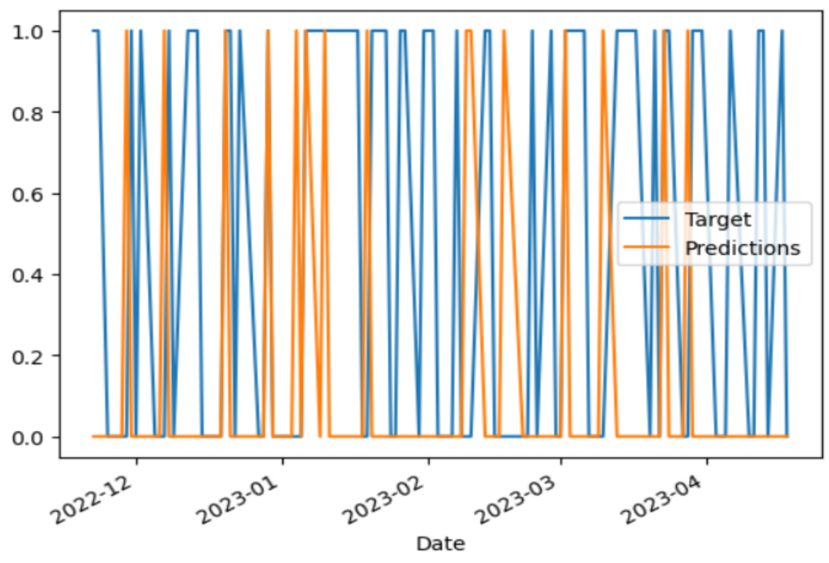
We checked how many trades we would have made. We can again do this with value_counts.

```
# Shows how many trades we would make

predictions["Predictions"].value_counts()
```

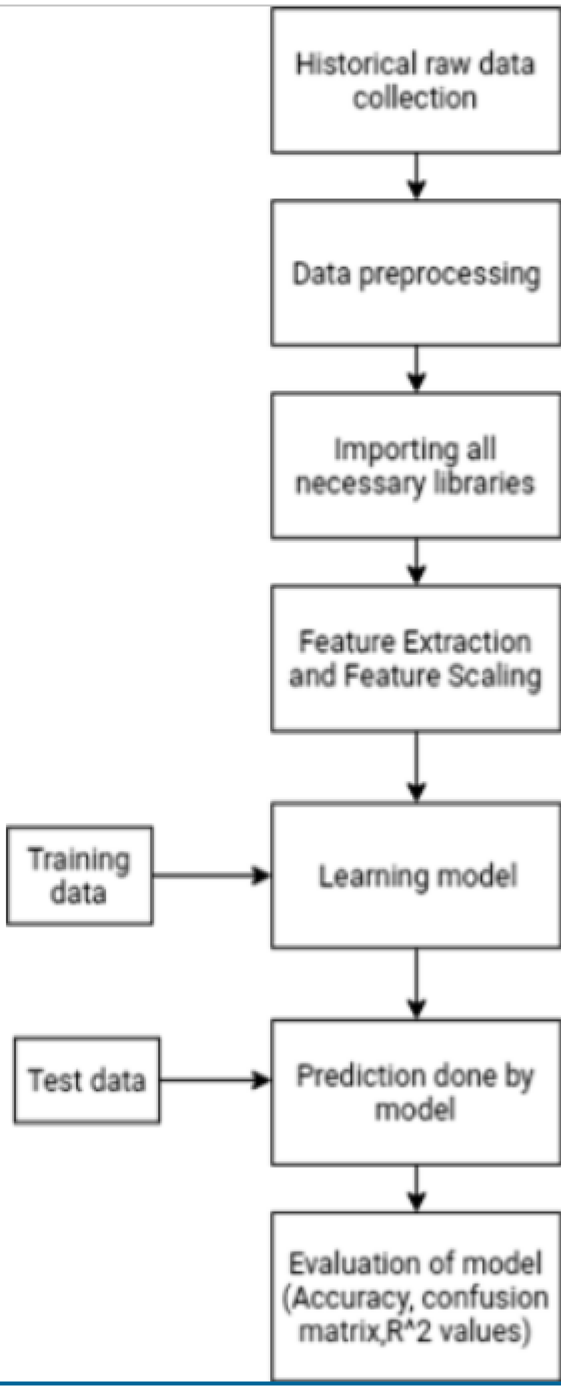
```
0.0    7671
1.0     313
Name: Predictions, dtype: int64
```

```
# Trades made in the last 100 days
predictions.iloc[-100:].plot()
```

```
<AxesSubplot:xlabel='Date'>
```



FLOWCHART


Historical raw data collection
Data preprocessing
Importing all necessary libraries
Feature Extraction and Feature Scaling
Training data
Learning model
Test data
Prediction done by model
Evaluation of model (Accuracy, confusion matrix,R^2 values)

RESULT ANALYSIS AND FUTURE SCOPE

The input dataset obtained from Yahoo Finance was efficiently preprocessed and various significant attributes were added to the dataset like momentum, volatility and sector details. Learning and understanding the various terminologies and techniques present in the stock market was very helpful in preprocessing the dataset in order to achieve best possible results.

Various neural network techniques can be applied to the processed data to make the machine more powerful. Advanced neural network techniques can be applied to the processed dataset. The neural network techniques make use of time series to accurately predict the stock values. Various other features like asset value, equity ratio, etc. can be taken into consideration to further improve the accuracy. Related tweets from twitter can also be considered while predicting the future stock prices.

The methods mentioned in the paper can be applied to real time data to get real time predictions of the stock value. Thus it can be deployed in real world applications providing a more reliable way of understanding stock price changes. Accurate graphs can be plotted for a particular company to understand the patterns of stock values and thus making it easy to understand specific patterns.