Microsoft Stock Prices Prediction using Machine Learning

Bhuvaneswari Gnanasekar, Prateek Kammili, Sasweth Rajanarayanan, Vinay Shanmukh Aradhya
Group - H

Abstract—**Stock prediction has been one of the most intricate learning problems in the statictics and Machine Learning space. This is because there is no one single best way to do it and the factors affecting stock prices cannot be determined with a good confidence. Any unforeseen situation could have affected the stock prices of a company and modeling datasets with variables that cannot be fathomed, is indeed what makes Stock prediction difficult. In this work, we attempt to model a dataset of stock prices of Microsoft, augmented with some common features important for stock prices, such as Inflation rate, unemployment rate and the Gross Domestic Product, in an attempt to perform stock price prediction post a 28 day period. We compare classification models that model the dataset to classify an investment decision for a given day as a profit or loss, and regression models that predict the returns, post a 28 day (20 working day) period. The models are compared mutually and the best one of them is used for the calculation of the income returned if the model is used as the basis for investment.**

I. Introduction

This work aims at predicting the stock prices of Microsoft Corporation (”MSFT”) and also classifying the investment as profitable or not, post a 28-day period, while learning from the past 5 years' stock prices pertaining to the company. We research and experiment on several factors that potentially affect stock prices, such as the inflation rate for the 5-year span, the unemployment rate, and the Gross Domestic Product(GDP), and attempt various feature engineering practices to improve the model's performance. We attempt a total of 3 regression models - a Linear Regression model, a Multi Layer Perceptron with regression loss, and an ensemble regressor, and a total of 3 classification models - a Logistic Regression model, a Multi Layer Perceptron with classification loss, and an ensemble classifier, attempting to predict the stock price and classifying the investment as a profit or loss respectively, post a 28-day period. We also compare the performances of the models mutually on unseen data. Further, we also calculate the income returned post a 20 working day period, when the best model out from all the comparisons in this work is used to make a purchase decision. The reasoning behind why a single category of models perform well on the datasets is also attempted throughout the work and in the discussions section. This work is divided as follows: Section II discusses the dataset used in this work and feature engineering, section III deals with Modelling and Results pertaining to classification, while section IV does the same for regression. Section V and VI deal with the Income calculation with the best model and discussions respectively, leaving Conclusions for Section VII. Table 1 shows the particulars of the datasets used for this work and has all been collected on 2022-04-26.

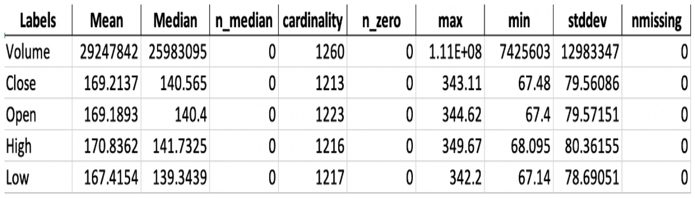
Dataset	Description	Date
Microsoft Stock Prices	Stock Prices of MSFT https://www.nasdaq.com/market-activity/stocks/msft/historical	2022-04-26
Inflation Rate	Inflation rate day-wise https://fred.stlouisfed.org/series/T5YIE	2022-04-26
GDP	Gross Domestic Product - Quarterly https://fred.stlouisfed.org/series/GDP	2022-04-26
Unemployment Rate	Unemployment rate month-wise https://fred.stlouisfed.org/series/ UNRATE	2022-04-26

Table I
Dataset Descriptions, Sampling Date: ’2022-04-26’

II. Dataset and Feature Engineering

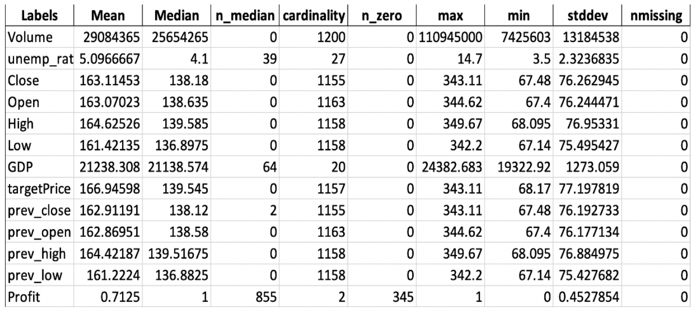
The datasets that have been used in the work are shown in table 1. The dataset for Microsoft's stock prices hosted for a span of 5 years, the date of each working day from 2017 to 2022, the volume of the stock, the closing, opening, high and low prices of the stock on that day. The dataset for inflation hosted for each day, for a span of 5 years, the inflation rate. These also had days on which the stock prices were unavailable (the non-working days), so those examples were eliminated during data preprocessing. The unemployment rate dataset hosted for the past 5 years, the monthly unemployment rate. These values were interpolated to daily unemployment rate data to match the size of the stocks dataset. The GDP dataset hosted for the past 5 years, the quarterly GDP. These values were interpolated to daily GDP data to match the size of the stocks dataset.

The exploratory analysis of the principle dataset - the stock prices dataset revealed interesting insights on the data. The data quality report of this principle dataset is shown in fig. 1. The unemployment rate, inflation rate and the GDP are added to the principle dataset, which is used for modelling. This work tests the potential of multiple versions of this dataset to predict stock prices post a 28 day period and classifying the investing decision on a given day as a profit or loss, post a 28-day period. Besides daily unemployment rate, inflation rate and the Gross Domestic Product, the previous days' closing price, opening price, high and low prices of the stock were also added as features to the dataset. The data quality report of the final dataset has been shown in fig. 2. The final dataset hosted a total of 1200 rows and 13 columns.



Labels	Mean	Median	n_median	cardinality	n_zero	max	min	stddev	nmissing
Volume	29247842	25983095	0	1260	0	1.11E+08	7425603	12983347	0
Close	169.2137	140.565	0	1213	0	343.11	67.48	79.56086	0
Open	169.1893	140.4	0	1223	0	344.62	67.4	79.57151	0
High	170.8362	141.7325	0	1216	0	349.67	68.095	80.36155	0
Low	167.4154	139.3439	0	1217	0	342.2	67.14	78.69051	0

Figure 1. DQR - Raw Dataset



Labels	Mean	Median	n_median	cardinality	n_zero	max	min	stddev	nmissing
Volume	29084365	25654265	0	1200	0	110945000	7425603	13184538	0
unemp_rat	5.0966667	4.1	39	27	0	14.7	3.5	2.3236835	0
Close	163.11453	138.18	0	1155	0	343.11	67.48	76.262945	0
Open	163.07023	138.635	0	1163	0	344.62	67.4	76.244471	0
High	164.62526	139.585	0	1158	0	349.67	68.095	76.95331	0
Low	161.42135	136.8975	0	1158	0	342.2	67.14	75.495427	0
GDP	21238.308	21138.574	64	20	0	24382.683	19322.92	1273.059	0
targetPrice	166.94598	139.545	0	1157	0	343.11	68.17	77.197819	0
prev_close	162.91191	138.12	2	1155	0	343.11	67.48	76.192733	0
prev_open	162.86951	138.58	0	1163	0	344.62	67.4	76.177134	0
prev_high	164.42187	139.51675	0	1158	0	349.67	68.095	76.884975	0
prev_low	161.2224	136.8825	0	1158	0	342.2	67.14	75.427682	0
Profit	0.7125	1	855	2	345	1	0	0.4527854	0

Figure 2. DQR for the Final Dataset

In this work, we conceive multiple ways to generate the target variable, both for regression and classification. The variable 'targetPrice' serves as the target variable for our regression task and the variable 'Profit' serves as the target variable for our classification task. To generate these variables, we identify different approaches. First is to use the lowest price of the stock as the investment price and the highest price on the 20th working day from the investment day as the returns/target price. This is however, not very practical. Second, we use the closing price of a stock as the investment and the closing price on the 20th working day as the returns/target price. This is more practical than approach 1. The third approach is to repeat step 2 with feature stacking from the previous day, i.e., we stack the closing, opening, high and low prices from the previous days also as features, while still using the current day's closing price as the investment and the closing price on the 20th day as the returns/target price. For each of these datasets, we create a 'Profit' column that represents the investment decision for that day - to invest or not, which is decided based on a comparison between the target price and the lowest price for the first dataset and the target price and the closing price for the second and third datasets.

III. Modeling and Results - Classification

The classification problem considers the 'Profit' column as the target variable and predicts if the investment on a day would be profitable or not, post a 20 working day period. In this work, we attempt a comparison of the performance of three different classification models on the three datasets. The comparison is both vertical and horizontal - vertical because we compare the models mutually on the datasets and horizontal because we compare each model's performance on the three datasets.

A. *Logistic Regression*

The Logistic Regression model is a classification model that is popularly used in binary classification tasks. We train a logistic regression model on the datasets and compare them mutually. The datasets are split into train and test datasets with a 70-30 train-test split. The train and test sets are min-max normalized prior to being used for modeling. The training and test accuracies of the model was decent, with the model showing a decent generalization towards unseen data. Table 2 shows the training and test accuracies of the Logistic Regression model.

Dataset	Train Accuracy	Test Accuracy
Dataset 1	82.62%	82.82%
Dataset 2	73.45%	73.96%
Dataset 3	74.07%	74.57%

Table II
TRUE POSITIVE RATES FOR VARYING TREE DEPTHS FOR THE ENTROPY CRITERION.

The train-test accuracies from table 2 show that the performance on the dataset 1 is the best. However, it is to be noted that the way the target variable 'Profit' is generated for Dataset 1 is unrealistic, as no one would know if the amount that he/she is investing is the lowest for the day. Dataset 3, thus shows the performance of the model on realistically engineered data. It is to be noted that Dataset 3 is the feature stacked version of Dataset 2 and shows that feature stacking indeed results in a better model performance. The reason why we still consider dataset 1 is that it gives an advantage in comparisons. Going forward, we actually see the same set of models performing better on dataset 3 than dataset 1.

B. *MultiLayer Perceptrons*

In this work, we experiment on different configurations of a Multi Layer Perceptron (MLP), with respect to the number of hidden layers, the number of nodes in each of the hidden layers and the type of activation function used in the hidden layers, all with early stopping enabled, with tolerance of 1e-4, and a learning rate of 0.001, a regularization penalty of 0.001, and using the Stochastic Gradient Descent optimizer. The experimental results for datasets 1, 2, and 3, are shown in tables 3, 4, and 5, while those for the datasets 1, 2, and 3 with polynomial features of degree 2 are shown in tables 6, 7 and 8.

Hidden Layer	Activation	Test Accuracy	AUROC
(4, 4)	identity	71.6%	0.5
(4, 4)	logistic	71.6%	0.5
(4, 4)	tanh	28.3%	0.5
(4, 4)	relu	71.6%	0.5
(1, 2, 6, 9)	identity	71.6%	0.5
(1, 2, 6, 9)	logistic	71.6%	0.5
(1, 2, 6, 9)	tanh	71.6%	0.5
(1, 2, 6, 9)	relu	71.6%	0.5
(5, 5, 5)	tanh	72.2%	0.53
(5, 5, 5)	identity	71.3%	0.52

Table III
PERFORMANCE OF SEVERAL(SELECTED) MLPS ON DATASET 1. HIDDEN LAYER COLUMN REFERS TO THE CONFIGURATION OF THE HIDDEN LAYER. (2, 4) MEANS THAT THERE ARE TWO HIDDEN LAYERS, WITH 2 NEURONS IN THE FIRST AND 4 NEURONS IN THE SECOND.

Hidden Layer	Activation	Test Accuracy	AUROC
(4, 4)	identity	72.02	0.49
(4, 4)	logistic	72.3	0.5
(4, 4)	tanh	72.3	0.5
(4, 4)	relu	27.7	0.5
(1, 2, 6, 9)	identity	72.3	0.5
(1, 2, 6, 9)	tanh	72.3	0.5
(1, 2, 6, 9)	relu	72.3	0.5
(1, 2, 6, 9)	logistic	72.3	0.5
(5, 5, 5)	tanh	72.3	0.5
(5, 5, 5)	identity	72.3	0.5

Table IV
PERFORMANCE OF SEVERAL(SELECTED) MLPS ON DATASET 2. HIDDEN LAYER COLUMN REFERS TO THE CONFIGURATION OF THE HIDDEN LAYER. (2, 4) MEANS THAT THERE ARE TWO HIDDEN LAYERS, WITH 2 NEURONS IN THE FIRST AND 4 NEURONS IN THE SECOND.

Hidden Layer	Activation	Test Accuracy	AUROC
(4, 4)	identity	71.6	0.49
(4, 4)	logistic	71.6	0.5
(4, 4)	tanh	71.4	0.49
(4, 4)	relu	71.6	0.5
(1, 2, 6, 9)	identity	71.6	0.5
(1, 2, 6, 9)	tanh	71.6	0.5
(1, 2, 6, 9)	relu	71.6	0.5
(1, 2, 6, 9)	logistic	71.6	0.5
(5, 5, 5)	tanh	71.6	0.5
(5, 5, 5)	identity	71.6	0.5

Table V
PERFORMANCE OF SEVERAL(SELECTED) MLPS ON DATASET 3. HIDDEN LAYER COLUMN REFERS TO THE CONFIGURATION OF THE HIDDEN LAYER. (2, 4) MEANS THAT THERE ARE TWO HIDDEN LAYERS, WITH 2 NEURONS IN THE FIRST AND 4 NEURONS IN THE SECOND.

Hidden Layer	Activation	Test Accuracy	AUROC
(4, 4)	identity	71.6%	0.5
(4, 4)	logistic	28.3%	0.5
(4, 4)	tanh	28.3%	0.5
(4, 4)	relu	28.3 %	0.5
(1, 2, 6, 9)	identity	71.6%	0.5
(1, 2, 6, 9)	tanh	71.6%	0.5
(1, 2, 6, 9)	relu	71.9%	0.54
(1, 2, 6, 9)	logistic	71.6%	0.5
(5, 5, 5)	tanh	71.6%	0.503
(5, 5, 5)	identity	71.9%	0.505

Table VI
PERFORMANCE OF SEVERAL(SELECTED) MLPS ON DATASET 1 WITH POLYNOMIAL FEATURES. HIDDEN LAYER COLUMN REFERS TO THE CONFIGURATION OF THE HIDDEN LAYER. (2, 4) MEANS THAT THERE ARE TWO HIDDEN LAYERS, WITH 2 NEURONS IN THE FIRST AND 4 NEURONS IN THE SECOND.

Hidden Layer	Activation	Test Accuracy	AUROC
(4, 4)	identity	73.4%	0.52
(4, 4)	logistic	27.7%	0.5
(4, 4)	tanh	72.3%	0.5
(4, 4)	relu	72.3 %	0.52
(1, 2, 6, 9)	identity	72.3%	0.5
(1, 2, 6, 9)	tanh	72.3%	0.5
(1, 2, 6, 9)	relu	72.3%	0.5
(1, 2, 6, 9)	logistic	72.3%	0.5
(5, 5, 5)	tanh	72.3%	0.5
(5, 5, 5)	identity	72.02%	0.501

Table VII
PERFORMANCE OF SEVERAL(SELECTED) MLPS ON DATASET 2 WITH POLYNOMIAL FEATURES. HIDDEN LAYER COLUMN REFERS TO THE CONFIGURATION OF THE HIDDEN LAYER. (2, 4) MEANS THAT THERE ARE TWO HIDDEN LAYERS, WITH 2 NEURONS IN THE FIRST AND 4 NEURONS IN THE SECOND.

Hidden Layer	Activation	Test Accuracy	AUROC
(4, 4)	identity	71.6%	0.5
(4, 4)	logistic	71.6%	0.5
(4, 4)	tanh	71.6%	0.5
(4, 4)	relu	71.6%	0.5
(1, 2, 6, 9)	identity	71.6%	0.5
(1, 2, 6, 9)	tanh	71.6%	0.5
(1, 2, 6, 9)	relu	71.6%	0.5
(1, 2, 6, 9)	logistic	28.3%	0.5
(5, 5, 5)	tanh	71.6%	0.5
(5, 5, 5)	identity	71.6%	0.5

Table VIII
PERFORMANCE OF SEVERAL(SELECTED) MLPS ON DATASET 3 WITH POLYNOMIAL FEATURES. HIDDEN LAYER COLUMN REFERS TO THE CONFIGURATION OF THE HIDDEN LAYER. (2, 4) MEANS THAT THERE ARE TWO HIDDEN LAYERS, WITH 2 NEURONS IN THE FIRST AND 4 NEURONS IN THE SECOND.

The comparisons show that the performance of the MLP with 2 hidden layers with 4 neurons in each of the layers, with the identity activation function modeling dataset 3 with polynomial features, shows the best generalization and AUROC combination.

C. Ensemble Classifiers

In this work, we also push ensemble classifiers to the maximum extent on the datasets 2 and 3, which we believe are more representative of realism. We attempt a performance comparison between a Random Forest Classifier with several values for the number of estimators, a Bagging Classifier with Decision Tree as the base estimator and with several values for the number of decision trees participating in modeling the dataset, an AdaBoost Classifier with Decision Tree as the base estimator and with several values for the number of decision trees participating in modeling the dataset, a Gradient Boosting Classifier, with several values for the number of estimators participating in modeling the dataset, an XGBoost Classifier, with several values for the number of estimators and a Voting Classifier consisting of a Random Forest Classifier with several values for the number of decision trees, a Bagging Classifier and a Logistic Regressor. Tables IX - XIV show the performance of the aforementioned classifiers on dataset 2, and dataset 3. From the tables, it can be seen that the best performing model in terms of generalization towards unseen data, is the Bagging Classifier with a total of 10 estimators, boasting a test accuracy of 85.75%.

Number of Estimators in the Random Forest	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
10	84.5%	83.5%
20	84.5%	82%
40	84.75%	83%
60	83.25%	83.25%
70	84%	84.25%
100	84.25%	84%

Table IX
PERFORMANCE OF SEVERAL(SELECTED) VOTING CLASSIFIERS ON DATASET 2 AND 3.

Number of Estimators	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
10	83.25%	85.75%
20	82.75%	83.75%
40	85%	84.5%
60	84.75%	83.75%
70	84.75%	84%
100	85.5%	85%

Table X
PERFORMANCE OF SEVERAL(SELECTED) BAGGING CLASSIFIERS ON DATASET 2 AND 3.

Number of Estimators	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
10	84.75%	81.75%
20	83%	80.75%
40	83%	82%
60	83.75%	82.25%
70	84.25%	83.25%
100	83.75%	83.5%

Table XI
PERFORMANCE OF SEVERAL(SELECTED) RANDOM FOREST CLASSIFIERS ON DATASET 2 AND 3.

Number of Estimators	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
10	81.75%	81.5%
20	81.25%	81.25%
40	82%	81.75%
60	81.25%	82.75%
70	80.75%	80.75%
100	81%	82.25%

Table XII
PERFORMANCE OF SEVERAL(SELECTED) ADABOOST CLASSIFIERS ON DATASET 2 AND 3.

Number of Estimators	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
10	74%	74.25%
20	74.25%	74.75%
40	76%	78.75%
60	80.5%	79.25%
70	82%	80%
100	82%	80.25%

Table XIII
PERFORMANCE OF SEVERAL(SELECTED) GRADIENT BOOSTING CLASSIFIERS ON DATASET 2 AND 3.

Number of Estimators	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
10	83%	83.5%
20	85.25%	83.25%
40	83.5%	84.75%
60	82.5%	84.75%
70	83.25%	84.5%
100	84.25%	83.5%

Table XIV
PERFORMANCE OF SEVERAL(SELECTED) XGBOOST CLASSIFIERS ON DATASET 2 AND 3.

IV. REGRESSION

This work compares the performance of four linear regression models on the datasets 2 and 3 - a Linear Regression model, a RidgeCV regression model, an MLP, and an Ensemble Regression model.

A. Linear Regression

The linear regression model in this work, was expected to show the best results, as practically, Microsoft's stock prices have always seen a net increase since 2017 and has not had any sharp dips. This means that the feature and target variables would also have a linear relationship that is increasing. So, a linear regression model is expected to perform better.The tables below show the performances of the linear regression model on the training and test sets, in terms of the R2 score and Mean Square Error. The target variable in this task is the 'targetPrice' variable.The tables XV and XVI below shows the performance of Linear and RidgeCV regression models on datasets 1, 2, and 3. The tables show that Linear and RidgeCV regression models perform equally well and the best on unseen data. They offer great generalization capabilities.

Dataset	R2 Train	R2 Test	MSE Train	MSE Test
1	0.9832	0.9836	0.001372	0.001213
2	0.9819	0.9801	0.001421	0.001575
3	0.9821	0.9802	0.001407	0.001569

Table XV
PERFORMANCE OF LINEAR REGRESSION MODELS ON DATASET 1, 2, AND 3.

Dataset	R2 Train	R2 Test	MSE Train	MSE Test
1	0.9832	0.9836	0.001372	0.001213
2	0.9819	0.9802	0.001419	0.001569
3	0.9821	0.9802	0.00147	0.001569

Table XVI
PERFORMANCE OF RIDGECV REGRESSION MODELS ON DATASET 1, 2, AND 3.

B. MultiLayer Perceptrons

In this work, we also attempt a performance comparison of several configurations of multilayer perceptrons, each using the relu activation function and the Adam optimizer, while using a regularization penalty of 0.001. Table XVII shows the performance of several configurations of MLPs on the train and test sets of datasets 2 and 3. The table shows that the deeper models perform better on unseen data and offer great generalization. The performance on dataset 3 is better than that in 2 for most of the configurations.

Hidden Layer	Test Accuracy (Dataset 2)	Test Accuracy (Dataset 3)
(100)	78.28	93.73
(200)	94.26	97.13
(300)	95.35	97.06
(100, 200)	97.82	97.67
(200, 100)	97.82	97.51
(100, 200, 300)	97.82	97.86
(200, 300, 400)	98.06	98.08

Table XVII
THE REGRESSION PERFORMANCE OF VARIOUS MLPS.

C. Ensemble Regression

The ensemble regression models tried in this work are a Random Forest Regressor with several values for the number of estimators, a Bagging Regressor with Decision Tree as the base estimator and with several values for the number of decision trees participating in modeling the dataset, an AdaBoost Regressor with Decision Tree as the base estimator and with several values for the number of decision trees participating in modeling the dataset, a Gradient Boosting Regressor, with several values for the number of estimators participating in modeling the dataset, and a Voting Regressor with a Decision Tree Regressor and a Linear Regressor, with several values for the number of decision trees. Table XVIII shows the performance of the aforementioned regressors on dataset 3.

Ensemble Model	Number of Estimators (Best Performing)	R2 Test	MSE
Bagging Regressor	20	0.98068	0.0015
Random Forest Regressor	100	0.9942	0.0004524
Ada Boost Regressor	20	0.9803	0.00156
Gradient Boosting Regressor	100	0.99201	0.000633
Voting Classifier	20	0.991316	0.000688

Table XVIII
THE REGRESSION PERFORMANCE OF VARIOUS ENSEMBLE MODELS.

The best performing model among the ensemble regressors, is the Random Forest regressor, with a test MSE of 0.0004524 and a test R2 score of 0.9942. This again, can be attributed to the dataset being perfect for linear regression problems. This is infact, better than the linear regression model on Dataset 3. This could be because of the ensemble advantage that the model gets from multiple decision tree regressors. The number of estimators of the contestants in table XVIII are the best performing configurations in their category, with the overall best being the Random Forest Regressor with 100 decision tree regressors.

The best performing model out of all the models are the Bagging Classifier with a total of 10 estimators, boasting a test accuracy of 85.75% and the Random Forest Regressor with a total of 100 estimators, boasting a test MSE of 0.0004524 and a test R2 of 0.9942. Figures 3 and 4 show the plots for the actual and predicted prices for the full 5 years and for the past 3 months respectively.

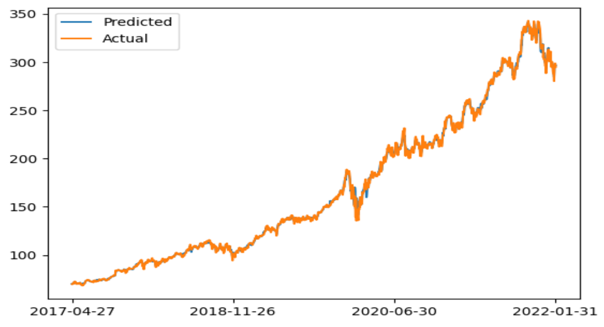


Figure 3. The plot of actual and predicted prices for the full 5 years.

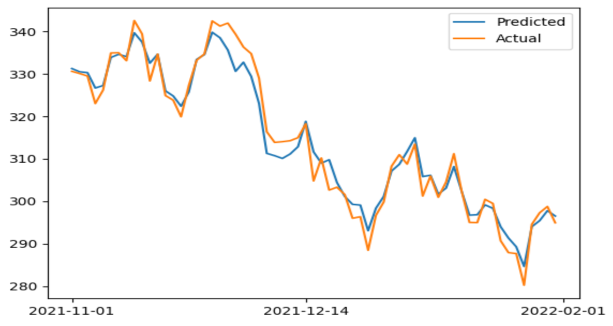


Figure 4. The plot of actual and predicted prices for the past 3 months.

The plots from fig. 3 and 4 show that the predictors and target variable being linear has helped the regression models perform well on the dataset.

V. Income Calculation

The income earn-able while using the best model - the Random Forest regressor is **12514.3**.

VI. Discussions

The models attempted in the work consisted of three classification models aiming at classifying the profit or loss of a potential investment on the stocks on a given day, after 20 working days and three regression models, each aiming to predict the returns on the 20th day since the investment. The regression models, in general, performed a lot better on the datasets because of the nature of the data itself. As can be seen from the trend of the actual prices (orange line) over time for the past 5 years, the relationship between the predictors and the target variable is almost linear. This is the benefit given by the company (Microsoft) to the problem. Microsoft's stock prices seldom saw a dip over the years and had a net increase. The same dataset, for a company that had been performing erratically on the stock market would have made the regression models struggle. The best performing model in this work - the Random Forest Regressor with 100 decision tree regressors modeling Dataset 3, performs extremely well on the test data too, as signified by a very high R2 score on the test set, which is a measure of how well the model fits the dataset, and a very low mean squared error. The computed income is a high positive value and is therefore a measure of reliability of the model to be used in Microsoft's stock prices prediction. The regression models, already show a great performance and generalization. The classification models can be improved by stacking more features from previous days and not just stacking the previous day's feature.

VII. Conclusions

This work attempted stock price prediction over a 20 working day period. Various classification models, ensemble classifiers, regression models and ensemble regressors, were all used to model the dataset with appropriate target variables. The data preprocessing stage involved the merging of different parameters that affect stock prices and the target variables were calculated using different strategies. Feature stacking of the previous day's stock prices proved to be useful and helped the models perform better. While the best classification model peaked at 85.75% accuracy on the test dataset, the regression models performed extremely well and the best of them has been used to calculate the income.

Data Preprocessing

```
import pandas as pd
import xlsxwriter
import numpy as np

df1 = pd.read_csv("stocksMid.csv")
del df1['Unnamed: 0.5']
del df1['Unnamed: 0.4']
del df1['Unnamed: 0.3']
del df1['Unnamed: 0.2']
del df1['Unnamed: 0.1']
df1['Date'] = pd.to_datetime(df1['Date'])
df = df1.sort_values(by='Date')
print(df)
open = []
close = []
high = []
low = []
# df = pd.read_excel("stocksMid.xlsx")
for i, row in df.iterrows():
   openD = str(row['Open'])
   openD = openD.split('$')[1]
   openD = float(openD)
   closeD = str(row['Close/Last'])
   closeD = closeD.split('$')[1]
   closeD = float(closeD)
   highD = str(row['High'])
   highD = highD.split('$')[1]
   highD = float(highD)
   lowD = str(row['Low'])
   lowD = lowD.split('$')[1]
   lowD = float(lowD)
   open.append(openD)
   close.append(closeD)
   high.append(highD)
   low.append(lowD)
del df['Close/Last']
del df['Open']
del df['Low']
del df['High']

data = {'Close': close, 'Open': open, 'High': high, 'Low': low}

df_new = pd.DataFrame(data)
frames = [df, df_new]
df = pd.concat(frames, axis=1)
print(df)

means = []
```

```
median = []
mode = []
cardinality = []
n_median = []
n_mode = []
n_zero = []
max = []
min = []
stddev = []
n_missing = []
labels = []
for label in df.columns:
   card = 0
   sum = (df[label].isnull()).sum()
   if sum != 0:
       card = 1
   if label not in ['Unnamed: 0', 'Date']:
       labels.append(label)
       means.append(
           df[label].mean(skipna=True))
       median.append(
           df[label].median(skipna=True))
       medVal = df[label].median(skipna=True)
       mode.append(
           df[label].mode(dropna=True))
       modeVal = df[label].mode(dropna=True)
       cardinality.append(
           len(df[label].unique()) - card)
       n_median.append(
           (df[label] == medVal).sum())
       n_mode.append(
           (df[label] == modeVal[0]).sum())
       n_zero.append(
           (df[label] == 0).sum())
       max.append(
           df[label].max())
       min.append(df[label].min())
       stddev.append(
           df[label].std())
       n_missing.append(
           (df[label].isnull()).sum()
           + (df[label] == -9999).sum())

modes = []
for i in mode:
   modes.append(i)

df9 = pd.DataFrame({'Labels':
                       labels,
                   'Mean': means,
                   'Median':
                       median,
                   'n_median':
                       n_median,
                   'cardinality':
                       cardinality,
                   'n_zero':
                       n_zero,
                   'max': max,
                   'min':
                       min,
                   'stddev':
                       stddev,
                   'nmissing':
                       n_missing})

# DQR of raw dataset stored as Excel file
df9.to_excel("StockDQRraw.xlsx")

df1 = pd.read_excel("unemprate.xlsx")

indices = df[df['Date'] > '2022-03-01'].index
# print(indices)
df.drop(indices, inplace=True)
df1.to_csv("unemprates.csv")
df1 = pd.read_csv("unemprates.csv")
dateList = df['Date']
unemp = []
for date in dateList:
   split = "" + date.split('-')[0] + "-" + date.split('-')[1]
   finalDate = split + "-01"
   urate = df1[df1['DATE'] == finalDate]
   unemp.append(float(urate['UNRATE']))
# print(unemp)
data = {'unemp_rate': unemp}
df_unemp = pd.DataFrame(data)
# print(df_unemp)
frames = [df, df_unemp]
df2 = pd.concat(frames, axis=1)
df2.to_hdf("tillUnemp.hdf", key="ML")
# print(df2)

df = pd.read_hdf("tillUnemp.hdf", key='ML')
dfInf = pd.read_csv("Inflation.csv")
dfInf['DATE'] = pd.to_datetime(dfInf['DATE'])
dfInf = dfInf.sort_values(by='DATE')
indices = dfInf[dfInf['DATE'] > '2022-03-01'].index
# print(indices)
```

```
dfInf.drop(indices, inplace=True)
print(dfInf.info())
#print(dateList)
datesList = []
for date in dateList:
   datesList.append(str(date))

dfInf.to_excel("inf.xlsx")
dfInf = pd.read_excel("inf.xlsx")
dfInf["T5YIE"] = np.where(dfInf["T5YIE"] == ".", 0.0, dfInf["T5YIE"])
inf = dfInf['T5YIE']
# print(inf1)
print(inf)

frames = [df, inf]
df2 = pd.concat(frames, axis=1)
print(df2)

df2.to_hdf("tillinf.hdf", key='ML')

df = pd.read_hdf("tillinf.hdf", key='ML')
print(df)
open = []
close = []
high = []
low = []
for i, row in df.iterrows():
   openD = str(row['Open'])
   openD = openD.split('$')[1]
   openD = float(openD)
   closeD = str(row['Close/Last'])
   closeD = closeD.split('$')[1]
   closeD = float(closeD)
   highD = str(row['High'])
   highD = highD.split('$')[1]
   highD = float(highD)
   lowD = str(row['Low'])
   lowD = lowD.split('$')[1]
   lowD = float(lowD)
   open.append(openD)
   close.append(closeD)
   high.append(highD)
   low.append(lowD)
del df['Close/Last']
del df['Open']
del df['Low']
del df['High']

data = {'Close': close, 'Open': open, 'High': high, 'Low': low}
```

```
df_new = pd.DataFrame(data)
frames = [df, df_new]
df = pd.concat(frames, axis=1)
print(df['Open'])
df.to_hdf("moneyCorrected.hdf", key='ML')

df = pd.read_hdf("moneyCorrected.hdf", key='ML')
df_gdp = pd.read_excel("GDP_final.xlsx")
gdp = df_gdp['GDP']
frames = [df, gdp]
df2 = pd.concat(frames, axis=1)
print(df2)
df2.to_hdf("tillGDP.hdf", key='ML')

df = pd.read_hdf("tillGDP.hdf", key='ML')
targetPrice = []
for i, row in df.iterrows():
   if i < 20:
       continue
   targetPrice.append(row['Close'])

for i in range(20):
   targetPrice.append(0)

data = {'targetPrice': targetPrice}
dfT = pd.DataFrame(data)
frames = [df, dfT]
df2 = pd.concat(frames, axis=1)
# print(df2)

df2 = df2[df2['targetPrice'] > 0.00]
# print(df2)
profit = []
for i, rows in df2.iterrows():
   if rows['targetPrice'] > (rows['Close']):
       profit.append(1)
   else:
       profit.append(0)
prev_close = []
prev_open = []
prev_high = []
prev_low = []

prev_c = 0
prev_o = 0
prev_h = 0
prev_l = 0
for i, rows in df2.iterrows():
   if i == 0:
       prev_c = rows['Close']
       prev_o = rows['Open']
       prev_h = rows['High']
       prev_l = rows['Low']
       prev_close.append(0)
       prev_open.append(0)
       prev_high.append(0)
       prev_low.append(0)
       continue
   prev_close.append(prev_c)
   prev_open.append(prev_o)
   prev_high.append(prev_h)
   prev_low.append(prev_l)
   prev_c = rows['Close']
   prev_o = rows['Open']
   prev_h = rows['High']
   prev_l = rows['Low']

data = {'prev_close': prev_close, 'prev_open': prev_open,
       'prev_high': prev_high, 'prev_low': prev_low,
       'Profit': profit}
dfP = pd.DataFrame(data)

frames = [df2, dfP]
df = pd.concat(frames, axis=1)
df = df[df['prev_close'] > 0]
print(df)
df.to_hdf("theFinaldf.hdf", key='ML')
df.to_excel("finalDataset.xlsx")

means = []
median = []
mode = []
cardinality = []
n_median = []
n_mode = []
n_zero = []
max = []
min = []
stddev = []
n_missing = []
labels = []
for label in df.columns:
   card = 0
   sum = (df[label].isnull()).sum()
   if sum != 0:
       card = 1
   if label not in ['Unnamed: 0', 'Date', 'T5YIE']:
       labels.append(label)
       means.append(
           df[label].mean(skipna=True))
       median.append(
           df[label].median(skipna=True))
       medVal = df[label].median(skipna=True)
       mode.append(
           df[label].mode(dropna=True))
       modeVal = df[label].mode(dropna=True)
       cardinality.append(
           len(df[label].unique()) - card)
       n_median.append(
           (df[label] == medVal).sum())
       n_mode.append(
           (df[label] == modeVal[0]).sum())
       n_zero.append(
           (df[label] == 0).sum())
       max.append(
           df[label].max())
       min.append(df[label].min())
       stddev.append(
           df[label].std())
       n_missing.append(
           (df[label].isnull()).sum()
           + (df[label] == -9999).sum())

modes = []
for i in mode:
   modes.append(i)
# print(mode)
df9 = pd.DataFrame({'Labels':
                       labels,
                   'Mean': means,
                   'Median':
                       median,
                   'n_median':
                       n_median,
                   'cardinality':
                       cardinality,
                   'n_zero':
                       n_zero,
                   'max': max,
                   'min':
                       min,
                   'stddev':
                       stddev,
                   'nmissing':
                       n_missing})
```

```
# DQR of preprocessed dataset stored as Excel file
df9.to_excel("StockDQRFinal.xlsx")

print(df)
```

Logistic Regression:

```
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.svm import SVR
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Modeling Dataset 3, which is dataset 2 with feature stacking
df = pd.read_excel("finalDataset.xlsx")
df1 = pd.read_excel("finalDataset.xlsx")
y = df['Profit']
del df['targetPrice']
# print(df)
x = df[df.columns[3:15]]
print(x)

scaler = MinMaxScaler()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
train_size=0.7, random_state=22222)
volume = x_test['Volume']
test_data = x_test

scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

clf = LogisticRegression()
clf.fit(x_train, y_train)
train = clf.score(x_train, y_train)
test = clf.score(x_test, y_test)
print(train)
print(test)
```

MLPs:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
import numpy as np
import matplotlib.pyplot as plt
from itertools import combinations
from tabulate import tabulate
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from matplotlib import pyplot as plt

"""# MLP classifier"""

def read_dataset(path):
  df=pd.read_excel(path)
  lab_encoder = preprocessing.LabelEncoder()
  df['Date'] = lab_encoder.fit_transform(df['Date'])
  X=df.copy()
  Y=X['Profit']
  X.drop(['Profit','Unnamed: 0','targetPrice','Unnamed:
0.1'],axis=1,inplace=True)

  scalerX = MinMaxScaler()
  scalerX.fit(X)
  X = scalerX.transform(X)
  return X,Y
def mlp(hl,lr,lri,activ,X,Y):
  trainX, testX, trainY, testY =train_test_split(X, Y, test_size=0.3,
random_state=24061)
  # Solve the problem using an artificial neural network
  regpenalty = 0.001
  clf = MLPClassifier(hidden_layer_sizes=hl, activation=activ, solver="sgd",
                      alpha=regpenalty, early_stopping=True, learning_rate=lr,
learning_rate_init=lri, validation_fraction=0.42)
  clf.fit(trainX,trainY)
  annPredY = clf.predict(testX)
  tn, fp, fn, tp =confusion_matrix(testY,annPredY).ravel()
  miscl=(fp+fn)/(tn+fp+fn+tp)
  accuracy=accuracy_score(testY, annPredY)
  auroc= metrics.roc_auc_score(testY, annPredY)
  return miscl,auroc,accuracy
```

```
"""# dataset 1

columns :['Date', 'Volume', 'unemp_rate', 'T5YIE', 'Close',
       'Open', 'High', 'Low', 'GDP', 'prev_close']

"""

path="/content/finalDataset.xlsx"
lrs=["adaptive"]
lris=[0.001]
activ=["identity", "logistic", "tanh", "relu"]
tolerance=1e-4
hls=[(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10),

(1,1,1),(2,2,2),(3,3,3),(4,4,4),(5,5,5),(6,6,6),(7,7,7),(8,8,8),(9,9,9),(10,10,
10)]
results_2=[]
X,Y=read_dataset(path)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)

# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_2.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_2.append([hl,lr,lri,tolerance,activation,acc,miscl,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

results_2.sort(key=lambda x: (x[2],x[3]),reverse=True)
headers3=["hidden_layer_size","actiavtion","accuracy","AUROC"]

print(tabulate(results_2[:10], headers=headers3))
print(tabulate(results_2[:100], headers=headers3))

"""# dataset 1 polynomial features

degree= 2
"""

poly_results_2=[]
X,Y=read_dataset(path)
trans = PolynomialFeatures(degree=2)
data = trans.fit_transform(X)

# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_2.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_2.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

print(tabulate(poly_results_2, headers=headers3))

poly_results_2.sort(key=lambda x: (x[2],x[3]),reverse=True)



"""# dataset 2
columns: ['Date', 'Volume', 'unemp_rate', 'T5YIE', 'Close',
       'Open', 'High', 'Low', 'GDP']

"""

path="/content/finalDataset (1).xlsx"

results_1=[]
```

```
X,Y=read_dataset(path)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation fuction
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

#results_acc_1=sorted(results_acc_1, key = lambda x: int(x[5]))
results_1.sort(key=lambda x: (x[2],x[3]),reverse=True)
#sort_AUROC=sorted(results_auroc, key = lambda x: int(x[5]))

print(tabulate(results_1[:100], headers=headers3))

"""# dataset 2 polynomial features
degree = 2
"""

#path="/content/finalDataset.xlsx"
poly_results_1=[]
X,Y=read_dataset(path)
trans = PolynomialFeatures(degree=2)
data = trans.fit_transform(X)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation function
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initial learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_1.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initain learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)


poly_results_1.sort(key=lambda x: (x[2],x[3]),reverse=True)


print(tabulate(poly_results_1[:10],headers=headers3))

print(tabulate(poly_results_1[:100],headers=headers3))

"""# dataset 3

columns: ['Date', 'Volume', 'unemp_rate', 'T5YIE', 'Close',
       'Open', 'High', 'Low', 'GDP', 'prev_close', 'prev_open',
'prev_high','prev_low']

"""

path="/content/finalDataset 2.xlsx"
results_3=[]
X,Y=read_dataset(path)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation function
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initial learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initial learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)

print(tabulate(results_3, headers=headers3))

results_3.sort(key=lambda x: (x[2],x[3]),reverse=True)

print(tabulate(results_3[:10],headers=headers3))

print(tabulate(results_3[:100],headers=headers3))

"""# dataset 3 polynomial features

degree = 2
"""

poly_results_3=[]
X,Y=read_dataset(path)
trans = PolynomialFeatures(degree=2)
data = trans.fit_transform(X)
for i in range(1,5):
  c=combinations([1, 2,3,4,5,6,7,8,9,10], i)
# different combination of hidden layer and activation function
for hl in hls:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc,=mlp(hl,lr,lri,activation,X,Y)
        poly_results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initial learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
for hl in c:
  for lr in lrs:
    for lri in lris:
      for activation in activ:
        miscl,auroc,acc=mlp(hl,lr,lri,activation,X,Y)
        poly_results_3.append([hl,activation,acc,auroc])
        print("hidden layer",hl,"learning rate",lr,"initial learning
rate",lri,"tolerance",tolerance,"activation",activation)
        print("accuracy",acc,"misclassification rate",miscl,"AUROC",auroc)
        print("*"*20)
print(tabulate(poly_results_3, headers=headers3))

poly_results_3.sort(key=lambda x: (x[2],x[3]),reverse=True)

print(tabulate(poly_results_3[:10], headers=headers3))

print(tabulate(poly_results_3[:100], headers=headers3))

"""#results"""

# result to excel
r=pd.DataFrame.from_records(results_1)
overall=pd.DataFrame.from_records(results_1,columns=['hidden_layer','activation
','ds_1_accu,','ds_1_auroc'])
r=pd.DataFrame.from_records(poly_results_1)
overall["poly_1_accu"]=list(r[2])
overall ["poly_1_auroc"]=list(r[3])
r=pd.DataFrame.from_records(results_2)
overall["ds_2_accu"]=list(r[2])
overall["ds_2_auroc"]=list(r[3])
r=pd.DataFrame.from_records(poly_results_2)
overall["poly_2_accu"]=list(r[2])
overall["poly_2_auroc"]=list(r[3])
r=pd.DataFrame.from_records(results_3)
overall["ds_3_accu"]=list(r[2])
overall["ds_3_auroc"]=list(r[3])
r=pd.DataFrame.from_records(poly_results_3)
overall["poly_3_accu"]=list(r[2])
overall["poly_3_auroc"]=list(r[3])

overall

overall.to_excel('/content/overall.xlsx')
```

"""#MLP regressor"""

```
# target price as target and performing regression on best performing dataset
def mlp(path,hl):
  df=pd.read_excel(path)
  lab_encoder = preprocessing.LabelEncoder()
  df['Date'] = lab_encoder.fit_transform(df['Date'])
  X=df.copy()
  Y=X['targetPrice']
  X.drop(['Profit','Unnamed: 0','targetPrice'],axis=1,inplace=True)

  scalerX = MinMaxScaler()
  scalerX.fit(X)
  X = scalerX.transform(X)
  trainX, testX, trainY, testY =train_test_split(X, Y, test_size=0.3,
random_state=24061)

  # Solve the problem using an artificial neural network
  regpenalty = 0.001
  clf = MLPRegressor(hidden_layer_sizes=hl, activation='relu', solver="adam",
                    alpha=regpenalty, early_stopping=True,
validation_fraction=0.35)
  clf.fit(trainX,trainY)
  sco=clf.score(testX,testY)
  pred=clf.predict(testX)
  return sco,testY,pred,testX,clf

hidden_layer=[(100,),(200,),(300,),(100,200),(200,100),(300,100),
              (100,200,300),(200,300,400)]
path1="/content/finalDataset.xlsx"
path2="/content/finalDataset 2.xlsx"
result=[]
for hl in hidden_layer:
  score,actual,pred,textX,clf=mlp(path1,hl)
  score1,actual1,pred1,textX1,clf1=mlp(path2,hl)
  result.append([hl,score,score1])
  print('hiddenlayer',hl,'activation','relu ---',score)
  print('hiddenlayer',hl,'activation','relu ---',score1)

# five year graph actual vs predicted

df1=pd.read_excel("/content/finalDataset.xlsx")
df=df1.copy()
lab_encoder = preprocessing.LabelEncoder()
df1['Date'] = lab_encoder.fit_transform(df1['Date'])
X=df1.copy()
Y=X['targetPrice']
X.drop(['Profit','Unnamed: 0','targetPrice'],axis=1,inplace=True)
scalerX = MinMaxScaler()
scalerX.fit(X)
X = scalerX.transform(X)
pred=clf.predict(X)
fig, ax = plt.subplots(figsize=(12, 6))
Y=df['targetPrice']
df['Date']  = pd.to_datetime(df['Date'])
print("dataset2")
plt.title('actual vs prediction')
plt.plot(df['Date'], Y, label = 'Actual')
plt.plot(df['Date'], pred, label = 'Prediction')
plt.legend()
plt.show()
```

```
df1=pd.read_excel("/content/finalDataset 2.xlsx")
df=df1.copy()
lab_encoder = preprocessing.LabelEncoder()
df1['Date'] = lab_encoder.fit_transform(df1['Date'])
X=df1.copy()
Y=X['targetPrice']
X.drop(['Profit','Unnamed: 0','targetPrice'],axis=1,inplace=True)
scalerX = MinMaxScaler()
scalerX.fit(X)
X = scalerX.transform(X)
pred=clf1.predict(X)

fig, ax = plt.subplots(figsize=(12, 6))
Y=df['targetPrice']
df['Date']  = pd.to_datetime(df['Date'])
print("dataset3")
plt.title('actual vs prediction')
plt.plot(df['Date'], Y, label = 'Actual')
plt.plot(df['Date'], pred, label = 'Prediction')
plt.legend()
plt.show()

# 3 month graph actual vs predicted

fig, ax = plt.subplots(figsize=(12, 6))
plt.title('actual vs prediction')
plt.plot(df['Date'][1138:], Y[1138:], label = 'Actual')
plt.plot(df['Date'][1138:], pred[1138:], label = 'Prediction')
plt.legend()
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
plt.title('actual vs prediction')
plt.plot(df['Date'][1138:], Y[1138:], label = 'Actual')
plt.plot(df['Date'][1138:], pred[1138:], label = 'Prediction')
plt.legend()
plt.show()

# write regression score to excel
overall=pd.DataFrame.from_records(result,columns=['hidden_layer','dataset1_acc'
,'dataset2_acc'])

overall.to_excel("regression_accuracy.xlsx")
```

Ensemble Classifiers:

```
from sklearn import tree
import pydotplus
import collections
import numpy as np
import pandas
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
AdaBoostClassifier, GradientBoostingClassifier, \
    VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import preprocessing as preproc
from sklearn import model_selection
from sklearn.metrics import accuracy_score
import xgboost as xgb
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# filename is for the dataset in use for the model. We have three datasets for
testing initially and we change the filename accordingly for each dataset we
use
filename = "finalDataset3.xlsx"
df = pandas.read_excel(filename)

# For each of our datasets, the feature columns can be different. Based on the
dataset in use, we use the features accordingly.
#features1 = df[['Volume', 'unemp_rate', 'T5YIE', 'Close', 'Open', 'High',
'Low', 'GDP']]
#features2 = df[['Volume', 'unemp_rate', 'T5YIE', 'Close', 'Open', 'High',
'Low', 'GDP', 'prev_close']]
features3 = df[['Volume', 'unemp_rate', 'T5YIE', 'Close', 'Open', 'High',
'Low', 'GDP', 'prev_close', 'prev_open', 'prev_high', 'prev_low']]

# For the Ensemble Model, we will predict the Profit column for the entire
dataset
targetColumn = df['Profit']

# Before test data split, we will perform scaling using MinMaxScaler for the
entire data of feature columns
scaler = preproc.MinMaxScaler()
scaler.fit(features3)
featureColumn = scaler.transform(features3)

# Using the seed value 42, we will split the dataset into test and train
accordingly. One third of the dataset is split into test set.
(X_train, X_test, Y_train, Y_test) =
model_selection.train_test_split(featureColumn, targetColumn, test_size=1/3,
random_state=42)
```

```
# When checking for VotingClassifier Model, we will use the following for-loop
code snippet to determine the accuracy score
# for estimator in range(10, 101, 10):
#     model1 = RandomForestClassifier(n_estimators=estimator)
#     model2 = XGBClassifier()
#     model3 = LogisticRegression()
#
#     model1.fit(X_train,Y_train)
#     model2.fit(X_train,Y_train)
#     model3.fit(X_train,Y_train)
#
#     model = VotingClassifier(estimators=[('dt', model1), ('knn', model2),
('lr', model3)], voting='hard')
#     model.fit(X_train, Y_train)
#     print("Accuracy using %d as estimator value: " % estimator,
model.score(X_test, Y_test))
#     print()

# When checking for RandomForest, Bagging, AdaBoost, GradientBoosting or XGB
Classifier, we will use the following for-loop code snippet to determine the
accuracy score
for estimator in range(10, 101, 10):
    #model = RandomForestClassifier(n_estimators = estimator,
criterion="entropy")
    model = BaggingClassifier(base_estimator=DecisionTreeClassifier(),
n_estimators=estimator)
    #model = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),
n_estimators=estimator)
    #model = GradientBoostingClassifier(n_estimators=estimator)
    #model = xgb.XGBClassifier(n_estimators=estimator)

    model.fit(X_train, Y_train)

    y_pred = model.predict(X_test)

    print("Accuracy using %d as estimator value: " %estimator,
accuracy_score(Y_test, y_pred))
```

Linear Regression:

```
import time
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
from sklearn import metrics
import numpy as np

df = pd.read_excel("/content/drive/MyDrive/Colab_Notebook/finalDataset.xlsx")
```

```
x =
df[['Date','Volume','unemp_rate','T5YIE','Close','Open','High','Low','GDP']]
y = df['targetPrice']

df.describe()

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

scaler = MinMaxScaler()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
train_size=0.7, random_state=22222)
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
y_train = y_train.to_frame()
y_test = y_test.to_frame()
scaler1 = MinMaxScaler()
scaler1.fit(y_train)
y_train = scaler1.transform(y_train)
y_test = scaler1.transform(y_test)

t1 = time.time()
reg = LinearRegression().fit(x_train, y_train)
print(time.time() - t1)

y_pred = reg.predict(x_test)
y1_pred = reg.predict(x_train)

print("MSE test score: ", metrics.mean_squared_error(y_test, y_pred))
print("R2 test score: ", reg.score(x_test, y_test))

print("R2 Train Score: ", reg.score(x_train, y_train))
print("MSE test score: ", metrics.mean_squared_error(y_train, y1_pred))
```

Ridge CV Regression:

```
import time
import pandas as pd
from sklearn.linear_model import RidgeCV
from sklearn import model_selection
from sklearn import metrics
import numpy as np

df = pd.read_excel("/content/drive/MyDrive/Colab_Notebook/finalDataset.xlsx")
#df = pd.read_csv("file.csv")
x =
df[['Date','Volume','unemp_rate','T5YIE','Close','Open','High','Low','GDP']]
```

```
y = df['targetPrice']

df.describe()

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

scaler = MinMaxScaler()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
train_size=0.7, random_state=22222)
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
y_train = y_train.to_frame()
y_test = y_test.to_frame()
scaler1 = MinMaxScaler()
scaler1.fit(y_train)
y_train = scaler1.transform(y_train)
y_test = scaler1.transform(y_test)

t1 = time.time()
reg = RidgeCV().fit(x_train, y_train)
print(time.time() - t1)

y_pred = reg.predict(x_test)
y1_pred = reg.predict(x_train)

print("MSE test score: ", metrics.mean_squared_error(y_test, y_pred))
print("R2 test score: ", reg.score(x_test, y_test))

print("R2 Train Score: ", reg.score(x_train, y_train))
print("MSE test score: ", metrics.mean_squared_error(y_train, y1_pred))
```

Ensemble Regression:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.svm import SVR
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
df = pd.read_excel("finalDataset.xlsx")
y = df['targetPrice']
```

```
del df['targetPrice']
x = df[df.columns[3:13]]
# print(df)

list = [10, 20, 40, 60, 100]


scaler = MinMaxScaler()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
train_size=0.7, random_state=22222)
volume = x_test['Volume']
test_data = x_test

scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
y_train = y_train.to_frame()
y_test = y_test.to_frame()
scaler1 = MinMaxScaler()
scaler1.fit(y_train)
y_train = scaler1.transform(y_train)
y_test = scaler1.transform(y_test)

print("Bagging Regressor:")
for num in list:
   reg = BaggingRegressor(base_estimator=LinearRegression(), n_estimators=num,
random_state=999)
   reg.fit(x_train, y_train)
   train_acc = reg.score(x_train, y_train)
   test_acc = reg.score(x_test, y_test)
   print(num)
   mse = mean_squared_error(y_test, reg.predict(x_test))
   print(mse)
   print(train_acc)
   print(test_acc)
   print(" ")

print("Random Forest Regressor:")
for num in list:
   reg = RandomForestRegressor(n_estimators=num, random_state=999)
   reg.fit(x_train, y_train)
   train_acc = reg.score(x_train, y_train)
   test_acc = reg.score(x_test, y_test)
   print(num)
   mse = mean_squared_error(y_test, reg.predict(x_test))
   print(mse)
   print(train_acc)
   print(test_acc)
   print(" ")
print("Ada Boost Regressor:")
for num in list:
   reg = AdaBoostRegressor(base_estimator=LinearRegression(), n_estimators=num,
random_state=999)
   reg.fit(x_train, y_train)
   train_acc = reg.score(x_train, y_train)
   test_acc = reg.score(x_test, y_test)
   print(num)
   mse = mean_squared_error(y_test, reg.predict(x_test))
   print(mse)
   print(train_acc)
   print(test_acc)
   print(" ")

print("Gradient Boost Regressor:")
for num in list:
   reg = GradientBoostingRegressor(n_estimators=num, random_state=999)
   reg.fit(x_train, y_train)
   train_acc = reg.score(x_train, y_train)
   test_acc = reg.score(x_test, y_test)
   print(num)
   mse = mean_squared_error(y_test, reg.predict(x_test))
   print(mse)
   print(train_acc)
   print(test_acc)
   print(" ")

reg2 = LinearRegression()
reg3 = DecisionTreeRegressor()
print("Voting Regressor:")
reg = VotingRegressor([('lr', reg2), ('dt', reg3)])
reg.fit(x_train, y_train)
train_acc = reg.score(x_train, y_train)
test_acc = reg.score(x_test, y_test)
mse = mean_squared_error(y_test, reg.predict(x_test))
print(mse)
print(train_acc)
print(test_acc)
```

Income Calculation (Using the Random Forest Regressor):

```
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.svm import SVR
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
df = pd.read_excel("finalDataset.xlsx")
df1 = pd.read_excel("finalDataset.xlsx")
y = df['targetPrice']
del df['targetPrice']
# print(df)
x = df[df.columns[3:15]]
# print(x)

scaler = MinMaxScaler()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
train_size=0.7, random_state=22222)
volume = x_test['Volume']
test_data = x_test

scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
y_train = y_train.to_frame()
y_test = y_test.to_frame()
scaler1 = MinMaxScaler()
scaler1.fit(y_train)
y_train = scaler1.transform(y_train)
y_test = scaler1.transform(y_test)

test_data.to_excel("testData.xlsx")
test_data = pd.read_excel("testData.xlsx")
# print(test_data)
reg = RandomForestRegressor(n_estimators=60, random_state=999)
reg.fit(x_train, y_train)
y_pred = reg.predict(x_test)
preds = {'preds': y_pred}
target = {'target': y_test}
data = pd.DataFrame(preds)
```

```
frames = [test_data, data]
df = pd.concat(frames, axis=1)
print(df)
income = 0
y_pred = df['preds']
y_pred = y_pred.to_frame()
y_test = scaler1.inverse_transform(y_test)

y_pred = scaler1.inverse_transform(y_pred)

for i, rows in df.iterrows():
   if y_pred[i][0] > rows['Close']:
        income += ((1000) * y_test[i][0] / rows['Close']) - 1000
print(income)
```