PREDICTION OF STOCK PRICES

Applied Machine Learning

Preksha Dhoot (1810110177)

CONTENTS

ABSTRACT

Data mining and machine learning approaches can be incorporated into business intelligence systems to help users for decision support in many real-life applications.
Recently, numerous investigations for stock price prediction and portfolio management using machine learning have been trying to develop efficient mechanical trading systems. The stock price prediction problem is considered as a Markov process which can be optimized by reinforcement learning based algorithms.
In this report, we intend to **forecast the closing price** of the stocks of the company-**Reliance,** using the techniques of Time Series Analysis. We have obtained the dataset from **Yahoo finance**.We have thoroughly analysed and processed the data so that it can be properly used for training. The data is divided into 2 sets- Training and Validation. The training set is trained on each of the algorithms while the validation set is used for computing the accuracy of the model. Each model is tested twice- once with the inclusion of 'open price' and then while excluding it. The accuracy of each model along with the interpretation of various values is presented.
We have thoroughly analysed and processed the data so that it can be properly used for training. The data is divided into 2 sets- Training and Validation. The training set is trained on each of the algorithms while the validation set is used for computing the accuracy of the model. Each model is tested twice- once with the inclusion of 'open price' and then while excluding it. The accuracy of each model along with the interpretation of various values is presented.
We have employed various models for Time Series analysis for the data. We have first processed and normalised the data and then applied various models to predict the closing price of the Stocks.

INTRODUCTION

Time series analysis is a statistical technique that deals with time series data, or trend analysis. Time series data means that data is in a series of particular time periods or intervals. The data of Stock prices of various companies and stock exchanges are some prime examples of Time Series Data. Analysing and forecasting the stock prices in near future is extremely significant for investors around the globe and helps the experts in analysing the movement of the market. Prediction of stock prices has been an important area of research for a long time. While supporters of the efficient market hypothesis believe that it is impossible to predict stock prices accurately, there are formal propositions demonstrating that accurate modeling and designing of appropriate variables may lead to models using which stock prices and stock price movement patterns can be very accurately predicted.
Here, our data set has 13 variables, including Date. The dataset is for 10 years from 2011 to 2021. We have employed Machine Learning Algorithms like Linear Regression, Ridge Regression, Lasso Regression, KNN and SVM regressors.

LITERATURE REVIEW

In [01] Hu Z. et al. (2013) had discussed the prediction of the stock market using SVM. It is a good idea to use SVM as it always gives unique results and works well even at local minima. The authors worked on a dataset from the Federal Reserve Bank of St. Louis over the data of 15 companies. There is no definitive way to define a good or poor, so they had considered it a good investment if the stock price of a company surges over a period of time. The study demonstrated that SVM produces results with good accuracy for the sample of data which is outside the training sample.

In [02], Ashish Sharma and his colleagues conducted a study which gave a theoretical approach to predict stock prices with the help of basic regression models.The study explained the regression models and how their application can be useful in price prediction. This study gave us an idea about how choosing appropriate factors affecting stock price as variables can give better predictions.

In [03], Pushkar Khanal and Shree Raj Shakya conducted a study that stated that the Support Vector Machine algorithm gave best results for prediction with an effective accuracy as compared to most other machine learning algorithms and traditional technical methods. This study effectively explained how classification can be useful in prediction.

In[04], SVMs construct minimization is described in detail in [6]. Using Australian Stock Exchange, SVMs were used strictly for stock selection. This was an attempt to identify stocks that are likely to outperform the market by having exception returns. The equally weighted portfolio formed by SVMs had a total return of 208% over a five year period, significantly outperforming the benchmark of 71%.

In[05], Sharma A. et al. (2017) used regression modelling to make predictions. The model was redefined each time the degree of the problem changes. The experimentation was performed on Polynomial Regression, RBF Regression, Sigmoid Regression and LR. For prediction purposes, LR proved to be the most suited and they were fitted using the least squares approach.

DATASET DESCRIPTION

The dataset describes the prices of stocks of the company over the years 2012-2021. The data contains **2243 rows and 13 columns.**
There are **12 variables** in the dataset- Open price,High price,Low price,Close price,WAP, No. of shares, No. of trades,Total turnover,Deliverable quantity,Percentage deliverable quantity to traded quantity to traded quantity,Spread high-low and Spread close open. All of them are **numeric data.**

No.	Feature Name	Feature Description
1	Open Price	Starting at which the stock is traded on a particular day
2	High Price	Maximum price of the share for the day
3	Low Price	Minimum price of the share for the day
4	Close Price	Final price at which the stock is traded on a particular day
5	WAP	The weighted average price of that day
6	No. of shares	The number of shares bought or sold in the day
7	No. of trades	The number of trades bought or sold in the day
8	Total Turnover	Turnover of the particular company on a given date
9	Deliverable Quantity	Number of shares that are marked for delivery
10	Percentage deliverable quantity to traded quantity	Ratio of deliverable quantity to traded quantity
11	Spread High-Low	Gives an idea about the total price movement over the tick period
12	Spread close-open	Gives an indication of the direction of the move

Date	Date	Open Price	High Price	Low Price	Close Price	WAP	No.of Shares	No. of Trades	Total Turnover (Rs.)	Deliverable Quantity	% Deli. Qty to Traded Qty	Spread High-Low	Spread Close-Open
2012-02-02	2012-02-02	836.50	842.35	824.00	828.60	831.89	696934	17024	579775429.00	224017.00	32.14	18.35	-7.90
2012-02-03	2012-02-03	832.00	839.40	815.55	837.75	827.91	660802	15706	547082582.00	229118.00	34.67	23.85	5.75
2012-02-06	2012-02-06	843.00	849.00	824.00	832.75	834.68	550496	16933	459485735.00	162179.00	29.46	25.00	-10.25
2012-02-07	2012-02-07	841.00	852.40	834.95	844.75	846.62	1011946	22870	856730998.00	371859.00	36.75	17.45	3.75
2012-02-08	2012-02-08	848.00	864.45	845.00	858.05	857.32	945448	20363	810551798.00	362375.00	38.33	19.45	10.05
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2021-02-12	2021-02-12	2059.70	2077.50	2015.55	2041.25	2051.01	640628	27971	1313933137.00	88301.00	13.78	61.95	-18.45
2021-02-15	2021-02-15	2049.00	2060.55	2025.40	2032.60	2042.73	325784	14989	665488339.00	48870.00	15.00	35.15	-16.40
2021-02-16	2021-02-16	2040.00	2079.00	2037.50	2059.65	2061.48	361465	17723	745152334.00	92295.00	25.53	41.50	19.65

DATA PREPROCESSING

REMOVING NULL VALUES

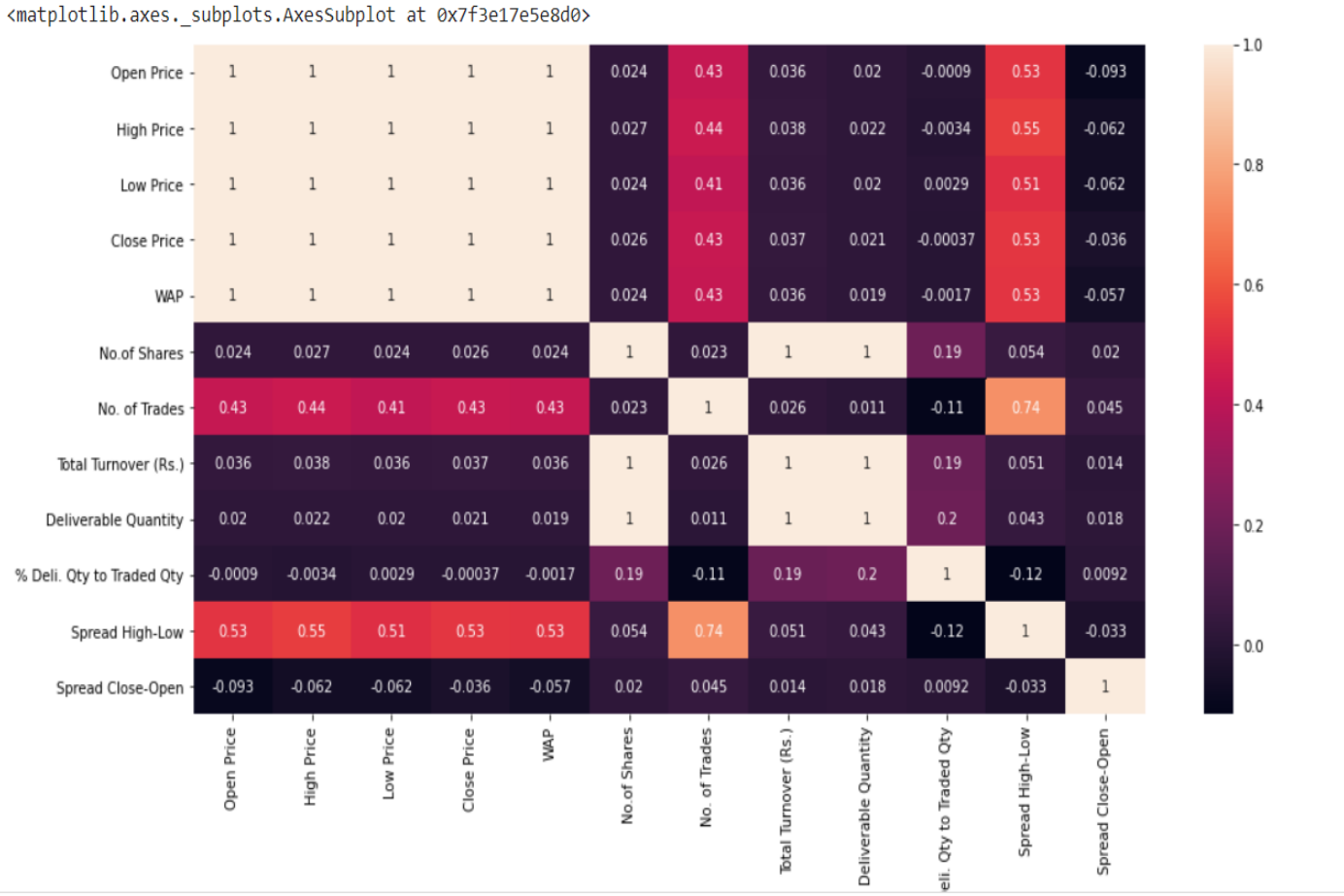
There was a null value in two rows in the dataset- in the deliverable quantity and spread high-low. Hence,we removed the two rows.

```
[ ]  data1=data.dropna()
     data1.isna().sum()
```

```
Date                          0
Open Price                    0
High Price                    0
Low Price                     0
Close Price                   0
WAP                           0
No.of Shares                  0
No. of Trades                 0
Total Turnover (Rs.)          0
Deliverable Quantity          0
% Deli. Qty to Traded Qty     0
Spread High-Low               0
Spread Close-Open             0
dtype: int64
```

AUTOCORRELATION

On checking the autocorrelation between the variables,we found that the features Open Price,High Price,Low Price and WAP are very highly correlated with the output variable-Close Price which is why we decided to exclude these columns from the dataset before training it. The features-No. Of Shares,Total Turnover and Deliverable Quantity are also very highly correlated with each other but do not affect Close Price too much.



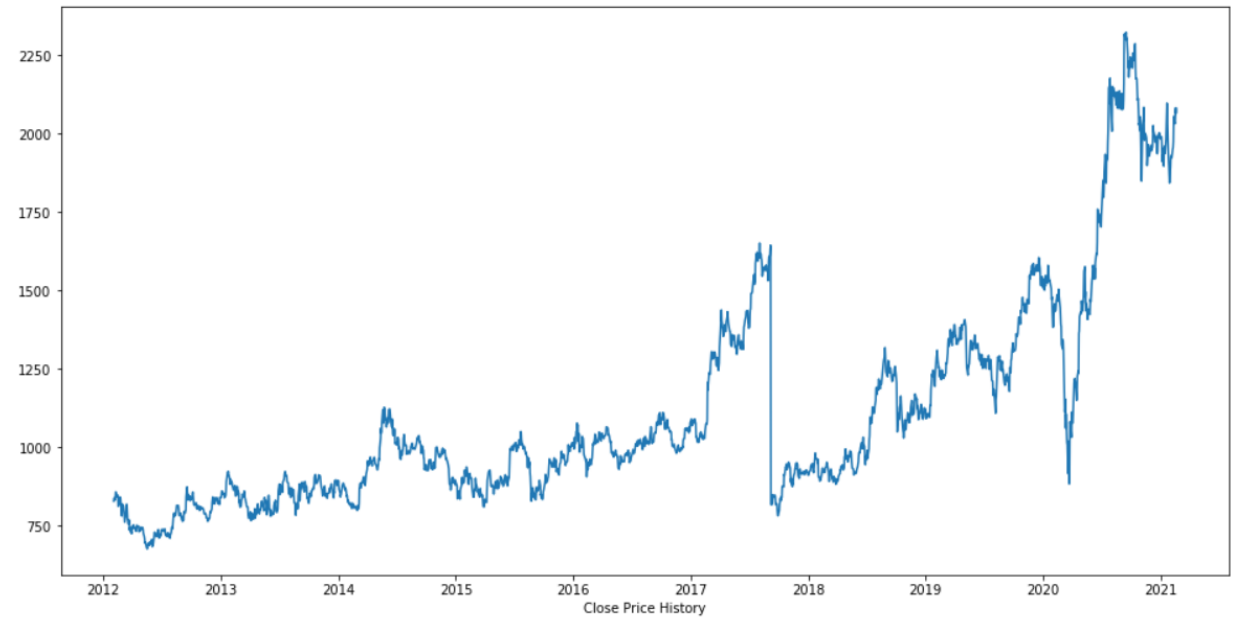
NUMERICAL VARIABLES

Visualizing numerical variable

We plotted the trend of variables so as to analyse how they are changing over time. Following are the various plots that we obtained. The plot of “Close Price History” with “Date” shows a dip during the early months of the year 2020. It is highly likely that the dip in prices was due to the onset of the Covid-19 Pandemic.

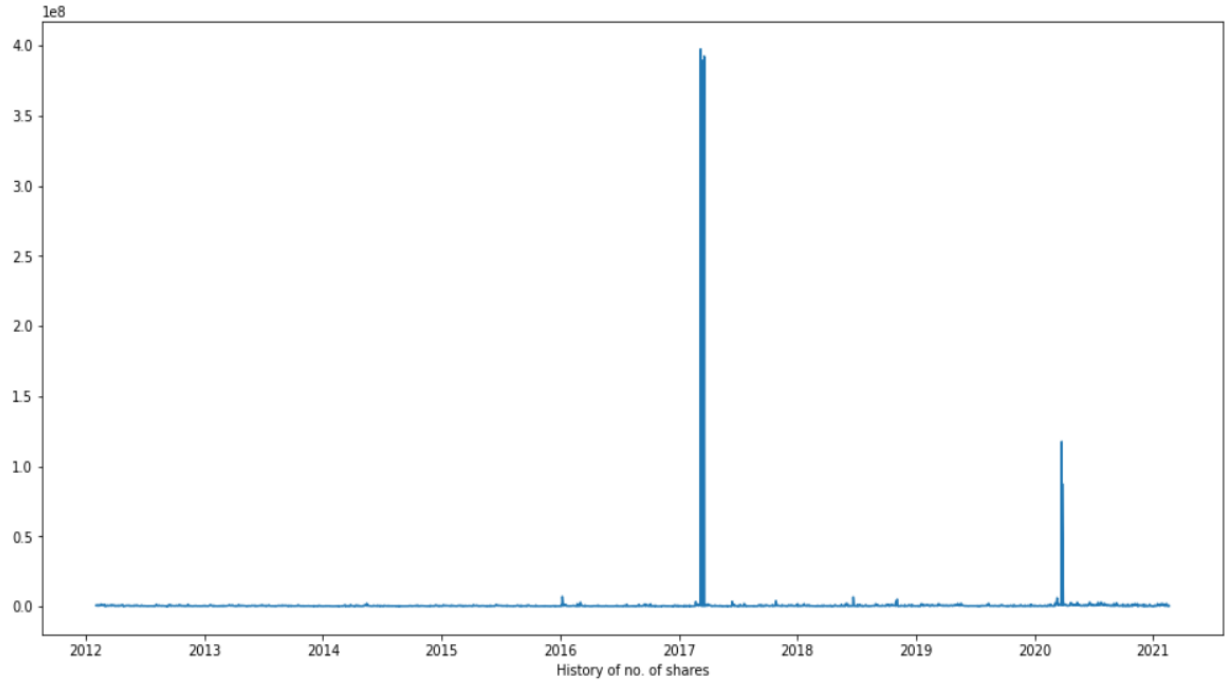
Trends of Close Price History

```
Text(0.5, 0, 'Close Price History')
```

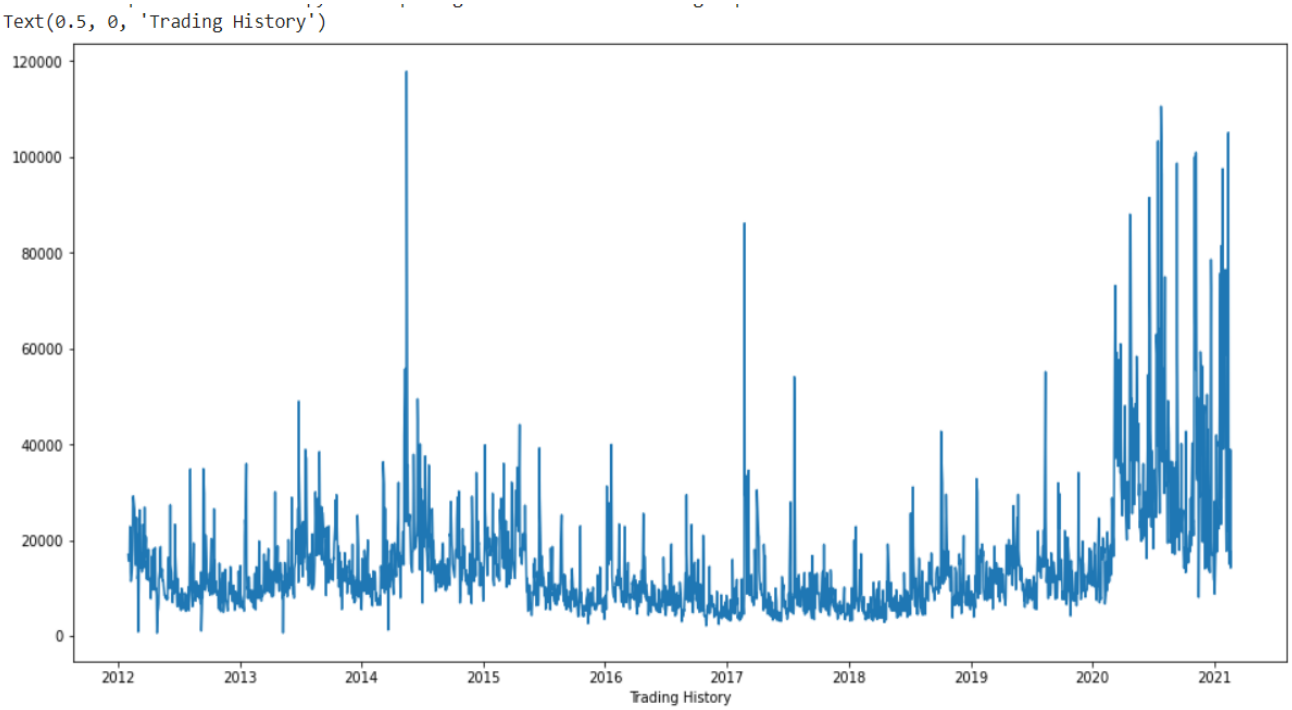


Trend of No. of Shares

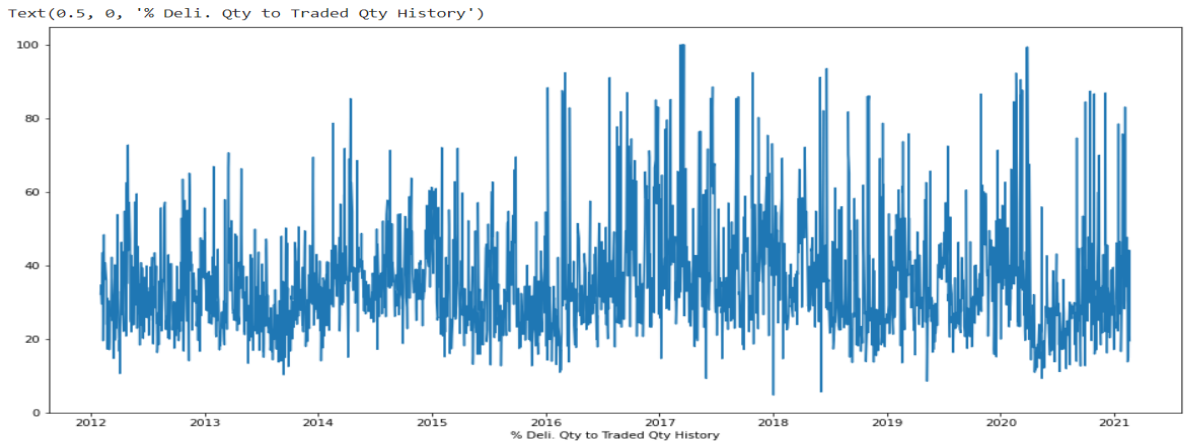
```
Text(0.5, 0, 'History of no. of shares')
```



Trend of No. of Trades

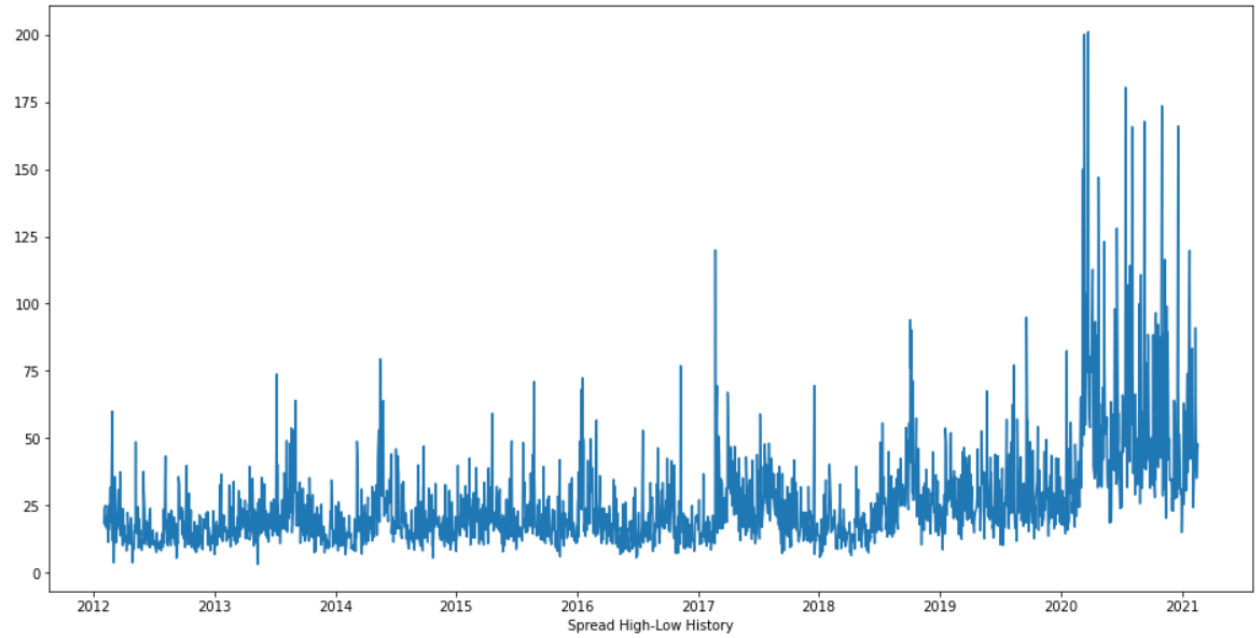


Trend of the percentage of Deliverable quantity to Traded Quantity



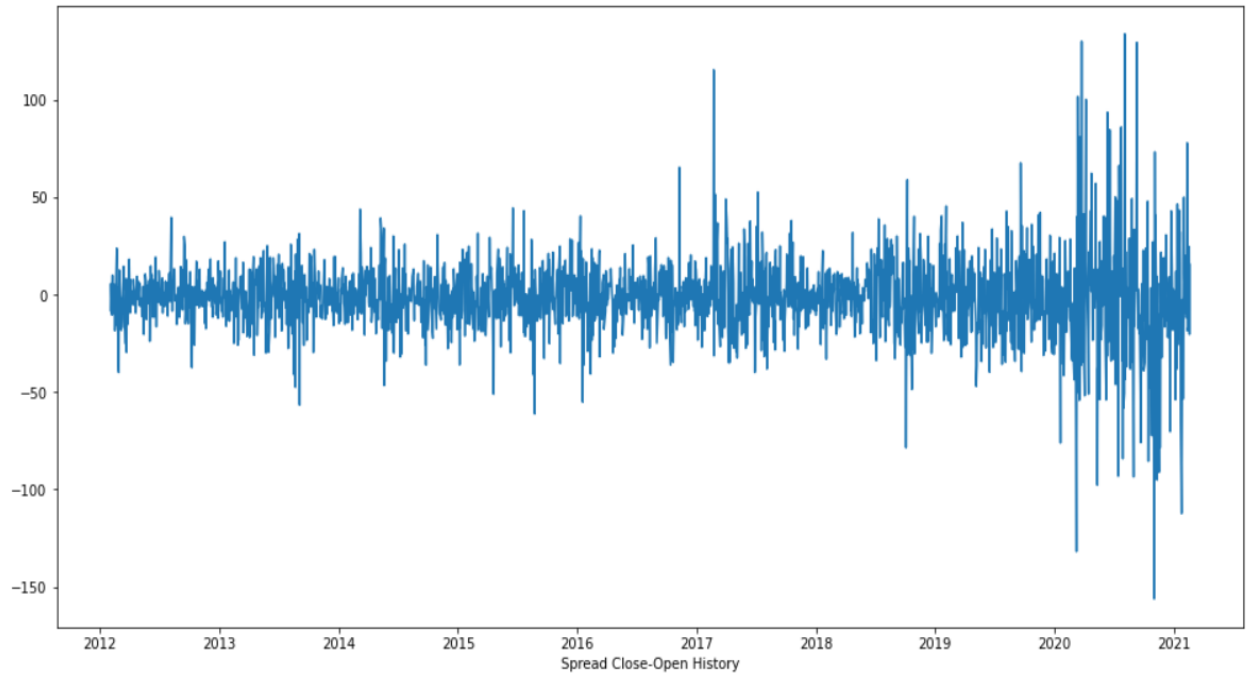
Trend of Spread High-Low

```
Text(0.5, 0, 'Spread High-Low History')
```



Trend of Spread of Open-Close History

```
Text(0.5, 0, 'Spread Close-Open History')
```



SORTING THE DATA

We have set the index as date and sorted the set using date as an index as we need to figure out the trend of the data with respect to time and arranging it in ascending order helps in more effective and systematic analysis of the data.

Date	Date	Open Price	High Price	Low Price	Close Price	WAP	No.of Shares	No. of Trades	Total Turnover (Rs.)	Deliverable Quantity	% Deli. Qty to Traded Qty	Spread High-Low	Spread Close-Open
2012-02-02	2012-02-02	836.50	842.35	824.00	828.60	831.894310	696934	17024	5.797754e+08	224017.0	32.14	18.35	-7.90
2012-02-03	2012-02-03	832.00	839.40	815.55	837.75	827.906971	660802	15706	5.470826e+08	229118.0	34.67	23.85	5.75
2012-02-06	2012-02-06	843.00	849.00	824.00	832.75	834.675883	550496	16933	4.594857e+08	162179.0	29.46	25.00	-10.25
2012-02-07	2012-02-07	841.00	852.40	834.95	844.75	846.617308	1011946	22870	8.567310e+08	371859.0	36.75	17.45	3.75
2012-02-08	2012-02-08	848.00	864.45	845.00	858.05	857.320337	945448	20363	8.105518e+08	362375.0	38.33	19.45	10.05
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2021-02-12	2021-02-12	2059.70	2077.50	2015.55	2041.25	2051.007975	640628	27971	1.313933e+09	88301.0	13.78	61.95	-18.45
2021-02-15	2021-02-15	2049.00	2060.55	2025.40	2032.60	2042.728737	325784	14989	6.654883e+08	48870.0	15.00	35.15	-16.40
2021-02-16	2021-02-16	2040.00	2079.00	2037.50	2059.65	2061.478522	361465	17723	7.451523e+08	92295.0	25.53	41.50	19.65

CONVERTING DATE FORMAT

As Date cannot be treated as a variable by python, we divided the date into usable features using the **add_datepart module of fastai** library available in python. The date is divided into the features such as Year, Month, Week, Day and the Day of week,Day of year, Is month end,Is month start,Is quarter end,Is quarter start, Is year end,Is year start to help us extract information from the date values.

Spread Close-Open	Year	Month	Week	Day	Dayofweek	Dayofyear	Is_month_end	Is_month_start	Is_quarter_end	Is_quarter_start	Is_year_end	Is_year_start
-7.9	2012	2	5	2	3	33	False	False	False	False	False	False
5.75	2012	2	5	3	4	34	False	False	False	False	False	False
-10.25	2012	2	6	6	0	37	False	False	False	False	False	False
3.75	2012	2	6	7	1	38	False	False	False	False	False	False
10.05	2012	2	6	8	2	39	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
-18.45	2021	2	6	12	4	43	False	False	False	False	False	False
-16.4	2021	2	7	15	0	46	False	False	False	False	False	False
19.65	2021	2	7	16	1	47	False	False	False	False	False	False

ADDING A NEW FEATURE

Stock prices are often highly influenced by the fact that if a certain day is the first and last day of the week. Hence, we have created a new variable called ‘mon_fri’ that gives importance to the first and last day of the week.

```
#adding a new feature that gives importance to the first and last day of the week
new_data_ts['mon_fri'] = 0
for i in range(0,len(new_data_ts)):
    if (new_data_ts['Dayofweek'][i] == 0 or new_data_ts['Dayofweek'][i] == 4):
        new_data_ts['mon_fri'][i] = 1
    else:
        new_data_ts['mon_fri'][i] = 0
```

Spread Close-Open	Year	Month	Week	Day	Dayofweek	Dayofyear	Is_month_end	Is_month_start	Is_quarter_end	Is_quarter_start	Is_year_end	Is_year_start	mon_fri
-7.9	2012	2	5	2	3	33	False	False	False	False	False	False	0
5.75	2012	2	5	3	4	34	False	False	False	False	False	False	1
-10.25	2012	2	6	6	0	37	False	False	False	False	False	False	1
3.75	2012	2	6	7	1	38	False	False	False	False	False	False	0
10.05	2012	2	6	8	2	39	False	False	False	False	False	False	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...

DATA NORMALISATION

Since the data has values of varying ranges, we have normalised the data using the MinMaxScalar function available in the Preprocessing module of Sklearn library**.**

	Close Price	No.of Shares	No. of Trades	% Deli. Qty to Traded Qty	Spread High-Low	Spread Close-Open	Year	Month	Week	Day	Dayofweek	Dayofyear
0	0.092495	0.001712	0.139880	0.287605	0.077059	0.510939	0.0	0.090909	0.076923	0.033333	0.500000	0.087671
1	0.098049	0.001621	0.128634	0.314181	0.104851	0.557967	0.0	0.090909	0.076923	0.066667	0.666667	0.090411
2	0.095014	0.001343	0.139104	0.259454	0.110662	0.502842	0.0	0.090909	0.096154	0.166667	0.000000	0.098630
3	0.102297	0.002505	0.189764	0.336029	0.072511	0.551077	0.0	0.090909	0.096154	0.200000	0.166667	0.101370
4	0.110369	0.002337	0.168372	0.352626	0.082617	0.572782	0.0	0.090909	0.096154	0.233333	0.333333	0.104110
...	...	...	...	...	...	...	...	...	...	...	...	...
2236	0.828483	0.001570	0.233290	0.094748	0.297372	0.474591	1.0	0.090909	0.096154	0.366667	0.666667	0.115068
2237	0.823233	0.000778	0.122516	0.107563	0.161950	0.481654	1.0	0.090909	0.115385	0.466667	0.000000	0.123288
2238	0.839650	0.000867	0.145845	0.218172	0.194037	0.605857	1.0	0.090909	0.115385	0.500000	0.166667	0.126027
2239	0.853670	0.001701	0.326641	0.413445	0.207428	0.623428	1.0	0.090909	0.115385	0.533333	0.333333	0.128767
2240	0.843201	0.000709	0.116065	0.167122	0.226882	0.468389	1.0	0.090909	0.115385	0.566667	0.500000	0.131507

APPLICATION OF MODELS AND RESULTS

We have implemented three machine learning algorithms- Linear Regression, Ridge Regression Lasso regression,KNN and SVM to predict the future stock price of the company. Before applying the models, we have divided the dataset into **Training Data and Test Data in the ratio 70:30**

```
X=scaled_data.drop('Close Price', axis =1)
Y=scaled_data['Close Price']

#split data into train and test
x_train = X[:1458]
x_test = X[1458:]
y_train = Y[:1458]
y_test = Y[1458:]
```

LINEAR REGRESSION

Linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The representation of a linear regression model is a linear equation that combines a specific set of input values (x). The solution to which is the predicted output for that set of input values (y).

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \ldots + w[n] \times x[n] + b$$

Linear model with n features for output prediction

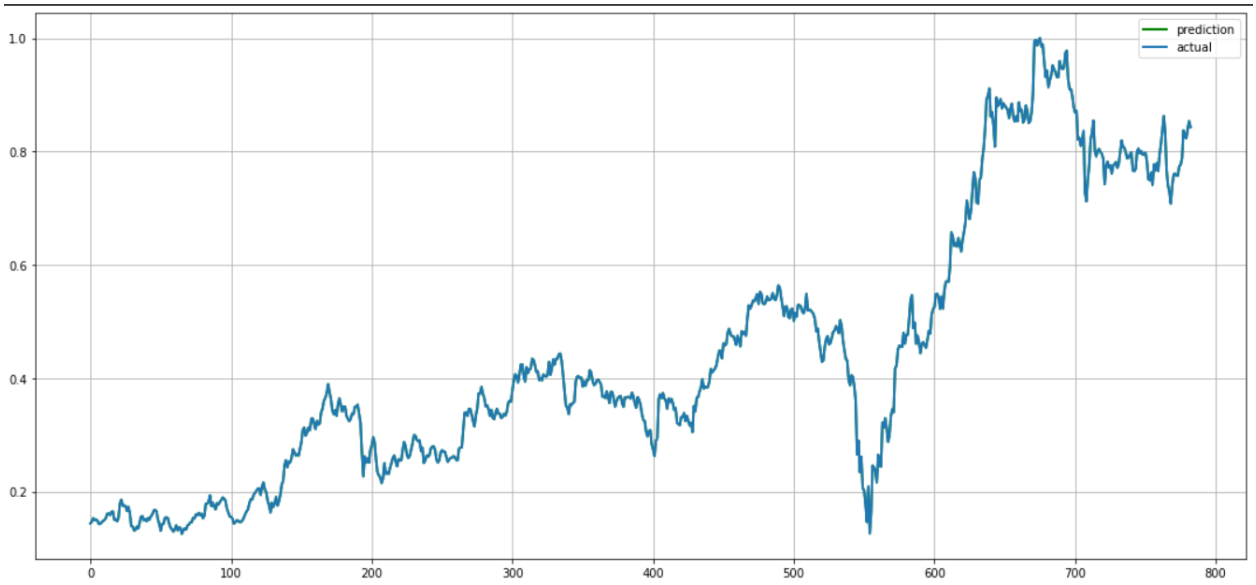
Cost function for linear regression would be given as:

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2$$

Cost function for simple linear model

Output from the linear regression model :

\



RIDGE REGRESSION

The ridge regression is a variant of Linear Regression where all the parameters are regularised equally. It is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors, Ridge regression aims at reducing the mean error.

In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients.

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}w_j^2$$

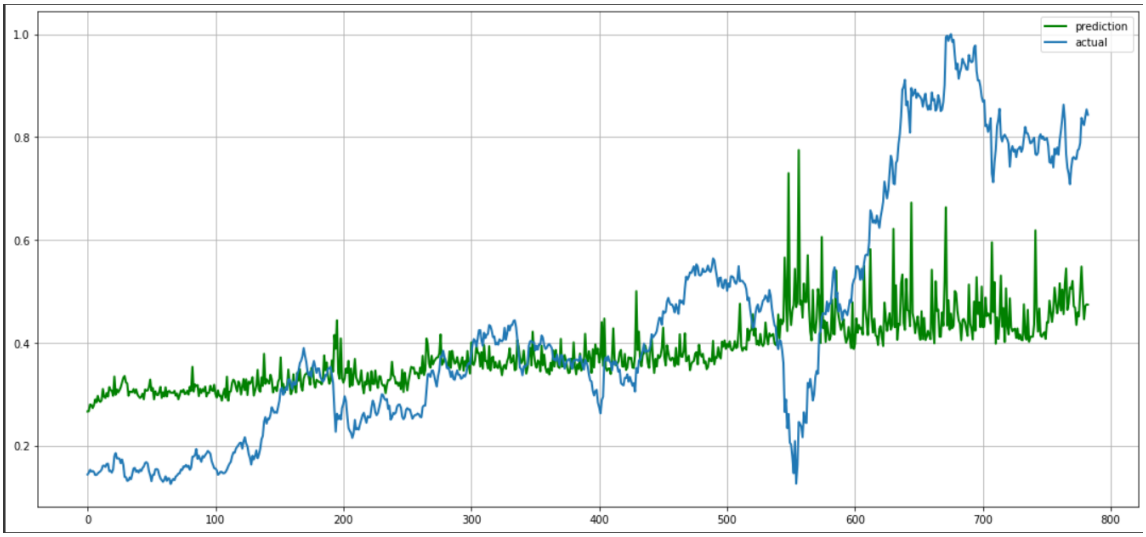
Cost function for ridge regression

Output of Ridge Regression model :

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=0.5)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.2011220496546831
0.25910946537850343
```



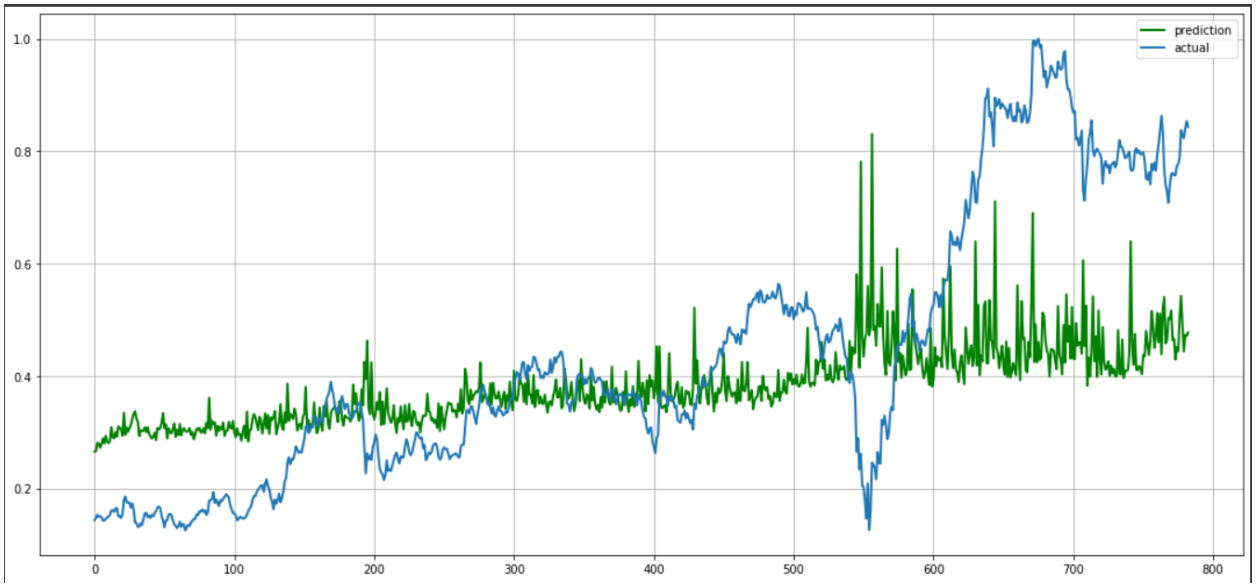
On varying the alpha to find its optimum value,we observed the following change in values-

For alpha=0.05,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=0.05)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.2021831328177737
0.25127123723930744
```

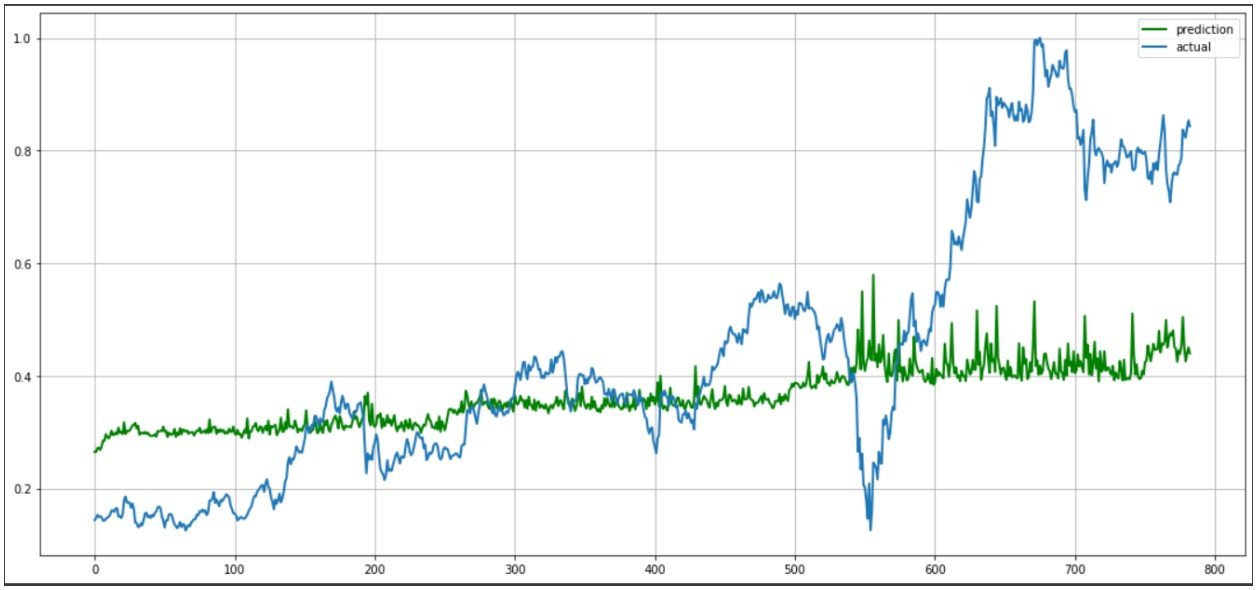


For alpha=5,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=5)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.21043142876989626
0.1889345872052941
```

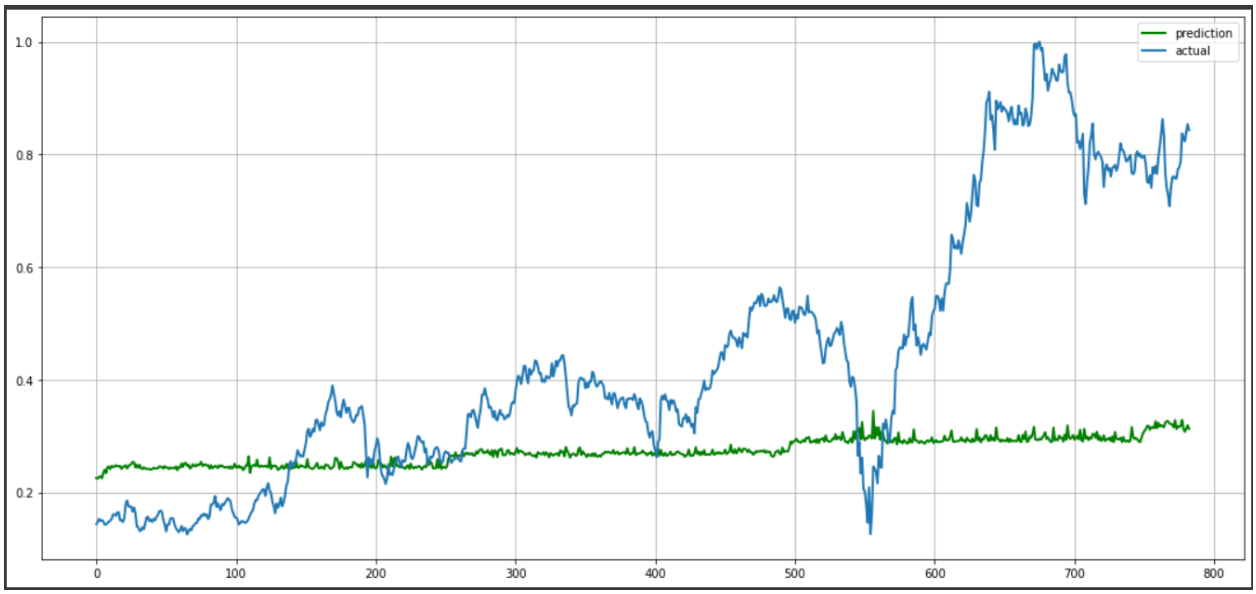


For alpha=50,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=50)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.27096974066112794
-0.344857487238144
```

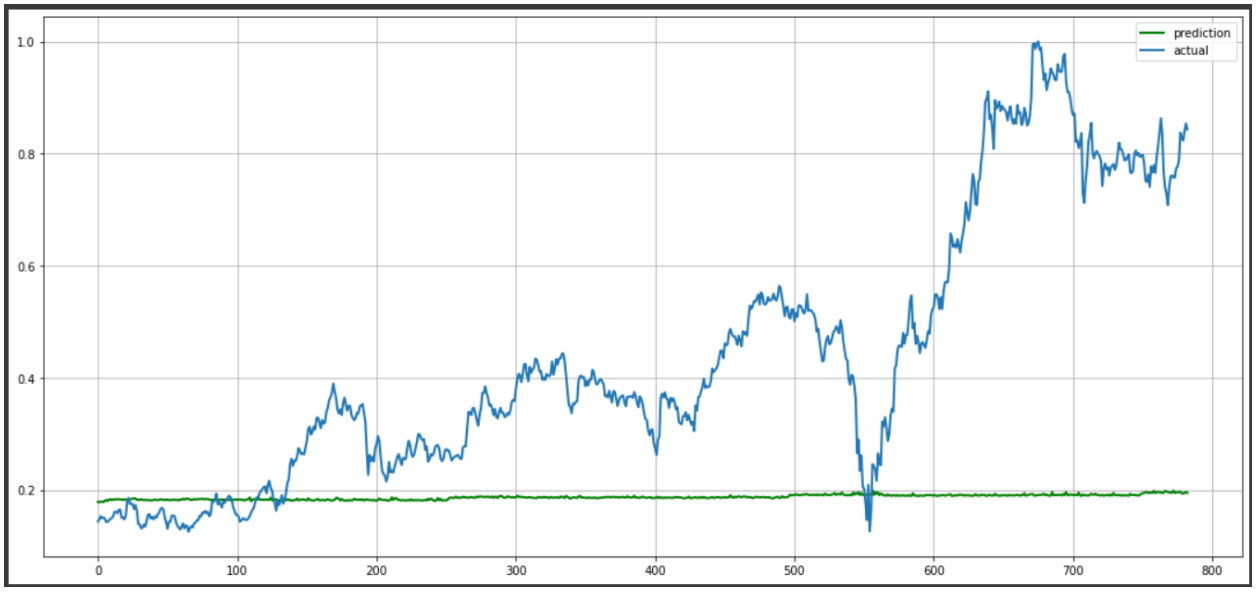


For alpha=500,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=500)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.3392094654256541
-1.1075146330870758
```



LASSO REGRESSION

The Lasso regression is another variant of Linear Regression. Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models.
The cost function for Lasso Regression would be given as:

$$\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} |w_j|$$

Cost function for Lasso regression

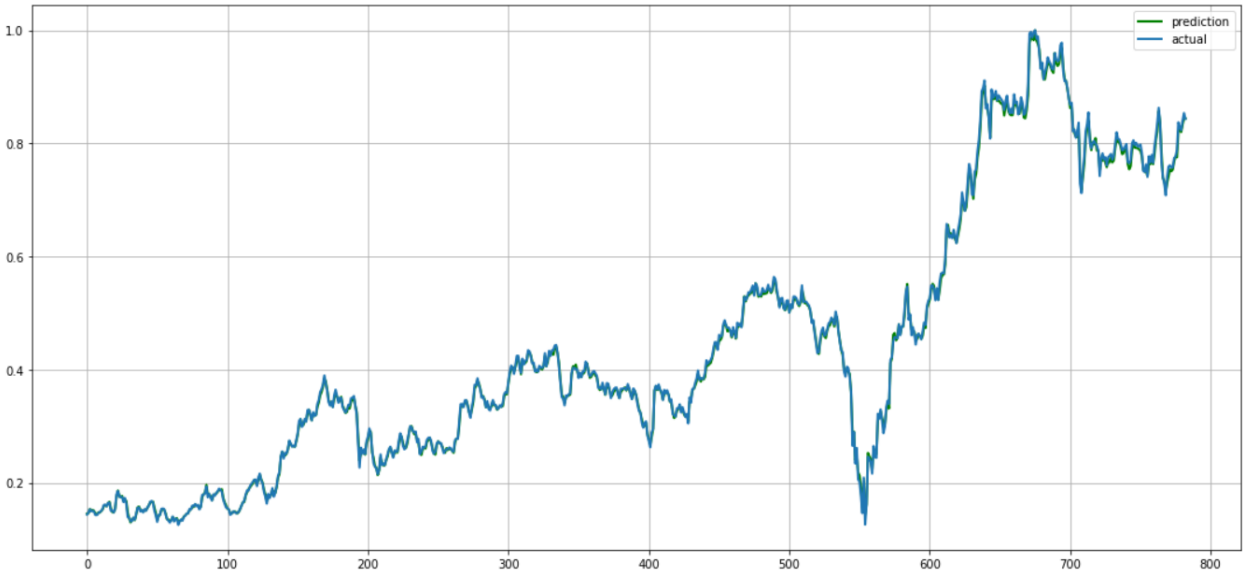
Output on applying Lasso Regression model :

For alpha = 0.0001

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=0.0001)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.00551296371458451
0.9994433208673841
```

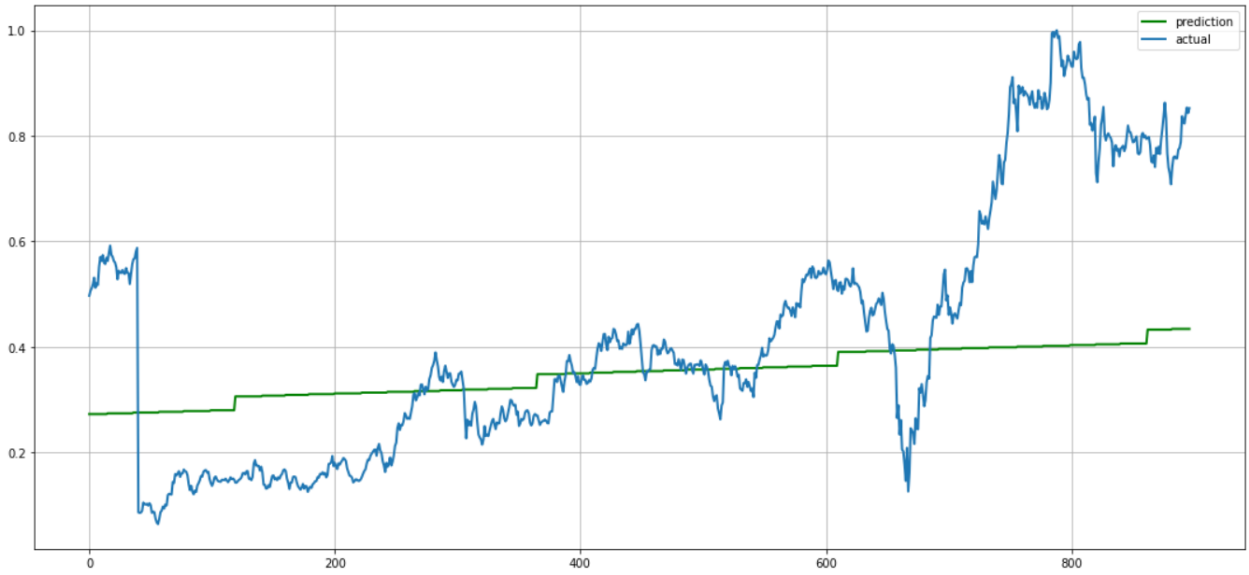


On varying the alpha value,we observed the following change in values-

For alpha = 0.001

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=0.001)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

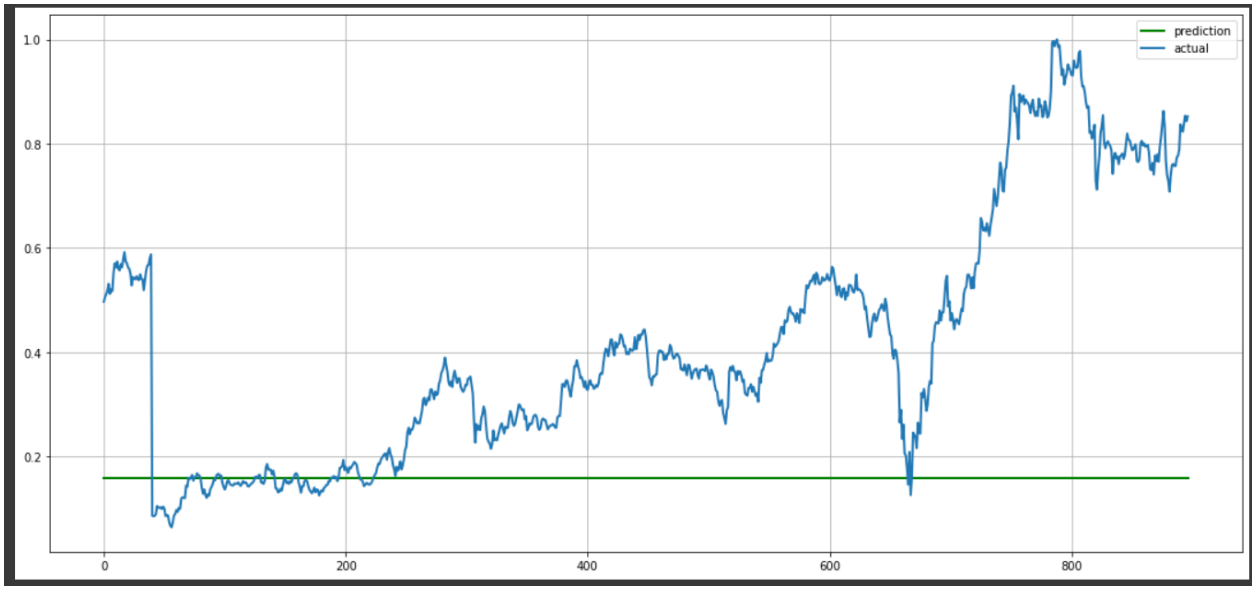


For alpha = 0.1

```
[53] from sklearn.linear_model import Lasso
     model_lasso = Lasso(alpha=0.1)
     model_lasso.fit(x_train,y_train)

     preds4 = model_lasso.predict(x_test)
     print(np.sqrt(mean_squared_error(y_test, preds4)))
     print(r2_score(y_test,preds4))

     0.34964853946982877
     -1.1901036573616817
```

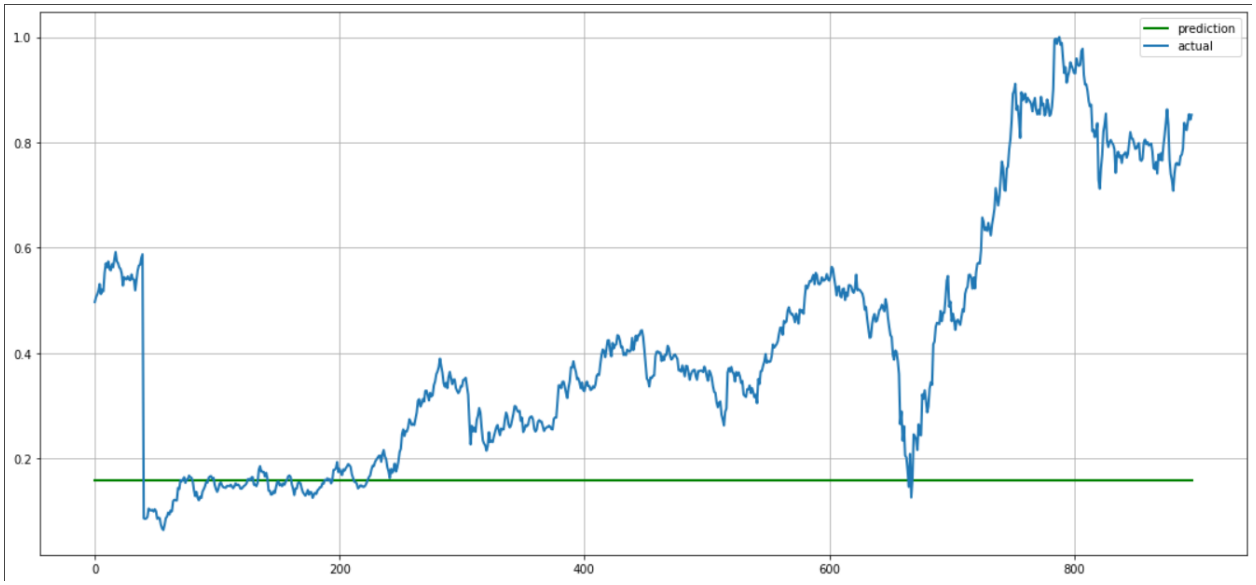


For alpha = 10

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=10)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.34964853946982877
-1.1901036573616817
```

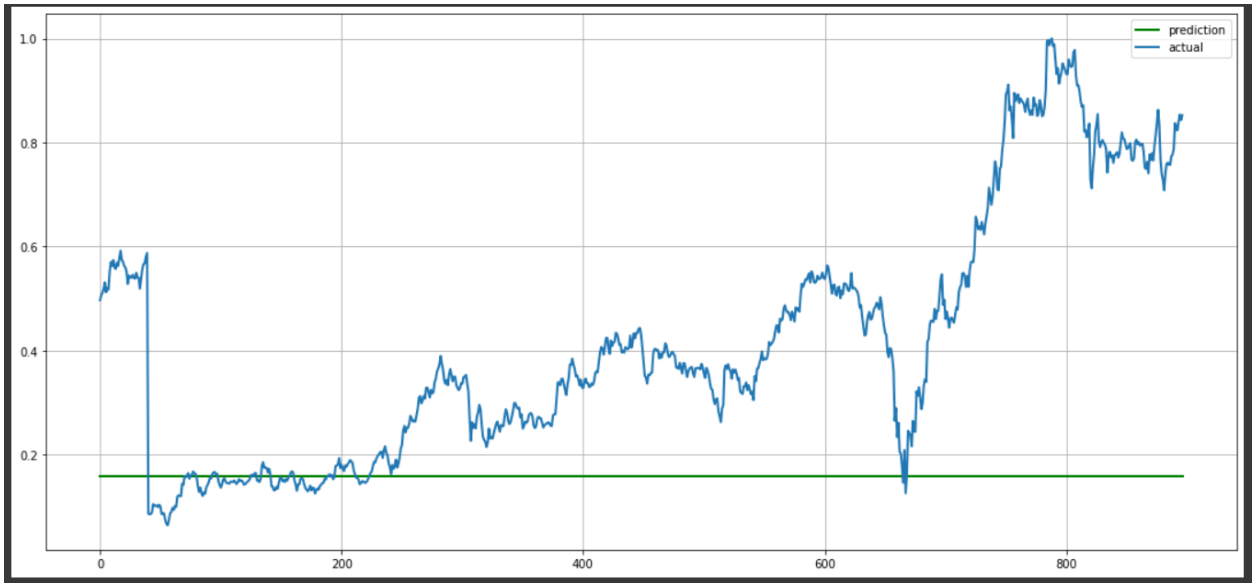


For alpha = 100

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=100)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.34964853946982877
-1.1901036573616817
```



We can see that there is an improvement in the values of R-square as we move from Linear to Ridge to Lasso regression.

KNN(k Nearest Neighbours)

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same *neighbourhood*.
K nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions)

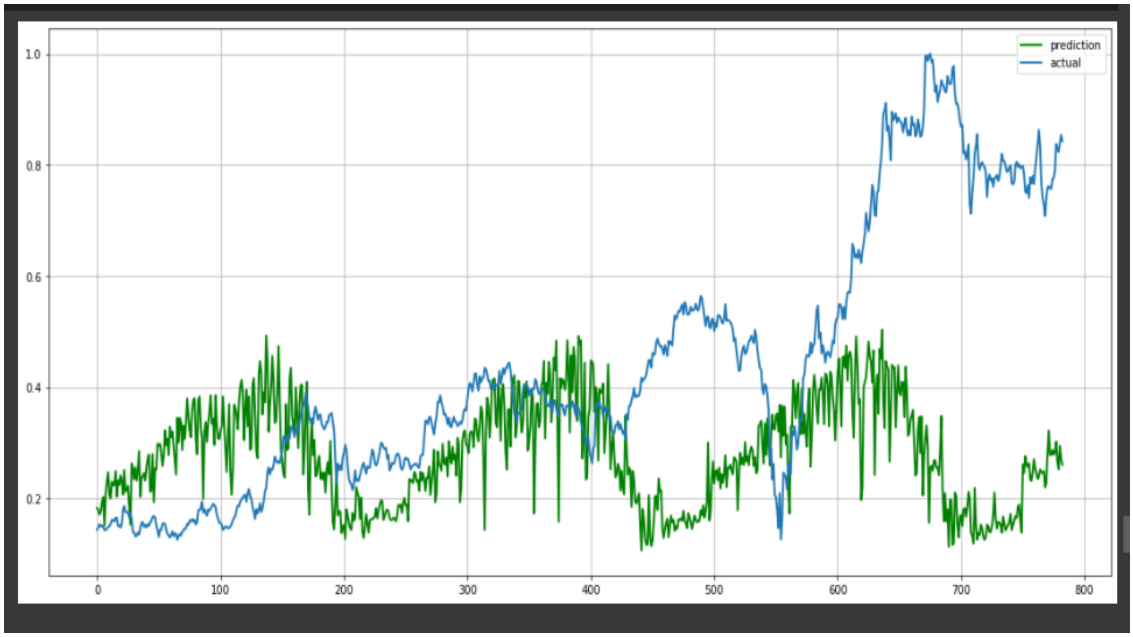
$$\hat{y}(x) = \frac{1}{k}\sum_{x_i \in N(x)} y_i$$

Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

```
from sklearn.model_selection import GridSearchCV
from sklearn import neighbors
params = {'n_neighbors':[1,2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model_knn = GridSearchCV(knn, params, cv=6)

model_knn.fit(x_train,y_train)
pred = model_knn.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, pred)))
print(r2_score(y_test,pred))
```

```
0.3033094905082993
-0.685026257765154
```



KNN did not perform well as there were multiple input values due to which selection of lesser number of inputs for this particular case is necessary.

SUPPORT VECTOR MACHINE

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.To separate the two classes of data points, there are many possible hyperplanes that could be

chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

The cost is 0 if the predicted value and the actual value are of the same sign.If they are not, we then calculate the loss value.
We also add a regularization parameter to the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost function looks as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

After achieving the loss function,we take the partial derivatives with respect to the weights to find the gradients.Using these gradients,we can update our weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

In case of no misclassification,i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularisation parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

In case of misclassification,i.e our model makes a mistake on the prediction of the class of our data point ,we include the loss along with the regularisation parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

The outputs using the SVM model are as follows :

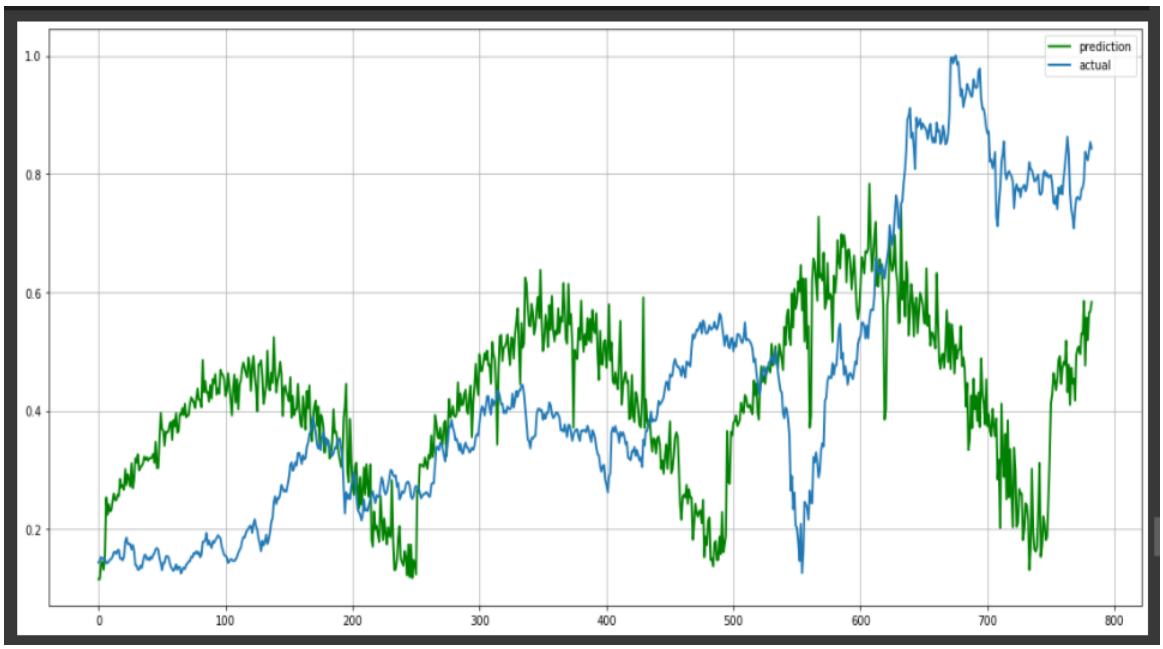
SVM(with kernel = linear)

```
from sklearn.svm import SVR
model_svm = SVR(kernel='linear')
model_svm.fit(x_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
pred5 = model_svm.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, pred5)))
print(r2_score(y_test,pred5))
```

```
0.21413627719544692
0.16012400227849033
```



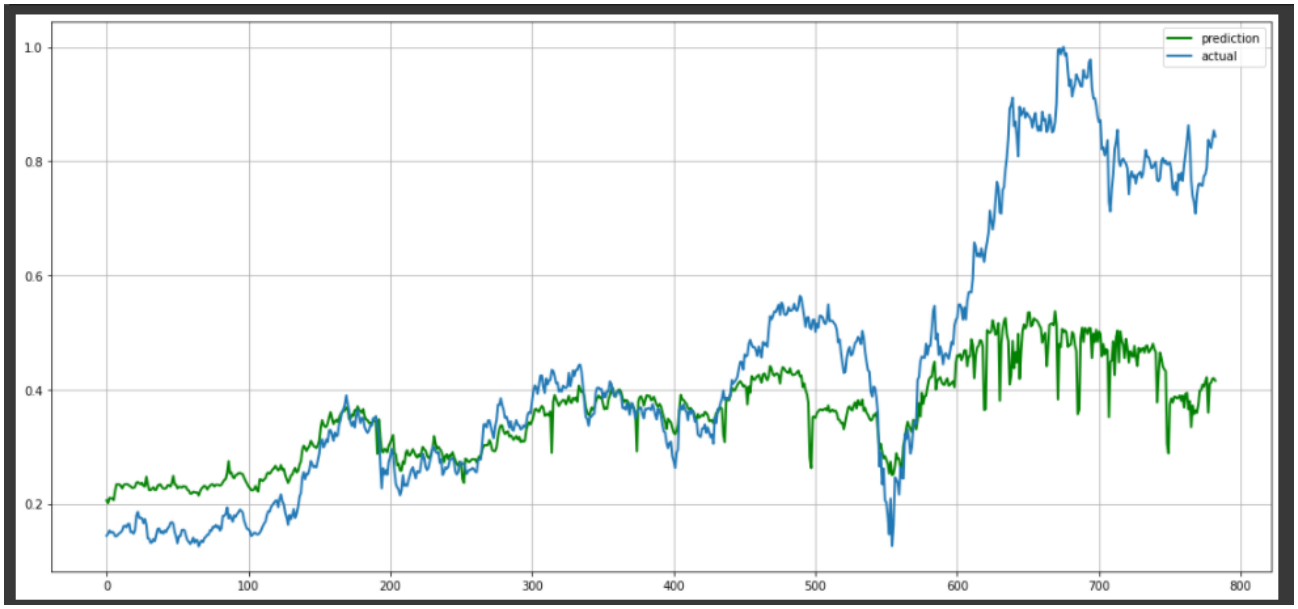
SVM(with Kernel = rbf)

```
from sklearn.svm import SVR
model_svm = SVR()
model_svm.fit(x_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
pred5 = model_svm.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, pred5)))
print(r2_score(y_test,pred5))
```

```
0.1838942176339435
0.3806005422522394
```



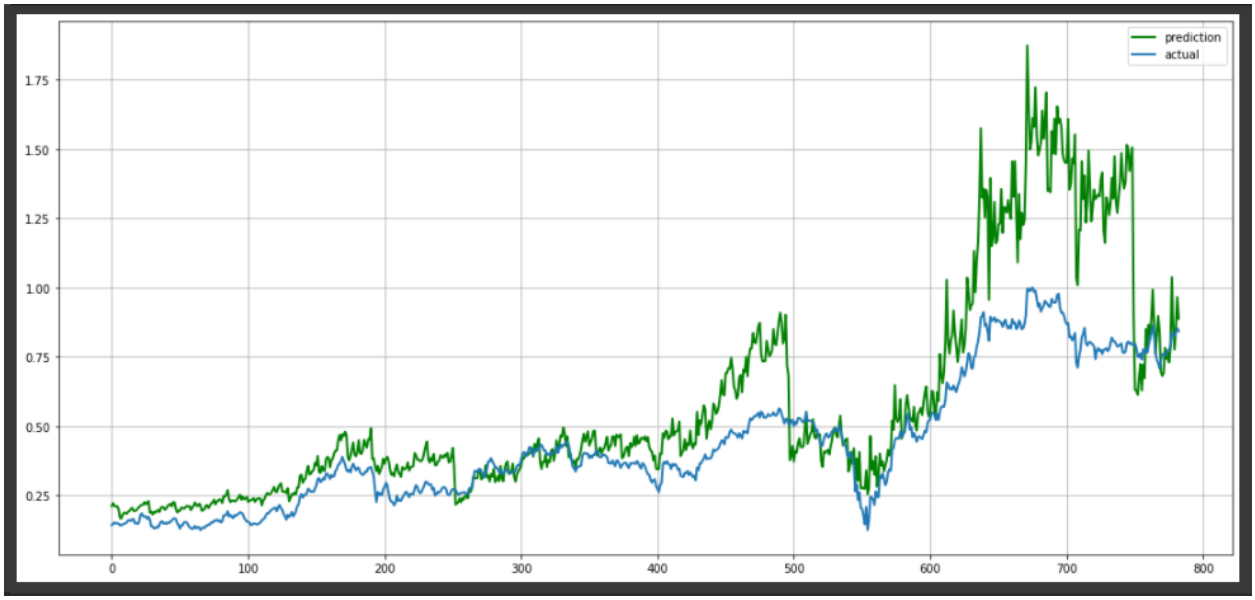
SVM(with kernel = poly)

```
from sklearn.svm import SVR
model_svm = SVR(kernel = 'poly')
model_svm.fit(x_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='poly', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
pred5 = model_svm.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, pred5)))
print(r2_score(y_test,pred5))
```

```
0.2290635900317974
0.03894819268767269
```



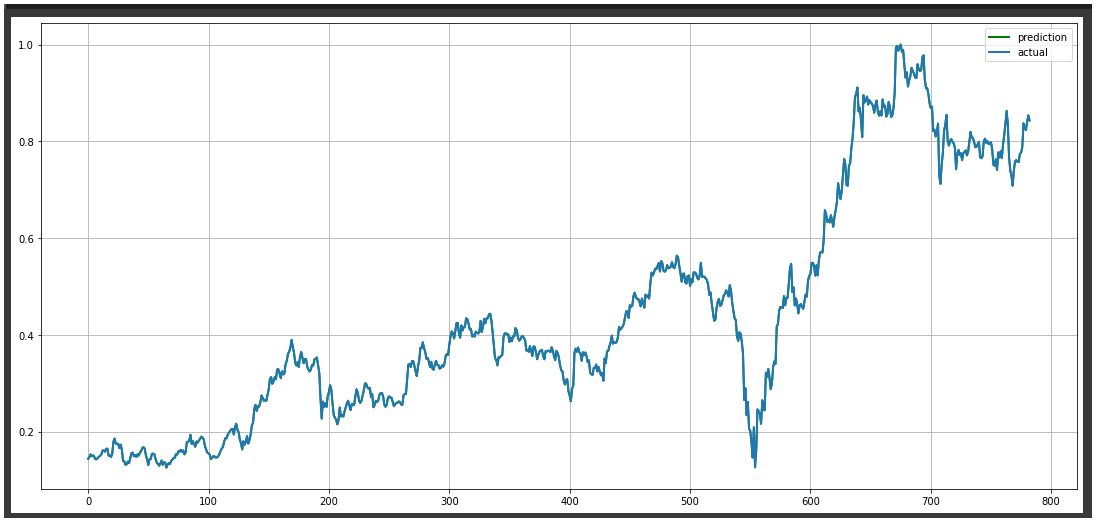
MODEL	RMSE	R-squared
Linear Regression	0.2171	0.1360
Ridge Regression	0.2011	0.2591
Lasso Regression	0.1981	0.2809
KNN	0.0301	-0.6804
SVM	0.2141	0.1601

Since we achieved very low values of R-squared, we can see that our models did not perform very well.So,we tried to include the previously excluded variables to see the change in performance of the data with respect to the variables. The applied models with their values and graphs are hereafter.

LINEAR REGRESSION

```
print(np.sqrt(mean_squared_error(y_test, preds1)))
print(r2_score(y_test,preds1))
```

```
2.793841541000079e-16
1.0
```

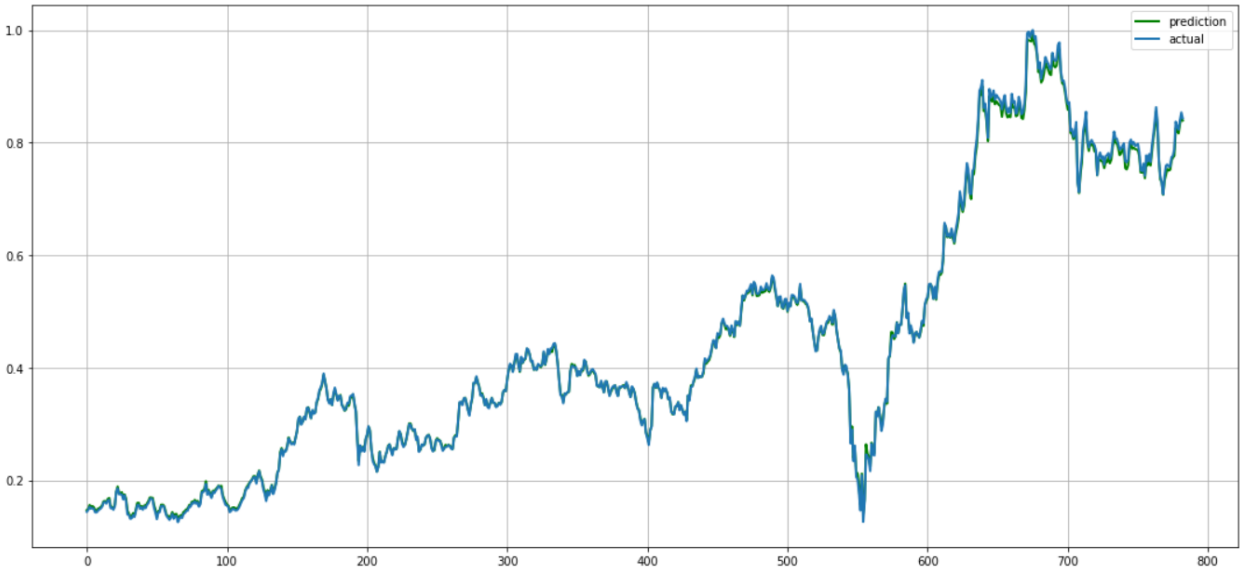


RIDGE REGRESSION

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=0.5)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.004979175707590706
0.9995459020115385
```



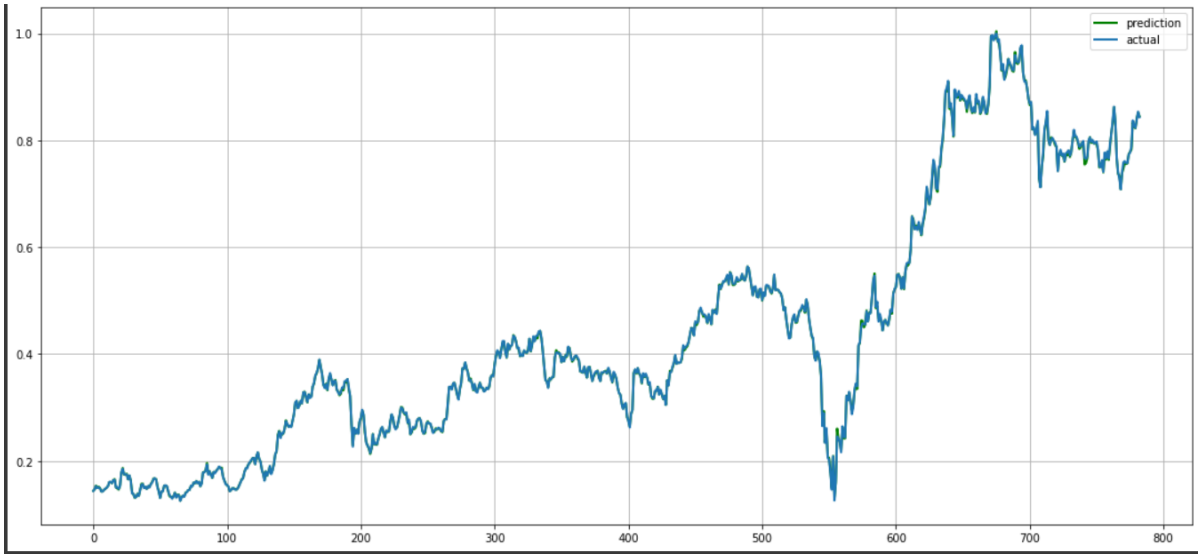
On varying the alpha to find its optimum value,we observed the following change in values-

For alpha=0.05,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=0.05)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.002182726294086497
0.9999127363781738
```

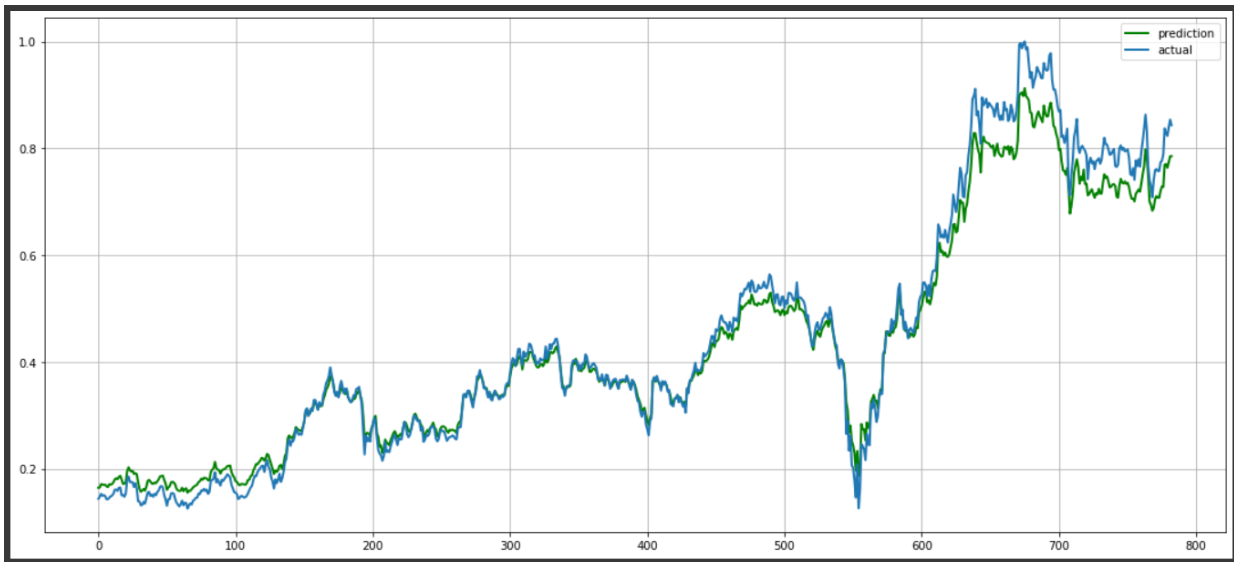


For alpha=5,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=5)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.03352424066114053
0.9794149193357783
```

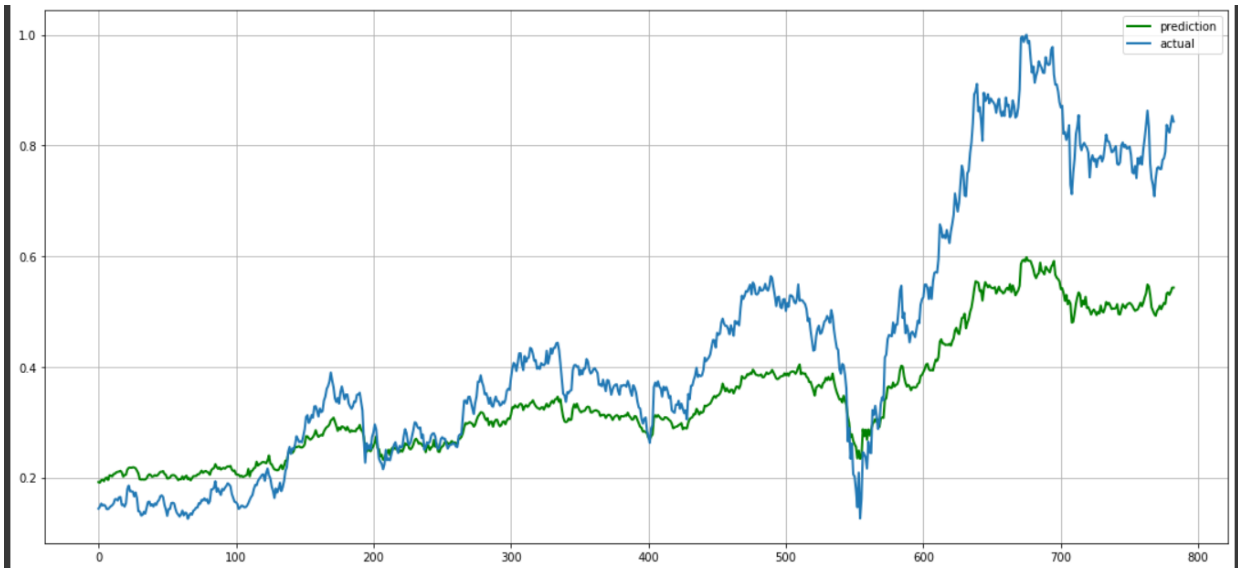


For alpha=50,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=50)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.15529008275857564
0.5583053726916278
```

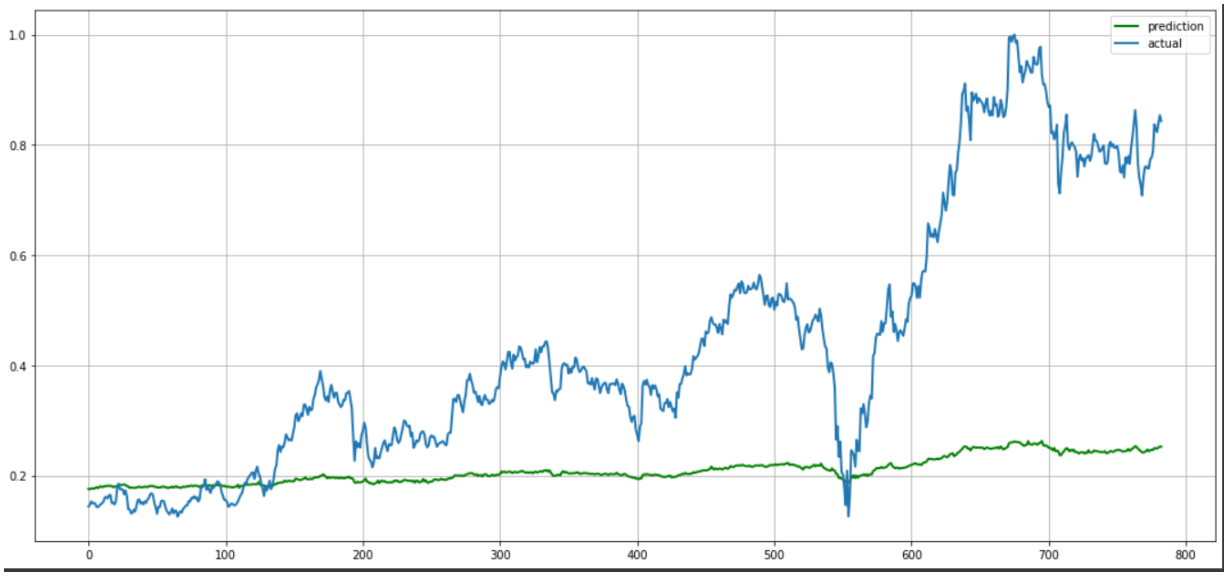


For alpha=500,

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=500)
model_ridge.fit(x_train,y_train)

preds2 = model_ridge.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds2)))
print(r2_score(y_test,preds2))
```

```
0.30954460441577786
-0.7550162842692774
```

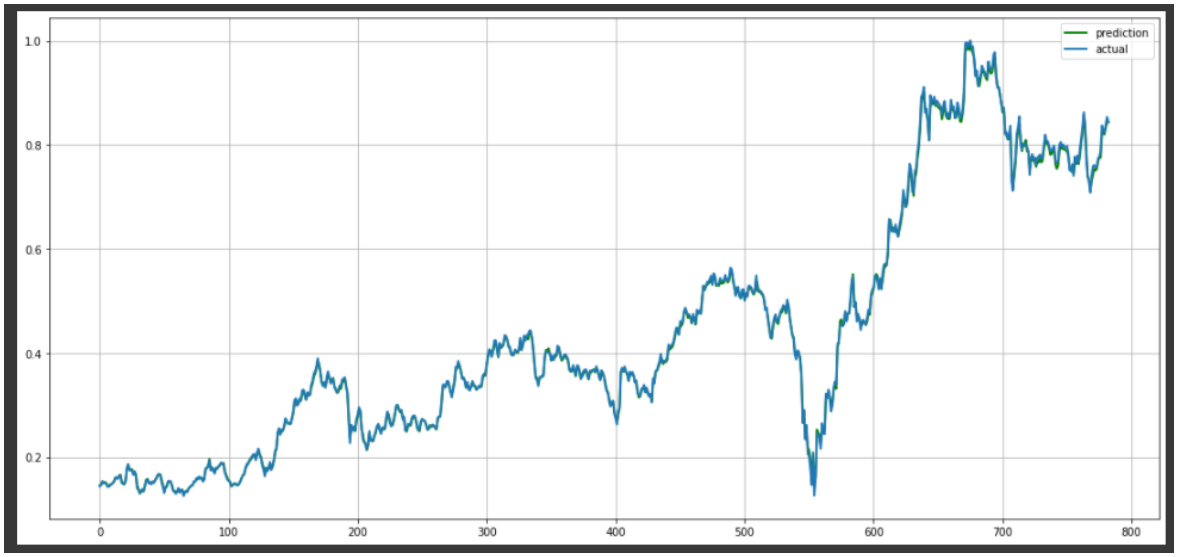


We observed that on varying the value of alpha from 0.05 to 500,the R-squared value decreases from 0.9999 to -0.755.

LASSO REGRESSION

```
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.00551296371458451
0.9994433208673841
```



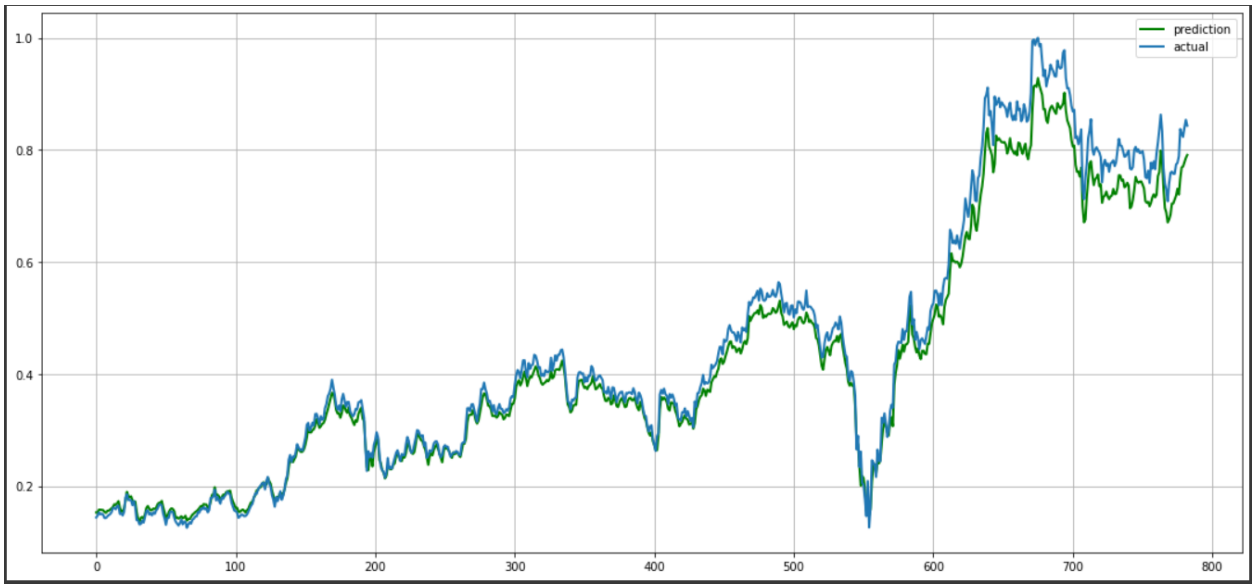
On varying the alpha to find it's optimum value-

For alpha = 0.001

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=0.001)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.03313634905515512
0.9798885220761862
```

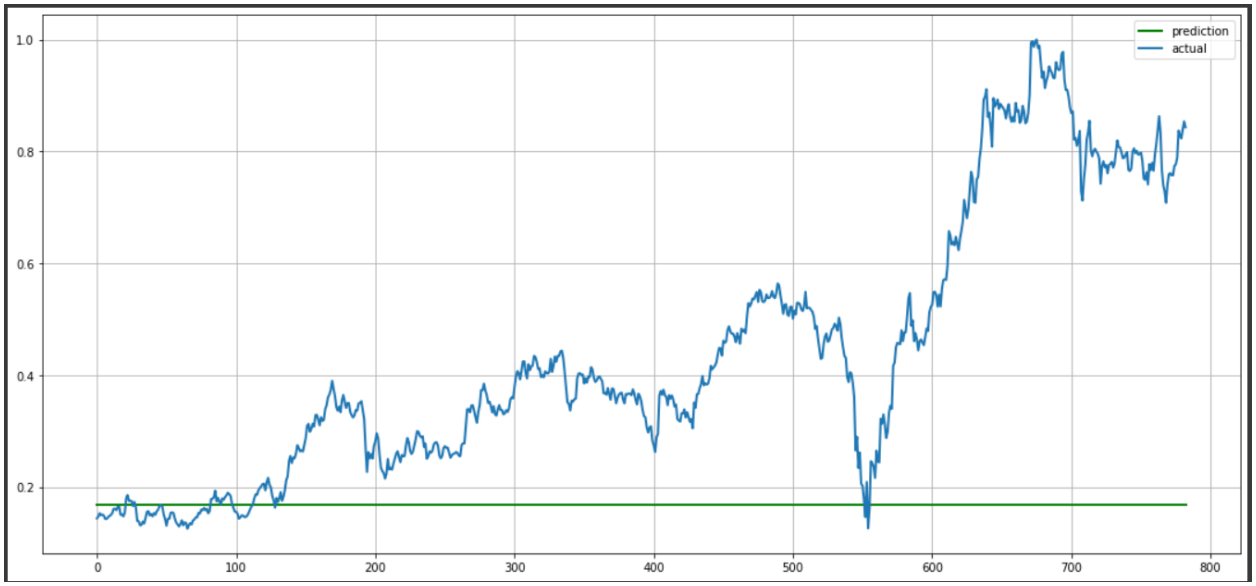


For alpha = 0.1

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=0.1)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.35561853148594136
-1.3163460554563176
```

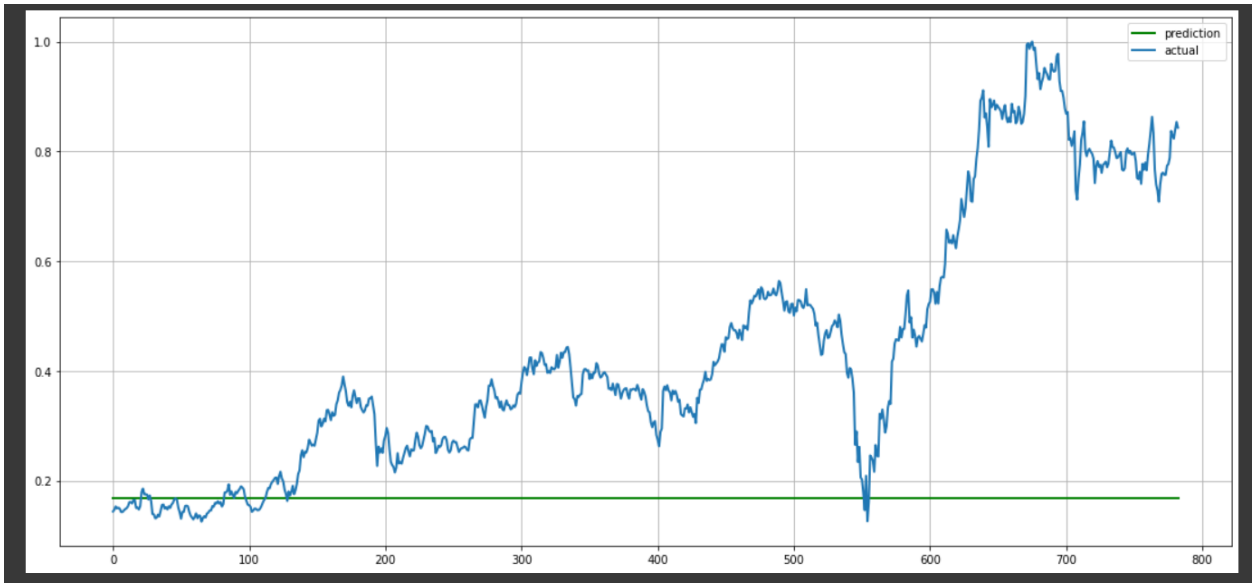


For alpha = 10

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=10)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

```
0.35561853148594136
-1.3163460554563176
```

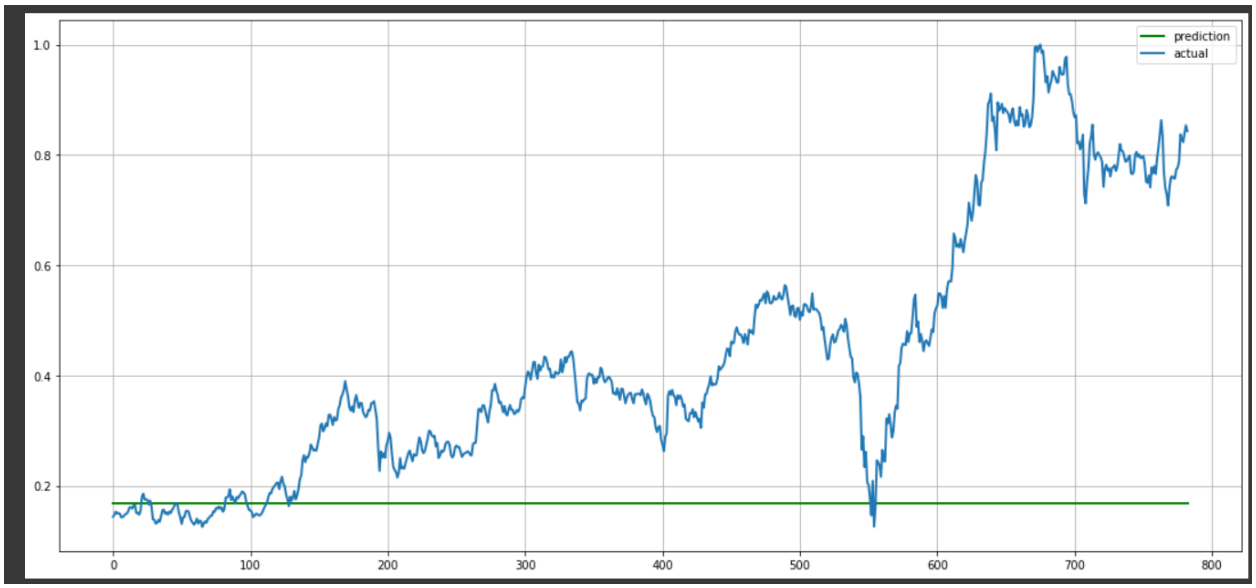


For alpha = 100

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=100)
model_lasso.fit(x_train,y_train)

preds4 = model_lasso.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, preds4)))
print(r2_score(y_test,preds4))
```

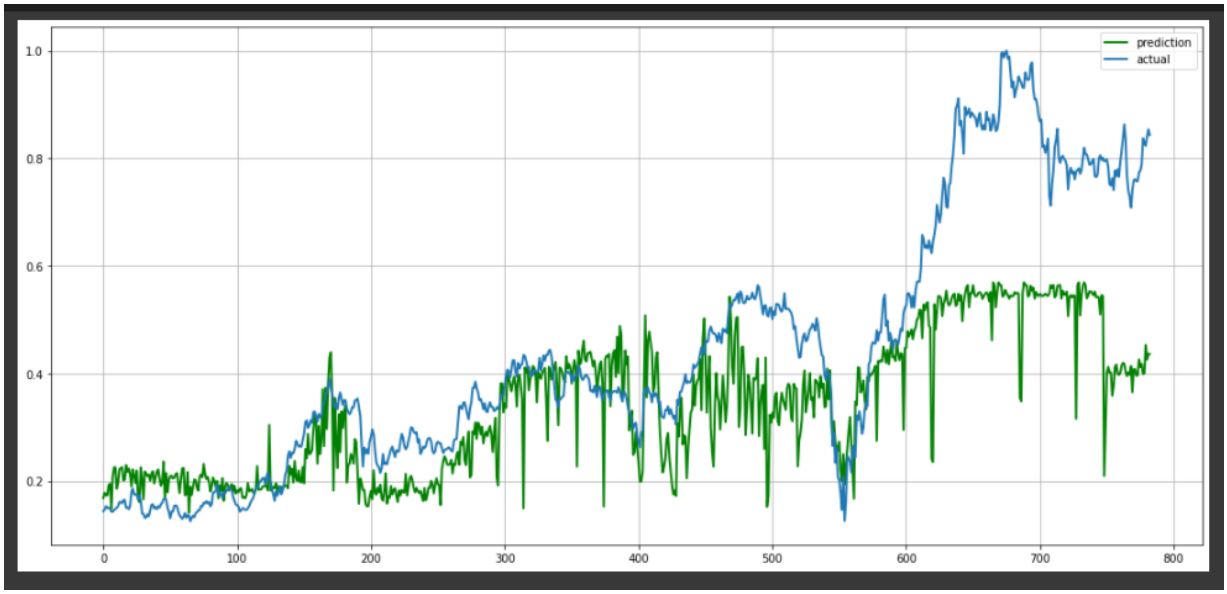
```
0.35561853148594136
-1.3163460554563176
```



KNN

```
print(np.sqrt(mean_squared_error(y_test, pred)))
print(r2_score(y_test,pred))
```

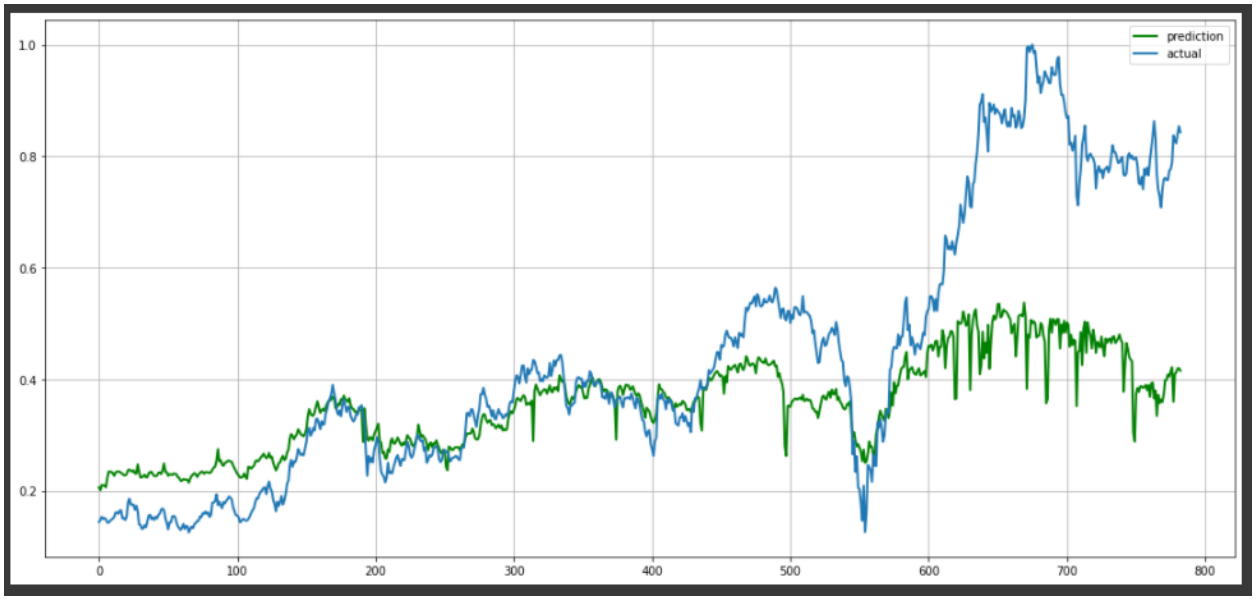
```
0.1707893252551876
0.4657357441857316
```



SVM

```
print(np.sqrt(mean_squared_error(y_test, pred5)))
print(r2_score(y_test,pred5))
```

```
0.1838942176339435
0.3806005422522394
```



Tried to use different kernels for SVM to verify which one of them gave the best results.

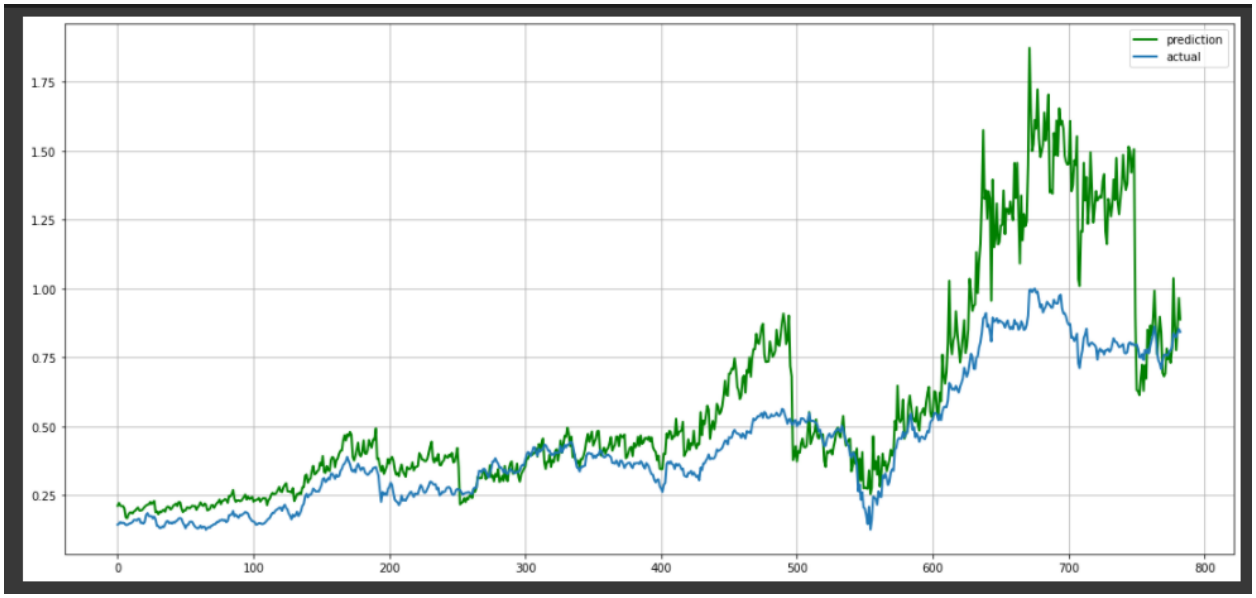
SVM(with kernel = ‘poly’)

```
from sklearn.svm import SVR
model_svm2 = SVR(kernel='poly')
model_svm2.fit(x_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='poly', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
pred6 = model_svm2.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, pred6)))
print(r2_score(y_test,pred6))
```

```
0.2290635900317974
0.03894819268767269
```



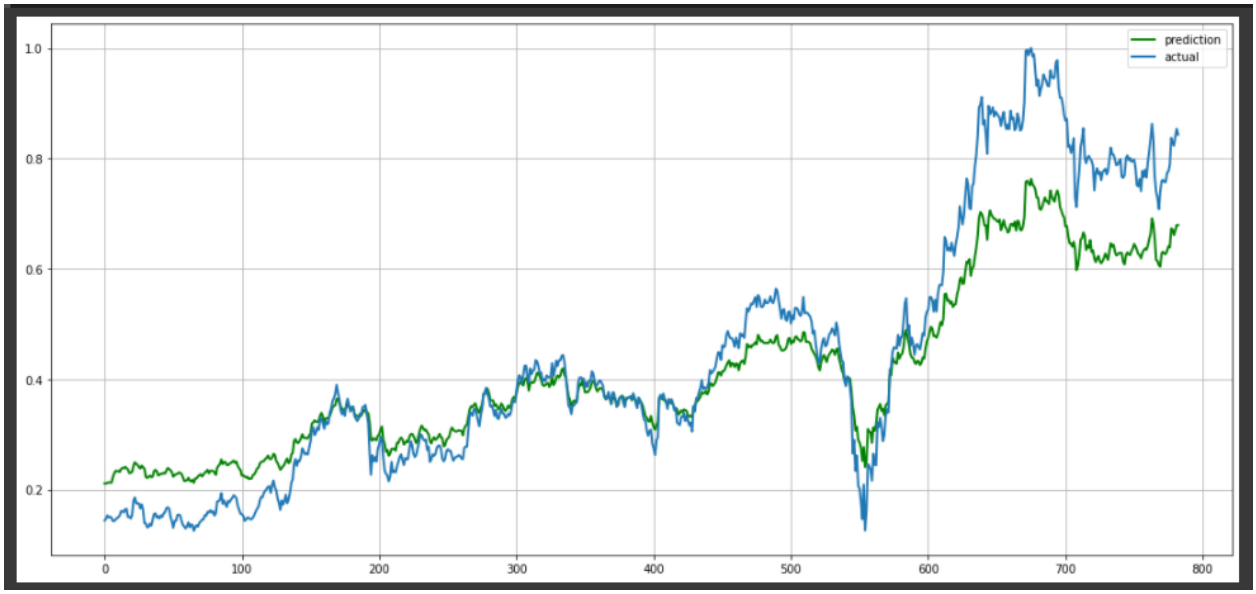
SVM(with kernel = ‘linear’)

```
from sklearn.svm import SVR
model_svm2 = SVR(kernel='linear')
model_svm2.fit(x_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
pred6 = model_svm2.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, pred6)))
print(r2_score(y_test,pred6))
```

```
0.09011716857020007
0.8512524744837324
```



MODEL	RMSE	R-squared
Linear Regression	2.70 x 10^-16	1.0000
Ridge Regression	0.0049	0.9995
Lasso Regression	0.0055	0.9994
KNN	0.1707	0.4657
SVM	0.0901	0.8512

CONCLUSION

As seen from the tables of RMSE and R-squared values, the best results are given by the **SVM model.** We attempted to achieve as accurate results as possible on the mentioned dataset. On excluding the open-price variable, the models did not perform well. However, it was observed that the Ridge Regression model performed better than Linear and Lasso performed the best out of all. On trying the same models while including the variable, the models performed significantly better. However, the accuracy of Linear Regression was 100%, which indicates overfitting.
For further improvement, employment of deep learning models is generally done for Stock price prediction.

REFERENCES

[01] Z. Hu, J. Zhu and K. Tse, “Stocks Market Prediction Using Support Vector Machine,” in the 6th International Conference on Information Management, Innovation Management and Industrial Engineering, 2013.

[02] Ashish sharma , dinesh bhuriya , upendra singh “Survey of Stock Market Prediction Using Machine Learning Approach ”International Conference on Electronics, Communication and Aerospace Technology ICECA 2017.

[03] Pushkar Khanal ,Shree Raj Shakya “Analysis and Prediction of Stock Prices of Nepal using different Machine Learning Algorithms” Department of Mechanical Engineering, Pulchowk campus, Institute of Engineering, Tribhuvan University, Nepal

[04] Fan, Alan and Marimuthu Palaniswami. Stock Selection using Support Vector Machines. IEEE 2001

[05] A. Sharma, D. Bhuriya and U. Singh, “Survey of Stock Market Prediction Using Machine Learning Approach,” in International Conference on Electronics, Communication and Aerospace Technology, 2017

[06] Aishwarya Singh, <https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/>