Sri Lanka Institute of Information Technology

Year 4 | Semester 1 | 2022

PREDICTING STOCK PRICE USING DIFFERENT MODELS AND COMPARE THE PERFORMANCE OF THE MODELS

Machine Learning (IT4060)

Assignment 02

Submitted By:

Student ID	Name
IT19139418	Rathnayaka R.M.N.A.
IT19142692	Anuththara K.G.S.N.
IT19127774	Bandara L.G.S.J.
IT19033938	Samaranayake S.L.

Table of Contents

1. Introduction to the Research problem

The stock market is notorious for its volatility, dynamic, and nonlinear nature. Multiple (macro and micro) aspects, such as politics, global economic circumstances, unforeseen occurrences, a company's financial performance, and so on, make accurate stock price forecast exceedingly difficult.

All of this, though, means that there is a lot of data to sort through. As a result, financial analysts, researchers, and data scientists continue to experiment with analytics to detect stock market patterns. This gave rise to the notion of algorithmic trading, which involves the execution of orders using automated, pre-programmed trading techniques.

Fundamental and technical assessments are at different ends of the market analysis spectrum when it comes to equities. Fundamental analysis assesses a company's stock by assessing its intrinsic worth, which includes but is not limited to tangible assets, financial statements, managerial effectiveness, strategic objectives, and customer habits; in other words, all a company's fundamentals. Not only is fundamental analysis a useful indication for long-term investment, but it also uses both historical and current data to calculate revenues, assets, expenses, liabilities, and so on. And, in general, the outcomes of basic analysis do not change in response to short-term news.

Technical analysis examines quantifiable data from stock market activities, such as stock prices, past returns, and the number of previous trades; in other words, quantitative data that can discover trading signals and record stock market movement patterns. Technical analysis, like fundamental analysis, focuses on past and present data, although it is mostly utilized for short-term trading reasons. Technical analysis results are often impacted by news because of their short-term nature. We'll use technical analysis, machine learning techniques like random forest, RNN, SVM, and linear regression to forecast stock values in this activity.

Before we start constructing the program to predict stock market prices, let's look at the data we'll be dealing with. The stock price of Tesla, Inc. [TSLA] Dataset will be examined in this section. The stock price information will be delivered in the form of a Comma Separated File (.csv), which can be accessed and studied in Excel or a Spreadsheet.

2. Introduction to the used Data set

In this research, we utilized Tesla stock values from June 29, 2010, to March 17, 2017. The dataset has 1693 items and seven columns. For respectable days, this information comprises the open price, closing price, maximum price, lowest price, and volume.

Column Name	Data Type	Description
Date	object (String)	The date
Open	float64	The opening price of the stock
High	float64	The high price of that day
Low	float64	The low price of that day
Close	float64	The closed price of that day
Volume	int64	The number of stocks traded during that day
Adj Close	float64	The stock's closing price has been amended to include any distributions/corporate actions that occur before next day's open

The URL for the dataset as below

https://www.kaggle.com/datasets/timoboz/tesla-stock-data-from-2010-to-2020

3. Methodology

3.1. Data Analyzing and Visualizing

Before doing anything with the data it is better to have an analysis of them.

- Count the number of elements present in the dataset.

```
dataSet.size

11844
```

- Get the size of the data. Output will give the total number of 1692 data rows and 7 total columns with sufficient amount of data to train the model.

- Visualize the data frame object which has the null values and get the summation of null value

```
dataSet.isnull().sum()

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

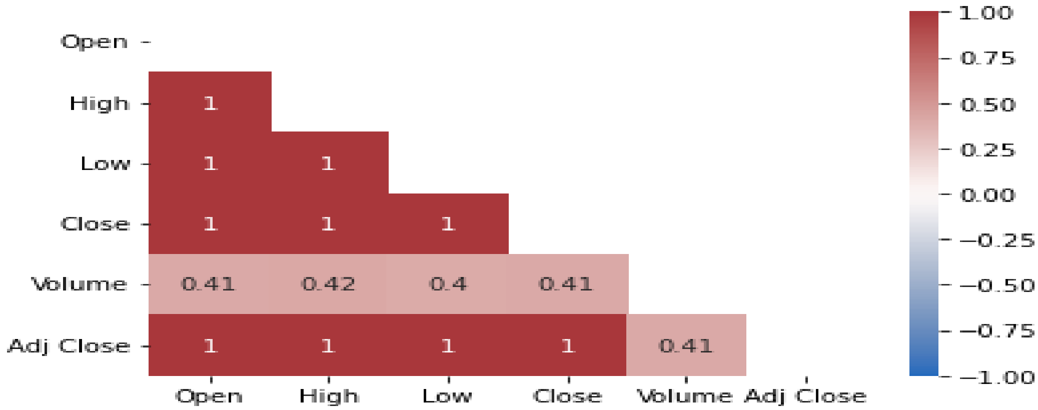
- Getting a concise summary of data

```
dataSet.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1692 entries, 0 to 1691
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1692 non-null   datetime64[ns]
 1   Open    1692 non-null   float64
 2   High    1692 non-null   float64
 3   Low     1692 non-null   float64
 4   Close   1692 non-null   float64
 5   Volume  1692 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 79.4 KB
```

- Remove unwanted data column

```
dataSet.drop('Adj Close' ,axis = 1 ,inplace = True)
```

- Get the Correlation Matrix. This shows the linear correlations of each feature. Correlation Matrix as a heat map shows below. Map is plots only to show the lower half of the matrix to reduce the redundancy and increases the understatement. The Correlation Matrix shows co-relation between numerical features. Usage of the Correlation Matrix makes it easier to understand the relationship between the used dataset.

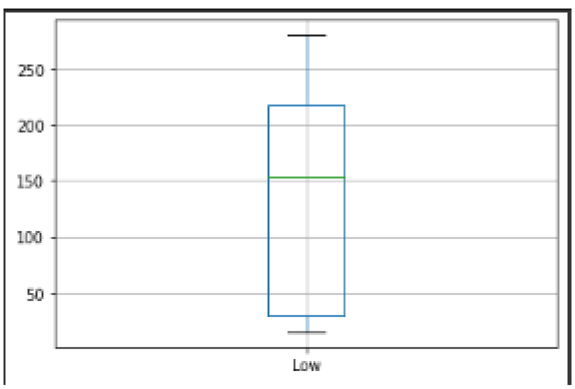


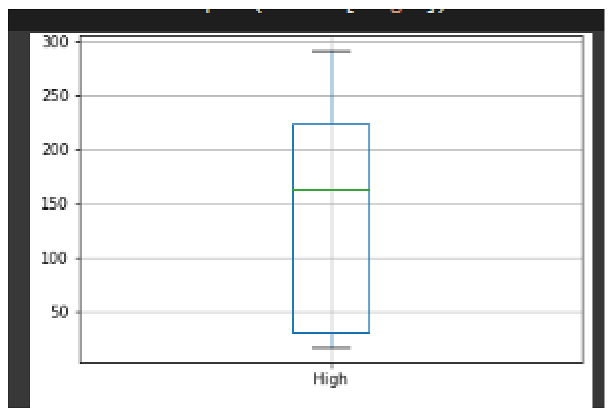
- Histogram per each numerical column.
- Get the statistics per each column.

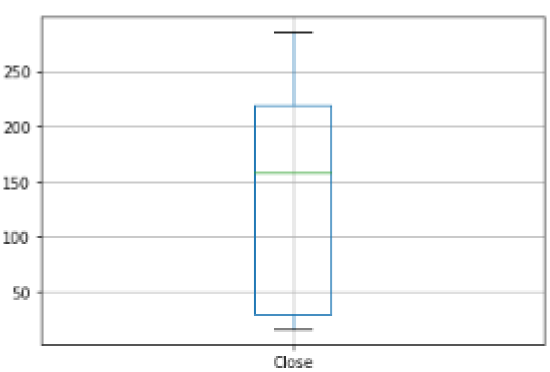
```
dataSet.describe()
```

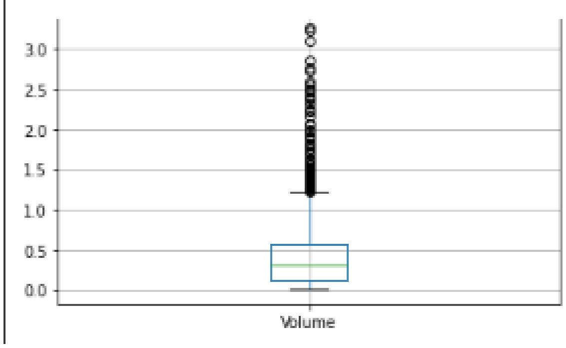
	Open	High	Low	Close	Volume	Adj Close
count	1692.000000	1692.000000	1692.000000	1692.000000	1.692000e+03	1692.000000
mean	132.441572	134.769698	129.996223	132.428658	4.270741e+06	132.428658
std	94.309923	95.694914	92.855227	94.313187	4.295971e+06	94.313187
min	16.139999	16.629999	14.980000	15.800000	1.185000e+05	15.800000
25%	30.000000	30.650000	29.215000	29.884999	1.194350e+06	29.884999
50%	156.334999	162.370002	153.150002	158.160004	3.180700e+06	158.160004
75%	220.557495	224.099999	217.119999	220.022503	5.662100e+06	220.022503
max	287.670013	291.420013	280.399994	286.040009	3.716390e+07	286.040009

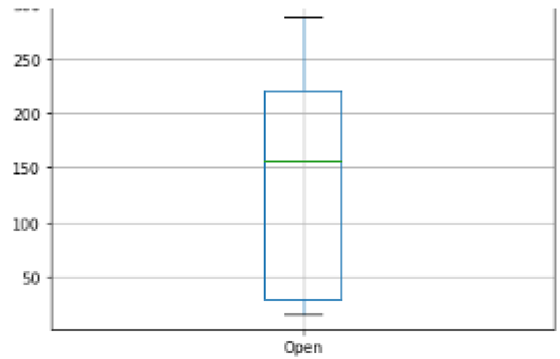
- Histogram per each numerical column.











- Get the nth number of rows. This helps to quickly go through dataset and verify whether the data type is as per required

```
dataSet.head(20)
```

	Date	Open	High	Low	Close	Volume
0	2010-06-29	19.000000	25.000000	17.540001	23.889999	18766300
1	2010-06-30	25.790001	30.420000	23.299999	23.830000	17187100
2	2010-07-01	25.000000	25.920000	20.270000	21.959999	8218800
3	2010-07-02	23.000000	23.100000	18.709999	19.200001	5139800
4	2010-07-06	20.000000	20.000000	15.830000	16.110001	6866900
5	2010-07-07	16.400000	16.629999	14.980000	15.800000	6921700
6	2010-07-08	16.139999	17.520000	15.570000	17.459999	7711400
7	2010-07-09	17.580000	17.900000	16.549999	17.400000	4050600
8	2010-07-12	17.950001	18.070000	17.000000	17.049999	2202500
9	2010-07-13	17.389999	18.639999	16.900000	18.139999	2680100
10	2010-07-14	17.940001	20.150000	17.760000	19.840000	4195200
11	2010-07-15	19.940001	21.500000	19.000000	19.889999	3739800
12	2010-07-16	20.700001	21.299999	20.049999	20.639999	2621300
13	2010-07-19	21.370001	22.250000	20.920000	21.910000	2486500
14	2010-07-20	21.850000	21.850000	20.049999	20.299999	1825300
15	2010-07-21	20.660000	20.900000	19.500000	20.219999	1252500
16	2010-07-22	20.500000	21.250000	20.370001	21.000000	957800
17	2010-07-23	21.190001	21.559999	21.059999	21.290001	653600
18	2010-07-26	21.500000	21.500000	20.299999	20.950001	922200

3.2. Data Cleaning

Data cleaning is related to fixing, removing, updating or replacing the parts of data that are incorrect, incomplete, unreliable, duplicated. Incorrectly formatted, or unavailable. This acts as one of the basic parts of a model training process which increases productivity and reduces the overall cost.

3.2.1 Used Data Cleaning methods

- Dropping the irrelevant column which is required to neglect from analysis

```
dataSet.drop('Adj Close' ,axis = 1 ,inplace = True)
```

- Dealing with missing values can be a misleading factor when it comes to predicting

```
dataSet.isnull().sum()

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

3.3. Data Pre-Processing

Data pre-processing is practiced while manipulating data before its usage of them in order to make sure high performance or enhance the mod

3.4. Model Training

2.4.1. Train-Test Data Splitting

Dataset will be divided into two parts Training data and test data. Splitting arrays into random train and test subset were done .1269 records of data will be training data and 423 records of data will be test data.

```
x = dataSet[['Open','High','Low','Volume']]
y = dataSet['Close']

X_train, X_test, Y_train, Y_test =  train_test_split(x ,y ,random_state=0)
```

```
[ ] len(X_train)
1269
[ ] len(X_test)
423
```

2.4.2. Used Algorithms

The most common and effective way to cope with the prediction of the stock price is focused on Machine Learning. Many soft computing methods such as SVM, ANN, BPNN, and LSTM were used frequently for the stock market prediction with different accuracy and result. Therefore, 4 machine learning models were trained using 4 algorithms in order to compare and select the most accurate model for this stock prediction

2.4.2.1 Linear Regression

Linear regression is a common supervised machine learning algorithm which is used to train data using a set of training data and output. It basically quantifies the relationship of the predictor variable with an outcome. Linear regression basically focuses on finding the best-fitting line for the input data. When it comes to determining relationships, linear regression tries to map one independent variable (x-value) to one dependent variable (y-value). Linear regression is used to estimate the value of a dependent variable (y) depending on the value of an independent Variable (x). When it comes to predicting assumptions were made believing the output values for the input that are not shown in the dataset will befall on the line

Code snippet for training the linear regression model.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
reg = LinearRegression()
```

```
reg.fit(X_train, Y_train)
```

```
LinearRegression()
```

2.4.1.1 Long short term memory

This is an artificial neural network which mostly uses sequential data or time series data. Since they use their memory as in taking information from prior inputs to influence current inputs it utilizes the provided training data for the learning. So, it is a state of Art algorithm where it memorizes previous inputs. It performed the function on every input and produced the output influenced by past computation. Once output was generated it was sent back to the recurrent network

LSTM unit is a recurrent unit, that is, a unit (or neuron) that contains cyclic connections, so an LSTM neural network is a recurrent neural network (RNN)**.** LSTM builds with the cell as in a memory part and three regulators called gates. Three gates are the input gate, output gate, and forget gate.

Code snippet for training the LSTM model.

```
Regressor = Sequential()

#Add first LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 60, return_sequences = True, input_shape = (x__train.shape[1], 5)))
Regressor.add(Dropout(0.2))

#Add second LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 60, return_sequences = True))
Regressor.add(Dropout(0.2))

#Add third LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 80, return_sequences = True))
Regressor.add(Dropout(0.2))

#Add fourth LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 120))
Regressor.add(Dropout(0.2))

#Add output layer
Regressor.add(Dense(units = 1))
```

2.4.1.2 Support Vector Machine(SVM)

SVM is a supervised learning algorithm that is mostly based on a statistical learning approach. When it comes to prediction SVM look for the maximum margin of the provided information or data set so that identification of hyperplane can be achieved. Hyperplane maybe chooses edges or the separators that are responsible to classify the identified data points. The measurement is built on the premise of the highlighted features of the numbers. This hyperplane will be able to optimally separate the classes. Vectors or cases that represent the hyperplane are the support vectors

Code snippet for train the SVM model.

```
def predict_prices(Open_date, open_price, x):
    Open_date = np.reshape(Open_date,(len(Open_date), 1))
    x = np.reshape(x,(len(x), 1))

    svr_lin  = SVR(kernel='linear', C=1e3)
    svr_poly = SVR(kernel='poly', C=1e3, degree=2)
    svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)

    svr_lin .fit(Open_date, open_price)
    svr_poly.fit(Open_date, open_price)
    svr_rbf.fit(Open_date, open_price)

    plt.scatter(Open_date, open_price, c='k', label='Data')
    plt.plot(Open_date, svr_lin.predict(Open_date), c='g', label='Linear model')
    plt.plot(Open_date, svr_rbf.predict(Open_date), c='r', label='RBF model')
    plt.plot(Open_date, svr_poly.predict(Open_date), c='b', label='Polynomial model')

    plt.xlabel('Date')
    plt.ylabel('Open')
    plt.title('Support Vector Regression')
    plt.legend()
    plt.show()

    return svr_rbf.predict(x)[0], svr_lin.predict(x)[0], svr_poly.predict(x)[0]
```

2.4.1.1 Random forest

In Random forest random vector is produced, which is independent of previous random vectors but has the same distribution at the same time a tree is constructed using the training set and resulting in a classifier, where x is an input vector. The random selection is made up of a set of independent random numbers ranging from 1 to K. The usage of random selections in tree building determines its character and dimensions. Once many trees have been produced, they vote on the most popular class.

Code snippet for training the Random Forest model.

```
rfc = RandomForestClassifier(n_estimators=16)
```

```
rfc.fit(X_train, Y_train)
```

```
RandomForestClassifier(n_estimators=16)
```

4. Model Evaluating and Discussion

4.1. R squared score

The proportion of the variation of the dependent variable is explained by the R Squared value, which has range from 0 to 1. A higher value indicates better models. Basically, R squared value gives whether the data and predictions are biased.

Model	**R Squared Value**
RNN	0.8581662301798857
Random Forest	0.9949031600407747
SVM	0.9227211892498876
Linear Regression	0.999703484441961

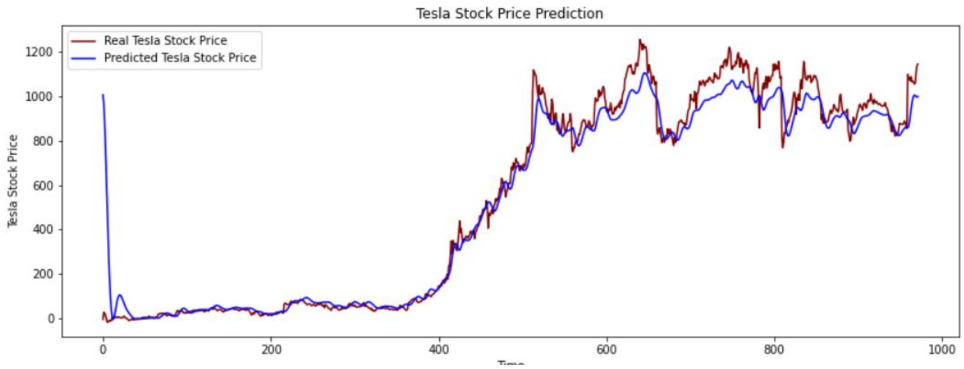
The Linear Regression algorithm produces the highest value of R Squared, therefore it gives better prediction than other algorithms.

4.2. Fitness of the models

As the second method, the fitness of the models was tested using graphs that were comparing actual and predicted values.

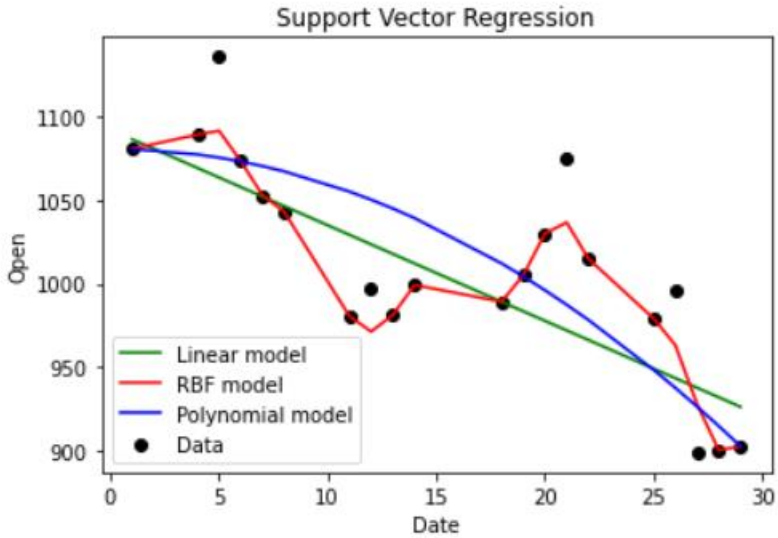
1. RNN

```
plt.figure(figsize=(14,5))
plt.plot(y__test, color = 'maroon', label = 'Real Tesla Stock Price')
plt.plot(y__pred, color = 'blue', label = 'Predicted Tesla Stock Price')
plt.title('Tesla Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Tesla Stock Price')
plt.legend()
plt.show()
```



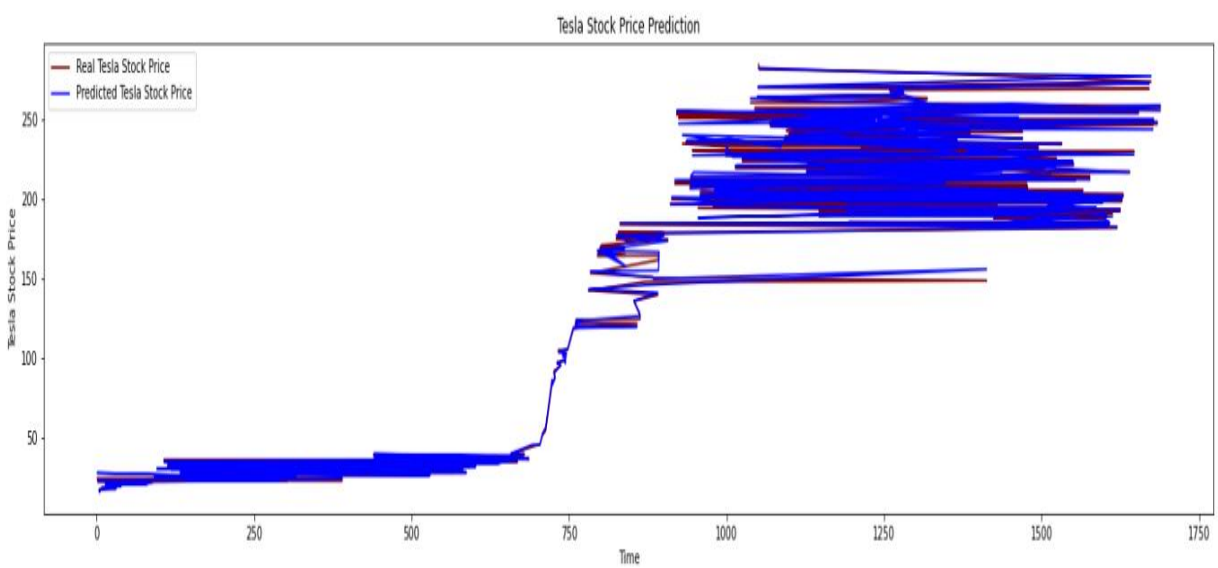
2. SVM

```
def predict_prices(Open_date, open_price, x):
    Open_date = np.reshape(Open_date,(len(Open_date), 1))
    x = np.reshape(x,(len(x), 1))

    svr_lin  = SVR(kernel='linear', C=1e3)
    svr_poly = SVR(kernel='poly', C=1e3, degree=2)
    svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)

    svr_lin .fit(Open_date, open_price)
    svr_poly.fit(Open_date, open_price)
    svr_rbf.fit(Open_date, open_price)

    plt.scatter(Open_date, open_price, c='k', label='Data')
    plt.plot(Open_date, svr_lin.predict(Open_date), c='g', label='Linear model')
    plt.plot(Open_date, svr_rbf.predict(Open_date), c='r', label='RBF model')
    plt.plot(Open_date, svr_poly.predict(Open_date), c='b', label='Polynomial model')

    plt.xlabel('Date')
    plt.ylabel('Open')
    plt.title('Support Vector Regression')
    plt.legend()
    plt.show()

    return svr_rbf.predict(x)[0], svr_lin.predict(x)[0], svr_poly.predict(x)[0]
```



3. Linear Regression

```
plt.figure(figsize=(24,5))
plt.plot(sd['Actual Price'], color = 'maroon', label = 'Real Tesla Stock Price')
plt.plot(sd['Predicted Price'], color = 'blue', label = 'Predicted Tesla Stock Price')
plt.title('Tesla Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Tesla Stock Price')
plt.legend()
plt.show()
```



	Actual Price	Predicted Price
6	17.459999	16.689894
568	28.490000	28.229043
229	28.980000	28.000102
124	30.090000	31.253574
642	34.099998	34.061340
446	35.000000	34.933071
646	36.000000	35.959804
692	43.930000	45.089164
891	139.339996	140.713590
1608	183.770004	185.106451
1628	198.690002	200.629306
1397	204.990005	202.532384
996	206.419998	203.932974
1014	219.460007	220.307066
1505	219.610001	220.689650
1129	222.600006	221.879469
1110	246.720001	243.620755
1688	258.000000	255.654999
1279	263.820007	265.429280
1281	266.149994	267.090753

According to the above graphs, we can say these 3 models have proper fitness with this dataset and scenario and they can estimate accurate outputs.

5. Future Work & Limitations

Limitations

Basically, stock market prediction is an act of determining the future values of company stock, so successful prediction of stock could provide significant profit. The stock market reflects the economic condition of the country so that relationships between one another remain interchangeable. The stock market volatility during the pandemic world and continuous disruption caused by the ongoing war creates a stressful situation. Regular political and economic events may have a major impact on the stock price. However, these should be quantified and should add to the prediction model. Apart from this, there are factors like psychological patterns which is very hard to capture using machine learning models. This constant volatility limit the access of Machin learning when it comes to prediction. Since volatilities are caused by human the unpredictability of human and their decision can limit the prediction. Usually Stock market changes can be caused by number of parameters. With huge number of parameters only few of them can be quantifiable in order to use in a machine learning model. Above factors can be the limitation of machine learning base stock prediction. Below factors will be the challenging points when it comes to stock prediction.

- Lack of Identification of distant relationships
- Quantity of required data
- Failure of integration of Human's Behavioral effect on the market
- Lack of usage in long Term Predictions

Future Work

- Enhancing the model for better prediction accuracy
- Increase the number of parameters used to predict the stock price
- Increase the interpretability of models
- Integration of more Human Behavioral effect on the market in the models
- Minimize the models overfitting

6. Individual Contribution

Student ID	Name	Workload distribution
IT19139418	Rathnayaka R.M.N.A.	• Model creation using Linear Regression • Work on the methodology of the report • Work on the future work on the report
IT19142692	Anuththara K.G.S.N.	• Model creation using Linear Regression • Work on the model evaluation and discussion of the report
IT19127774	Bandara L.G.S.J.	• Model creation using Random Forest. • Work on the Introduction to the Research problem
IT19033938	Samaranayake S.L.	• Model creation using Support Vector Model • Work on the Introduction to the dataset

7. References

[1] A. R. Kapil, "How Is Machine Learning Used for Stock Market Prediction?," Analyti Lab , [Online]. Available: https://www.analytixlabs.co.in/blog/stock-market-prediction-using-machine-learning/. [Accessed 25 05 2022].

[2] Y. R. F. T. R. M. T. R. P. T. F. Icha Mailinda, "Stock Price Prediction During the Pandemic Period with the SVM, BPNN, and LSTM Algorithm," in *2021 4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2021.

[3] B. Ostlin, "A comparison of support vector machines and partial least squares regression on spectral data," in *Magisterial dissertation. University of Nijmegen,*.

[4] En.wikipedia.org, "Multilayer perceptron - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Multilayer_perceptron. [Accessed 26 March 2022].

[5] M. C. R. L. M. G. M. A. S. V.-C. Andre R. Fonseca, "Testing the Application of Support Vector Machine (SVM) to Technical Trading Rules," in *2021 IEEE International Systems Conference (SysCon)*, 2021.

8. Appendix

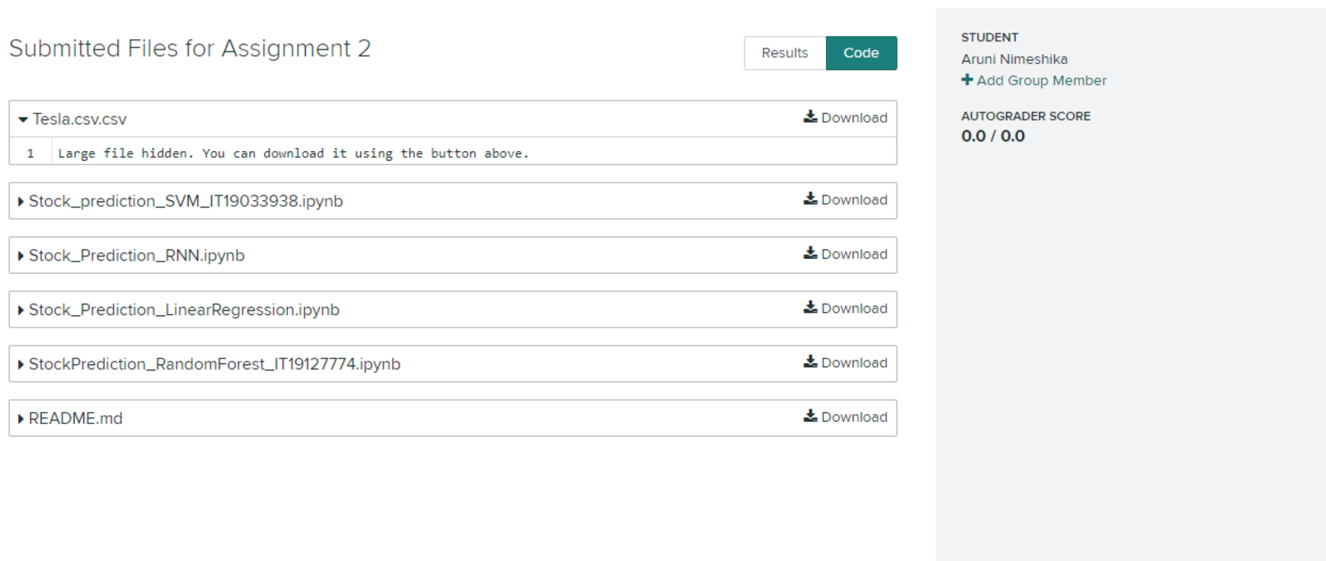
GitHub Link:

https://github.com/aruninimz/ML_Assignment2_IT19139418_IT19142692_-IT19127774_IT19033938

Video Demonstration:

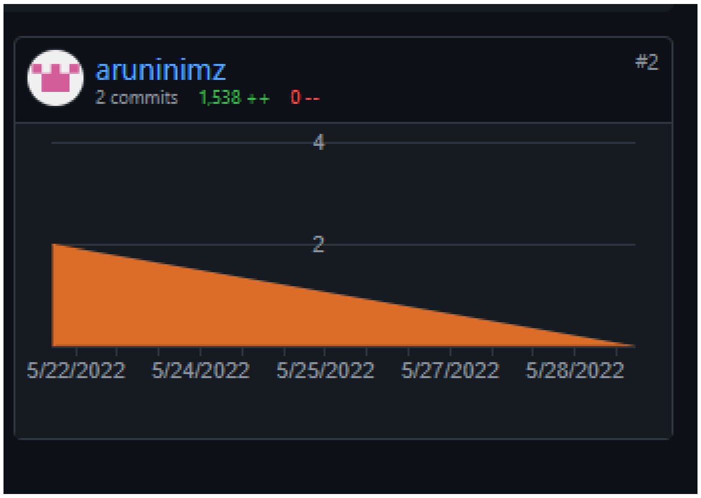
Google Drive: https://drive.google.com/file/d/1Bg_14GRtN2UD0He6ERZ-G9skz77LQD9v/view?usp=sharing

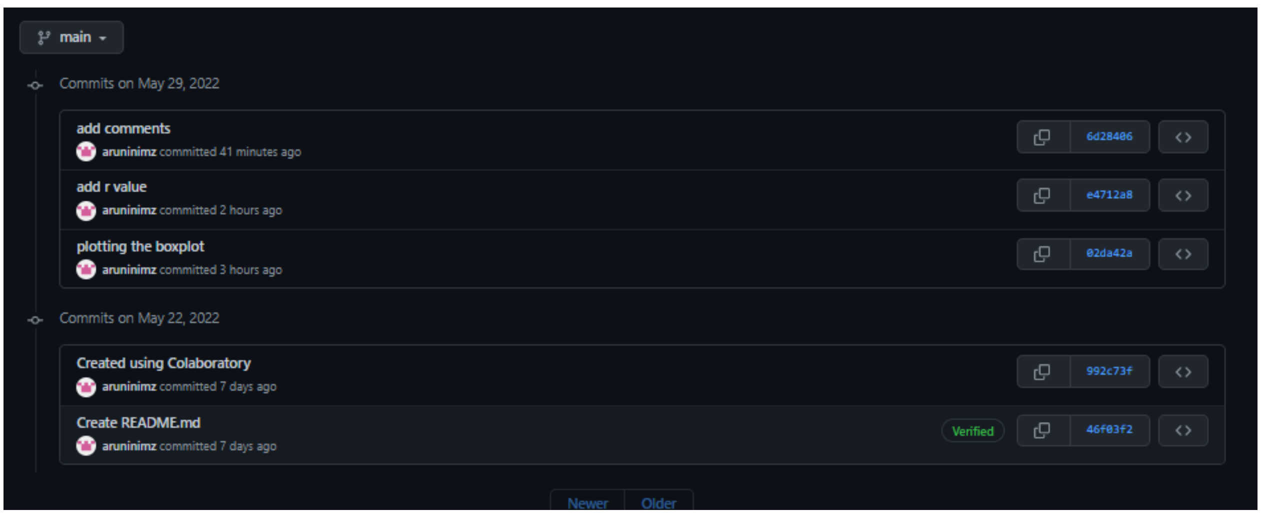
AUTOGRADER Score:



Evidence for Individual contribution – IT19139418

GitHub Commit:





Source code

```
import csv
import datetime
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

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

```
dataSet = pd.read_csv('/content/drive/MyDrive/ML Assignment2/Tesla.csv - Tesla.csv.csv')
dataSet
```

```
dataSet.drop('Adj Close' ,axis = 1 ,inplace = True)
```

```
dataSet.isnull().sum()
```

```
import seaborn as sn # for data visualization
```

```
# create seaborn heatmap
matrix = dataSet.corr().round(2)
mask = np.triu(np.ones_like(matrix, dtype=bool))
sn.heatmap(matrix, annot=True, vmax=1, vmin=-1, center=0, cmap='vlag', mask=mask)
plt.show()
```

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(dataSet['Close'])

# show plot
plt.show()
```

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(dataSet['Volume'])

# show plot
plt.show()
```

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(dataSet['Low'])

# show plot
plt.show()
```

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(dataSet['High'])

# show plot
plt.show()
```

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(dataSet['Open'])

# show plot
plt.show()
```

```python
fig, axes = plt.subplots(ncols=len(dataSet.columns), figsize=(20,10))
for col, ax in zip(dataSet, axes):
    dataSet[col].value_counts().sort_index().plot.bar(ax=ax, title=col)

plt.tight_layout()
plt.show()
```

```python
dataSet.describe()
```

```python
dataSet['Date'] = pd.to_datetime(dataSet.Date)
```

```python
dataSet.size
```

```python
dataSet.shape
```

```python
dataSet.head(20)
```

```python
dataSet.info()
```

```python
dataSet['Open'].plot(figsize=(20,10))
```

```python
x = dataSet[['Open','High','Low','Volume']]
y = dataSet['Close']
```

```python
X_train, X_test, Y_train, Y_test =  train_test_split(x ,y ,random_state=0)
```

```python
len(X_train)
```

```python
len(X_test)
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
reg = LinearRegression()
```

```python
reg.fit(X_train, Y_train)
```

```
LinearRegression()
```

```python
predict_price = reg.predict(X_test)
```

```python
from sklearn.metrics import r2_score
```

```python
r2_score(Y_test, predict_price)
```

```
predict_price, Y_test
```

```
print(X_test)
```

```
dframe = pd.DataFrame({'Actual Price':Y_test, 'Predicted Price' : predict_price})
```

```
sd=dframe.sort_values(by='Actual Price')
```

```
sd['Actual Price']
```

```
5        15.800000
4        16.110001
6        17.459999
31       17.600000
9        18.139999
           ...
1050    269.700012
1674    273.510010
1262    279.720001
1052    281.190002
1051    284.119995
Name: Actual Price, Length: 423, dtype: float64
```

```
print(sd)
```

```
      Actual Price  Predicted Price
5        15.800000        15.396379
4        16.110001        16.771326
6        17.459999        16.689894
31       17.600000        17.589502
9        18.139999        18.015058
...            ...              ...
1050    269.700012       270.305547
1674    273.510010       276.911968
1262    279.720001       279.107467
1052    281.190002       282.100091
1051    284.119995       282.007728

[423 rows x 2 columns]
```

```
dframe.head(20).sort_values(by='Actual Price')
```

	Actual Price	Predicted Price
6	17.459999	16.689894
568	28.490000	28.229043
229	28.980000	28.000102
124	30.090000	31.253574
642	34.099998	34.061340
446	35.000000	34.933071
646	36.000000	35.959804
692	43.930000	45.089164
891	139.339996	140.713590
1608	183.770004	185.106451
1628	198.690002	200.629306
1397	204.990005	202.532384
996	206.419998	203.932974
1014	219.460007	220.307066
1505	219.610001	220.689650

```
reg.score(X_test, Y_test)
```

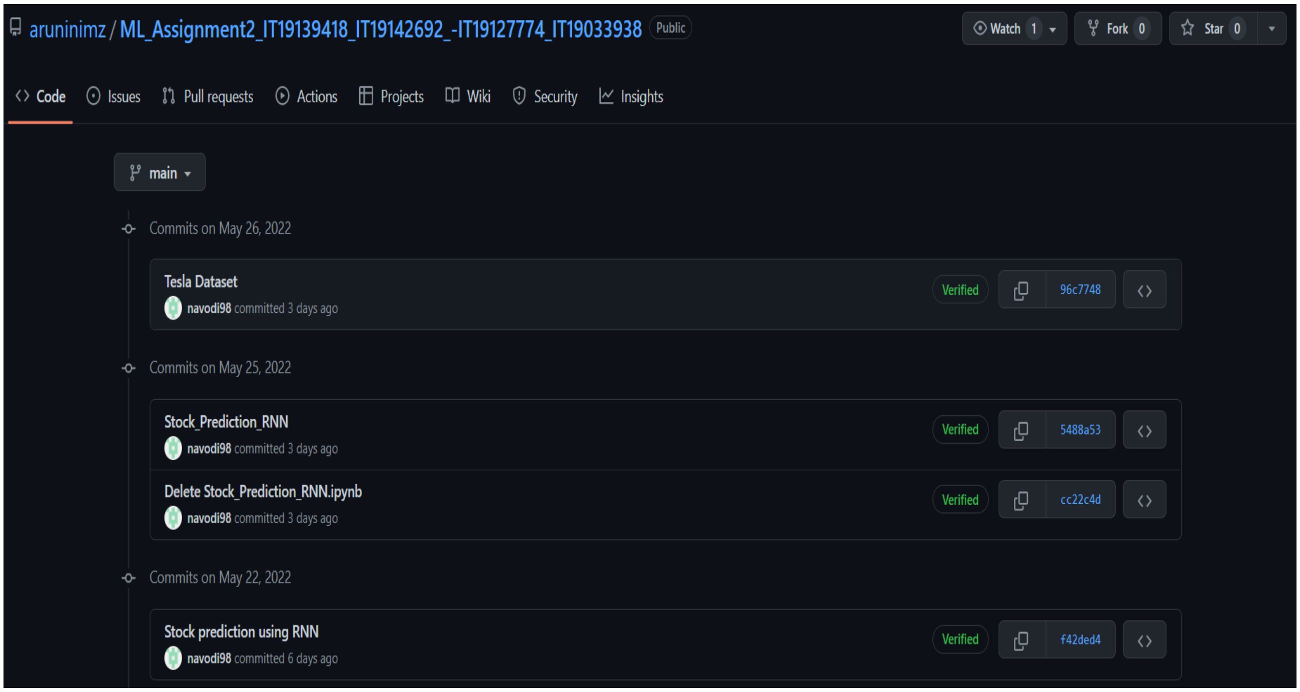
```
0.999703484441961
```

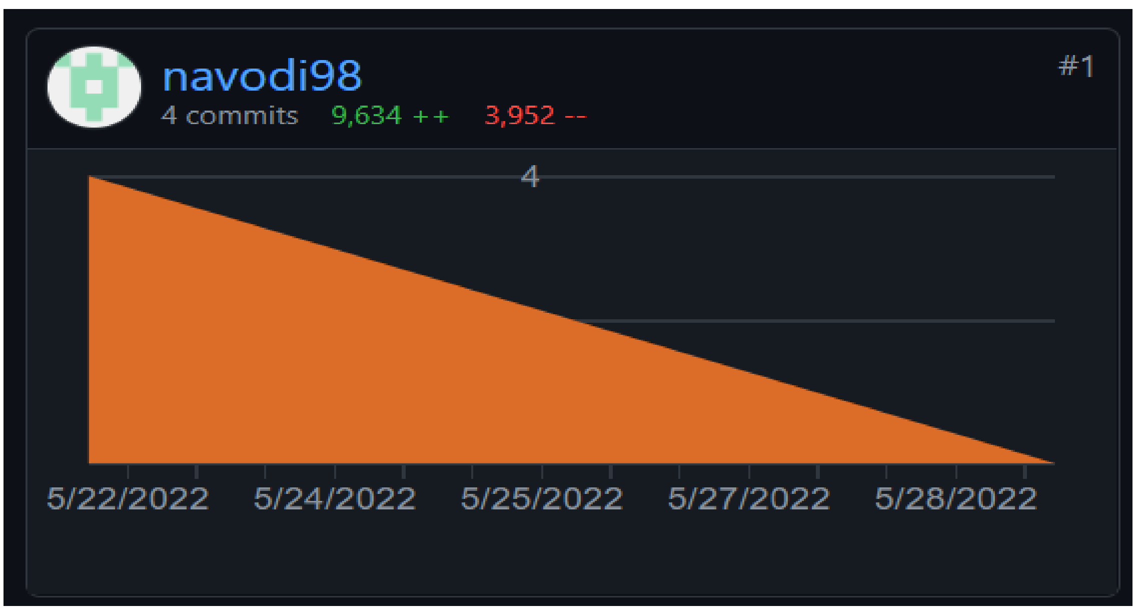
```
print('Absolute Error:', metrics.mean_absolute_error(Y_test,predict_price))
print('Squared Error:', metrics.mean_squared_error(Y_test,predict_price))
print('Root Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,predict_price)))
```

```
Absolute Error: 1.0928260736454647
Squared Error: 2.659515984376385
Root Squared Error: 1.6308022517694736
```

Evidence for Individual contribution – IT19142692

GitHub Commit:





Source code

In [1]:
```
from google.colab import files
```

In [2]:
```
uploaded=files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Tesla.csv.csv to Tesla.csv.csv
```

In [3]:
```
#Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

%matplotlib inline
```

In [4]:
```
data_set = pd.read_csv("Tesla.csv.csv", date_parser = True)
data_set.tail()
```

Out[4]:

	Date	Open	High	Low	Close	Volume	Adj Close
1687	3/13/2017	244.820007	246.850006	242.779999	246.169998	3010700	246.169998
1688	3/14/2017	246.110001	258.119995	246.020004	258.000000	7575500	258.000000
1689	3/15/2017	257.000000	261.000000	254.270004	255.729996	4816600	255.729996
1690	3/16/2017	262.399994	265.750000	259.059998	262.049988	7100400	262.049988
1691	3/17/2017	264.000000	265.329987	261.200012	261.500000	6475900	261.500000

In [5]:

```
training_data = data_set[data_set['Date']<'2017-02-16'].copy()
training_data
```

Out[5]:

	Date	Open	High	Low	Close	Volume	Adj Close
66	10/1/2010	20.690001	20.750000	20.309999	20.600000	597700	20.600000
67	10/4/2010	20.430000	21.170000	20.299999	20.990000	643600	20.990000
68	10/5/2010	21.150000	21.280001	21.010000	21.120001	332000	21.120001
69	10/6/2010	21.059999	21.260000	20.320000	20.459999	313400	20.459999
70	10/7/2010	20.570000	20.639999	20.340000	20.430000	141000	20.430000
...	...	...	...	...	...	...	...
1674	2/22/2017	280.309998	283.450012	272.600006	273.510010	8081400	273.510010
1675	2/23/2017	264.000000	264.660004	255.559998	255.990005	14867000	255.990005
1676	2/24/2017	252.660004	258.250000	250.199997	257.000000	8160500	257.000000
1677	2/27/2017	248.169998	248.360001	242.009995	246.229996	11432900	246.229996
1678	2/28/2017	244.190002	251.000000	243.899994	249.990005	6065600	249.990005

720 rows × 7 columns

In [6]:

```
testing_data = data_set[data_set['Date']>='2017-02-16'].copy()
testing_data
```

Out[6]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.000000	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.420000	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.920000	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.100000	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.000000	15.830000	16.110001	6866900	16.110001

In [6]:

```
testing_data = data_set[data_set['Date']>='2017-02-16'].copy()
testing_data
```

Out[6]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.000000	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.420000	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.920000	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.100000	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.000000	15.830000	16.110001	6866900	16.110001
...	...	...	...	...	...	...	...
1687	3/13/2017	244.820007	246.850006	242.779999	246.169998	3010700	246.169998
1688	3/14/2017	246.110001	258.119995	246.020004	258.000000	7575500	258.000000
1689	3/15/2017	257.000000	261.000000	254.270004	255.729996	4816600	255.729996
1690	3/16/2017	262.399994	265.750000	259.059998	262.049988	7100400	262.049988
1691	3/17/2017	264.000000	265.329987	261.200012	261.500000	6475900	261.500000

972 rows × 7 columns

In [7]:

```
trainingData = training_data.drop(['Date', 'Adj Close'], axis = 1)
trainingData.head()
```

Out[7]:

	Open	High	Low	Close	Volume
66	20.690001	20.750000	20.309999	20.600000	597700
67	20.430000	21.170000	20.299999	20.990000	643600
68	21.150000	21.280001	21.010000	21.120001	332000
69	21.059999	21.260000	20.320000	20.459999	313400
70	20.570000	20.639999	20.340000	20.430000	141000

In [8]:

```
#Dataset Normalization
Scale = MinMaxScaler()
trainingData = Scale.fit_transform(trainingData)
trainingData
```

Out[8]:

```
array([[0.00203729, 0.00175957, 0.00119871, 0.00210785, 0.014716  ],
       [0.00103786, 0.00333196, 0.00116004, 0.0036025 , 0.01612557],
       [0.0038055 , 0.00374378, 0.00390549, 0.00410072, 0.00655648],
       ...,
       [0.89371519, 0.89090632, 0.8901435 , 0.90809792, 0.2469659 ],
       [0.87645589, 0.85388038, 0.85847418, 0.86682247, 0.34745971],
       [0.86115704, 0.86376394, 0.86578248, 0.8812325 , 0.18263254]])
```

In [9]:

```
trainingData.shape[0]
```

Out[9]:

```
720
```

In [10]:

```
#60 time stamps back to predict the future

x__train = []
y__train = []

#Range is from 60 Values to END
for a in range(60, trainingData.shape[0]):

    x__train.append(trainingData[a-60:a])

    y__train.append(trainingData[a,0])

# Convert into Numpy Array
x__train = np.array(x__train)
y__train = np.array(y__train)

print(x__train.shape)
print(y__train.shape)
```

```
(660, 60, 5)
(660,)
```

Model

In [11]:

```
Regressor = Sequential()

#Add first LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 60, return_sequences = True, input_shape = (x__train.shape[1], 5)))
Regressor.add(Dropout(0.2))

#Add second LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 60, return_sequences = True))
Regressor.add(Dropout(0.2))

#Add third LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 80, return_sequences = True))
Regressor.add(Dropout(0.2))

#Add fourth LSTM layer and dropout regularisation
Regressor.add(LSTM(units = 120))
Regressor.add(Dropout(0.2))

#Add output layer
Regressor.add(Dense(units = 1))
```

In [12]:

```
Regressor.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 60)            15840

 dropout (Dropout)           (None, 60, 60)            0

 lstm_1 (LSTM)               (None, 60, 60)            29040

 dropout_1 (Dropout)         (None, 60, 60)            0

 lstm_2 (LSTM)               (None, 60, 80)            45120

 dropout_2 (Dropout)         (None, 60, 80)            0

 lstm_3 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 186,601
Trainable params: 186,601
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

```
#Compile
Regressor.compile(optimizer='adam', loss = 'mean_squared_error')
Regressor.fit(x__train, y__train, epochs=1, batch_size=32)
```

```
21/21 [==============================] - 13s 164ms/step - loss: 0.0518
```

Out[13]:

```
<keras.callbacks.History at 0x7f1b5f783f50>
```

In [14]:

```
testing_data.head()
```

Out[14]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001

In [16]:

```
past_60_days = testing_data.tail(60)
```

In [17]:

```
dataFrame = past_60_days.append(testing_data, ignore_index = True)
dataFrame = dataFrame.drop(['Date', 'Adj Close'], axis = 1)
dataFrame.head()
```

Out[17]:

	Open	High	Low	Close	Volume
0	229.339996	233.360001	226.919998	228.490005	2889000
1	227.949997	230.759995	226.600006	230.610001	2419100
2	230.699997	235.279999	230.240005	234.789993	3070800
3	235.500000	236.630005	229.380005	230.009995	4016300
4	229.369995	229.869995	221.399994	227.199997	3934400

In [18]:

```
input = Scale.transform(dataFrame)
input
```

Out[18]:

```
array([[0.80407456, 0.79772374, 0.80012378, 0.79883492, 0.0850807 ],
       [0.7987315 , 0.7879899 , 0.79888642, 0.80695969, 0.0706503 ],
       [0.80930232, 0.80491178, 0.81296167, 0.82297928, 0.09066369],
       ...,
       [0.91039785, 0.9012017 , 0.90588151, 0.9032307 , 0.14427636],
       [0.93115509, 0.91898464, 0.92440359, 0.92745172, 0.21441075],
       [0.93730541, 0.9174122 , 0.93267865, 0.92534392, 0.19523266]])
```

```
In [19]:
x__test = []
y__test = []

for a in range(60, input.shape[0]):
    x__test.append(input[a-60:a])
    y__test.append(input[a, 0])

x__test, y__test = np.array(x__test), np.array(y__test)
x__test.shape, y__test.shape

Out[19]: ((972, 60, 5), (972,))
```

```
In [20]:
y__pred = Regressor.predict(x__test)
```

```
In [21]:
from sklearn.metrics import r2_score
```

```
In [23]:
r2_score(y__test, y__pred)

Out[23]: 0.8581662301798857
```

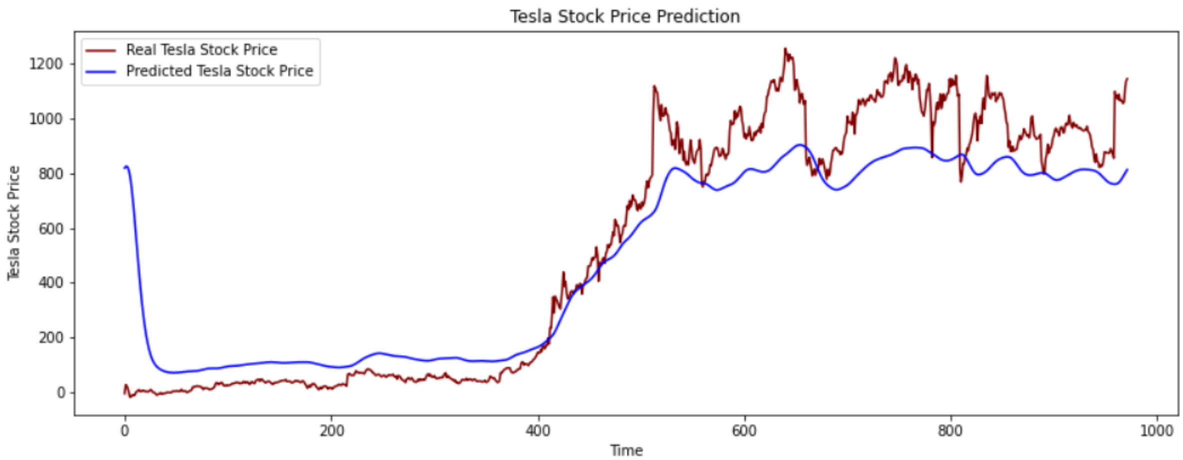
```
In [25]:
Scale.scale_

Out[25]: array([3.84393622e-03, 3.74377578e-03, 3.86682672e-03, 3.83244531e-03,
       3.07095126e-08])
```

```
In [26]:
scale = 1/8.18605127e-04
scale

Out[26]: 1221.5901990069017
```

```
In [27]:
y__pred = y__pred*scale
y__test = y__test*scale
```

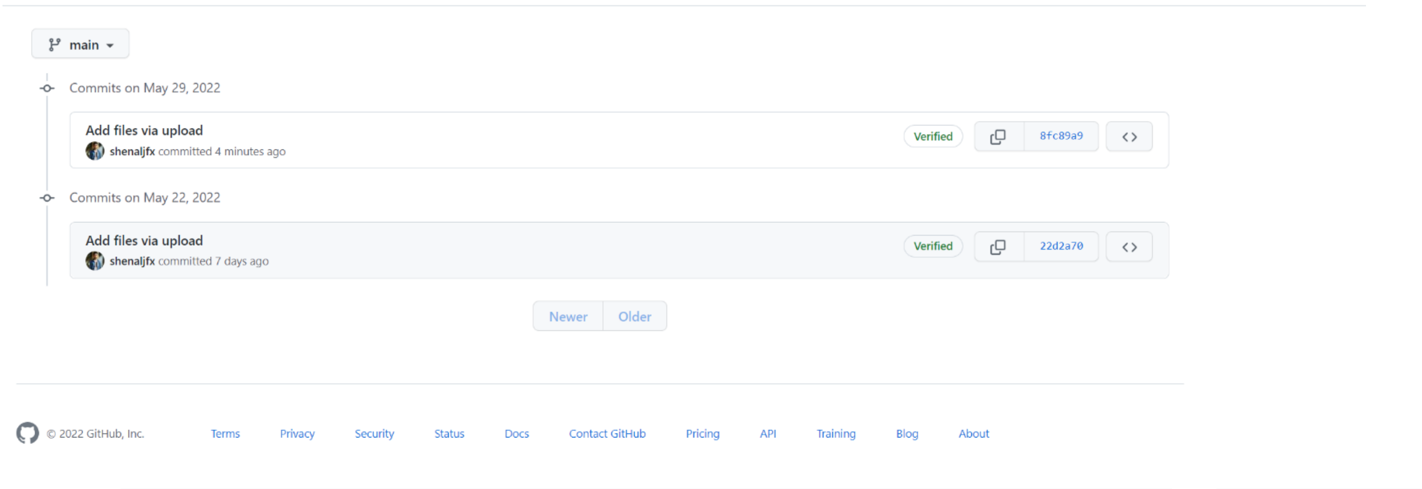
Visualization

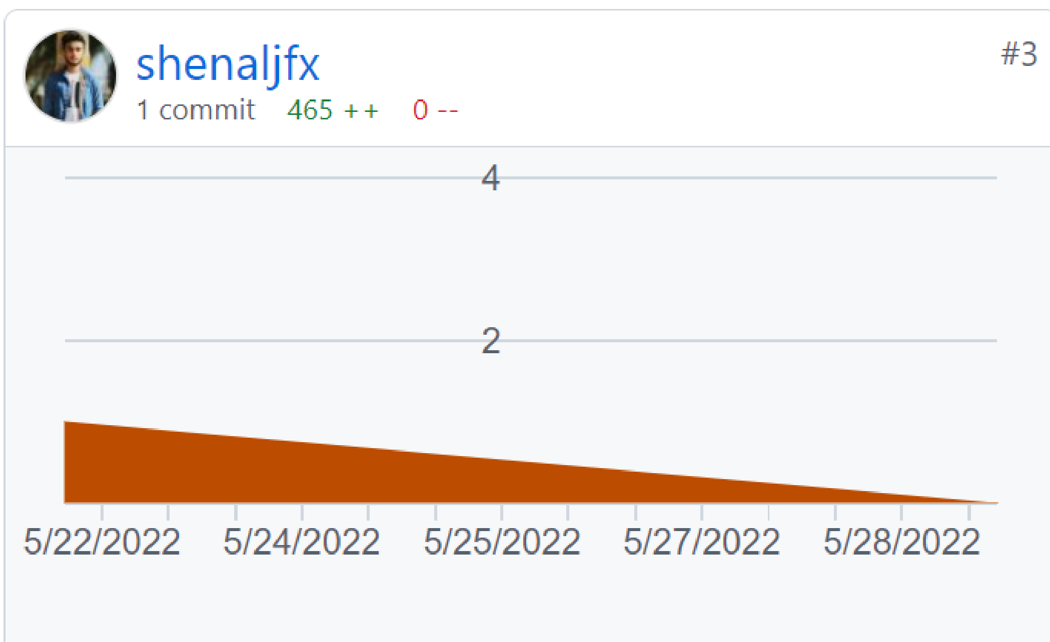
In [28]:
```
plt.figure(figsize=(14,5))
plt.plot(y__test, color = 'maroon', label = 'Real Tesla Stock Price')
plt.plot(y__pred, color = 'blue', label = 'Predicted Tesla Stock Price')
plt.title('Tesla Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Tesla Stock Price')
plt.legend()
plt.show()
```



Evidence for Individual contribution – IT19033938

GitHub Commit:





Source code

```
In [72]: import csv
         import datetime
         import numpy as np
         import pandas as pd
         from sklearn.linear_model import LinearRegression
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix, accuracy_score
         from sklearn.svm import SVR
         import matplotlib.pyplot as plt
```

```
In [63]: dataframe = pd.read_csv('TSLA_1M.csv')
         dataframe
```

Out[63]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-04-01	1081.150024	1094.750000	1066.640015	1084.589966	1084.589966	18087700
1	2022-04-04	1089.380005	1149.910034	1072.530029	1145.449951	1145.449951	27345300
2	2022-04-05	1136.300049	1152.869995	1087.300049	1091.260010	1091.260010	26691700
3	2022-04-06	1073.469971	1079.000000	1027.699951	1045.760010	1045.760010	29782800
4	2022-04-07	1052.390015	1076.589966	1021.539978	1057.260010	1057.260010	26482400
5	2022-04-08	1043.209961	1048.439941	1022.440002	1025.489990	1025.489990	18337900
6	2022-04-11	980.400024	1008.469971	974.640015	975.929993	975.929993	19785700
7	2022-04-12	997.640015	1021.190002	976.599976	986.950012	986.950012	21992000
8	2022-04-13	981.080017	1026.239990	973.099976	1022.369995	1022.369995	18373700
9	2022-04-14	999.289978	1012.710022	982.190002	985.000000	985.000000	19474100
10	2022-04-18	989.030029	1014.919983	973.409973	1004.289978	1004.289978	17238400
11	2022-04-19	1005.059998	1034.939941	995.330017	1028.150024	1028.150024	16615900
12	2022-04-20	1030.000000	1034.000000	975.250000	977.200012	977.200012	23570400
13	2022-04-21	1074.729980	1092.219971	996.419983	1008.780029	1008.780029	35138800
14	2022-04-22	1014.909973	1034.849976	994.000000	1005.049988	1005.049988	23232200
15	2022-04-25	978.969971	1008.619995	975.299988	998.020020	998.020020	22780400
16	2022-04-26	995.429993	1000.000000	875.000000	876.419983	876.419983	45377900
17	2022-04-27	898.580017	918.000000	877.359985	881.510010	881.510010	25652100
18	2022-04-28	899.979980	900.000000	821.700012	877.510010	877.510010	41649500
19	2022-04-29	902.250000	934.400024	870.000000	870.760010	870.760010	29313400

```
In [64]: def get_data(dataframe):
             data = dataframe.copy()
             data['Date'] = data['Date'].str.split('-').str[2]
             data['Date'] = pd.to_numeric(data['Date'])
             return [ data['Date'].tolist(), data['Open'].tolist() ]
```

```
In [65]: Open_date, open_price = get_data(dataframe)
```

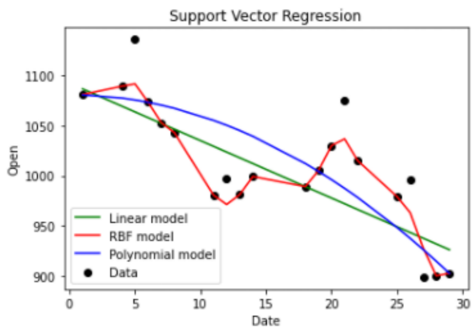
```
In [66]: def predict_prices(Open_date, open_price, x):
             Open_date = np.reshape(Open_date,(len(Open_date), 1))
             x = np.reshape(x,(len(x), 1))

             svr_lin  = SVR(kernel='linear', C=1e3)
             svr_poly = SVR(kernel='poly', C=1e3, degree=2)
             svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)

             svr_lin .fit(Open_date, open_price)
             svr_poly.fit(Open_date, open_price)
             svr_rbf.fit(Open_date, open_price)

             plt.scatter(Open_date, open_price, c='k', label='Data')
             plt.plot(Open_date, svr_lin.predict(Open_date), c='g', label='Linear model')
             plt.plot(Open_date, svr_rbf.predict(Open_date), c='r', label='RBF model')
             plt.plot(Open_date, svr_poly.predict(Open_date), c='b', label='Polynomial model')
```

```
    plt.show()

    return svr_rbf.predict(x)[0], svr_lin.predict(x)[0], svr_poly.predict(x)[0]
```

In [67]:
```
print(Open_date, open_price)
```

```
[1, 4, 5, 6, 7, 8, 11, 12, 13, 14, 18, 19, 20, 21, 22, 25, 26, 27, 28, 29] [1081.150024, 1089.380005, 1136.300049, 1073.469971,
1052.390015, 1043.209961, 980.400024, 997.640015, 981.080017, 999.289978, 989.030029, 1005.059998, 1030.0, 1074.72998, 1014.909
973, 978.969971, 995.429993, 898.580017, 899.97998, 902.25]
```

In [68]:
```
Predict_open_price = predict_prices(Open_date, open_price, [32])
```



In [85]:
```
Predict_open_price
```

Out[85]:
```
(987.6294760476794, 908.744592273018, 863.4717430329706)
```

In [70]:
```
#create two data set for independant and dependant variabels
x = dataframe[['Close','High','Low','Volume']]
y = dataframe['Open']
```

In [71]:
```
#split data set into  training annd testing  randomly
X_train, X_test, Y_train, Y_test =  train_test_split(x ,y ,random_state=0)
```

In [73]:
```
reg = LinearRegression()
```

In [74]:
```
#fit the model with data
reg.fit(X_train, Y_train)
```

Out[74]:
```
LinearRegression()
```

In [75]:
```
#predict the price using the trained model
predict_price = reg.predict(X_test)
```

In [76]:
```

from sklearn.metrics import r2_score
```

In [77]:
```
r2_score(Y_test, predict_price)
```

Out[77]:
```
0.9227211892498876
```

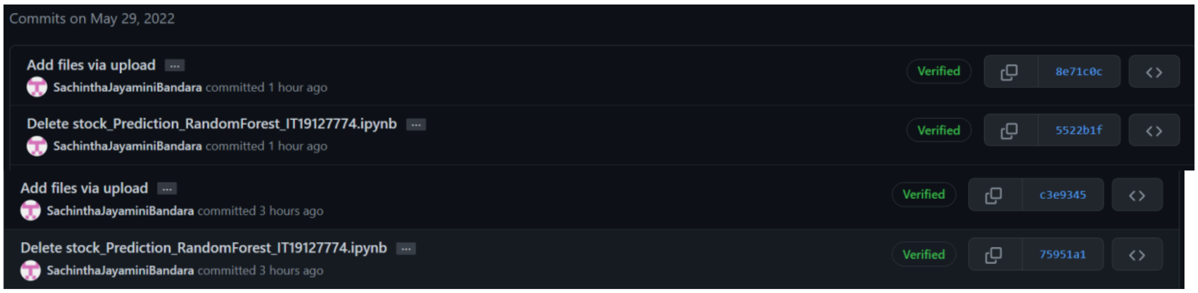
In [74]:
```
#fit the model with data
reg.fit(X_train, Y_train)
```
Out[74]:
```
LinearRegression()
```
In [75]:
```
#predict the price using the trained model
predict_price = reg.predict(X_test)
```
In [76]:
```

from sklearn.metrics import r2_score
```
In [77]:
```
r2_score(Y_test, predict_price)
```
Out[77]:
```
0.9227211892498876
```
In [78]:
```
predict_price, Y_test
```
Out[78]:
```
(array([ 866.8045828 , 1110.7499768 ,  918.08883244,  986.25304707,
         983.08178437]),
 18     899.979980
 1     1089.380005
 19     902.250000
 8      981.080017
 10     989.030029
 Name: Open, dtype: float64)
```
In [79]:
```
dataFrame = pd.DataFrame(Y_test,predict_price)
```
In [80]:
```
dframe = pd.DataFrame({'Actual Price':Y_test, 'Predicted Price' : predict_price})
```
In [81]:
```

sd=dframe.sort_values(by='Actual Price')
```
In [82]:
```
dframe.head(20).sort_values(by='Actual Price')
```
Out[82]:

	Actual Price	Predicted Price
18	899.979980	866.804583
19	902.250000	918.088832
8	981.080017	986.253047
10	989.030029	983.081784
1	1089.380005	1110.749977

In [ ]:

Evidence for Individual contribution – IT19127774

GitHub Commit:



Source code:

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

In [2]: import numpy as np
        import pandas as pd

        import matplotlib.pyplot as plt

In [3]: INFOSYS= pd.read_csv("C://Users//asus//Desktop//Stock market prediction using Random forest//INFY.NS.csv")
        INFOSYS = INFOSYS[['Date','Open', 'High', 'Low','Close','Adj Close','Volume']]
        INFOSYS.head()

In [4]: INFOSYS.describe()

In [5]: INFOSYS.info()

In [6]:
        import plotly as py
        import plotly.graph_objs as go
        from plotly.offline import plot

        from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
        init_notebook_mode(connected=True)

In [7]: layout = go.Layout(
            title='STOCK PRICE OF INFOSYS',
                xaxis=dict(
                    title='date',
                    titlefont=dict(
                        family='Courier New, monospace',
                        size=18,
                        color='blue'
                    )
                ),
                yaxis=dict(
                    title='Price',
                    titlefont=dict(
                        family='Courier New, monospace',
                        size=18,
                        color='red'
                    )
            )
        )
        INFOSYS_DATA = [{'x':INFOSYS['Date'], 'y':INFOSYS['Close']}]
        plot = go.Figure(data=INFOSYS_DATA, layout=layout)

In [8]: iplot(plot)

In [9]: INFOSYS['Open-Close']= INFOSYS.Close - INFOSYS.Open
        INFOSYS['High-Low']  = INFOSYS.High - INFOSYS.Low
        INFOSYS = INFOSYS.dropna()
        X= INFOSYS[['Open-Close', 'High-Low']]
        X.head()
```

In [10]:
```
Y= np.where(INFOSYS['Close'].shift(-1)>INFOSYS['Close'],1,-1)
```

In [11]:
```
split_percentage = 0.8
split = int(split_percentage*len(INFOSYS))

X_train = X[:split]
Y_train = Y[:split]

X_test = X[split:]
Y_test = Y[split:]
```

In [12]:
```
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

In [13]:
```
scores = []

for num_trees in range(1,41):
    clf = RandomForestClassifier(n_estimators = num_trees)
    scores.append(cross_val_score(clf, X, Y, cv=10))
```

In [14]:
```
print(scores[0])
```

In [15]:
```
print(scores[1])
```

In [16]:
```
rfc = RandomForestClassifier(n_estimators=16)
```

In [17]:
```
rfc.fit(X_train, Y_train)
```

In [18]:
```
rfc_pred = rfc.predict(X_test)
```

In [19]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [20]:
```
print(classification_report(Y_test, rfc_pred))
```

In [21]:
```
print(confusion_matrix(Y_test, rfc_pred))
```

In [22]:
```
INFOSYS['Predicted_Signal'] = rfc.predict(X)

INFOSYS['SPY_returns'] = np.log(INFOSYS['Close']/INFOSYS['Close'].shift(1))
Cumulative_SPY_returns = INFOSYS[split:]['SPY_returns'].cumsum()*100

INFOSYS['Startegy_returns'] = INFOSYS['SPY_returns']* INFOSYS['Predicted_Signal'].shift(1)
Cumulative_Strategy_returns = INFOSYS[split:]['Startegy_returns'].cumsum()*100

plt.figure(figsize=(10,5))
plt.plot(Cumulative_SPY_returns, color='r',label = 'SPY Returns')
plt.plot(Cumulative_Strategy_returns, color='g', label = 'Strategy Returns')
plt.legend()
plt.show()
```

In [23]:
```
Std = Cumulative_Strategy_returns.std()
Sharpe = (Cumulative_Strategy_returns-Cumulative_SPY_returns)/Std
Sharpe = Sharpe.mean()
print ('Sharpe ratio: %.2f'%Sharpe )
```

In [24]:
```
model = rfc.fit(X_train, Y_train)
model = rfc.fit (X_train,Y_train)
```

In [25]:
```
probability = model.predict_proba(X_test)
print(probability)
```

In [26]:
```
predicted = rfc.predict(X_test)
```

In [27]:
```
from sklearn import metrics
```

In [28]:
```
print(metrics.confusion_matrix(Y_test, predicted))
```

In [29]:
```
print(metrics.classification_report(Y_test, predicted))
```

In [30]:
```
print(model.score(X_train,Y_train))
```

n [69]:
```
dataFrame = pd.DataFrame(Y_test,predicted)
```

n [70]:
```
dataFrame
```

In [71]:
```
dframe = pd.DataFrame({'Actual Price':Y_test, 'Predicted Price' : predicted})
```

In [72]:
```
sd=dframe.sort_values(by='Actual Price')
```

In [73]:
```
sd['Actual Price']
```

In [74]:
```
print(sd)
```

In [75]:
```
dframe.head(20).sort_values(by='Actual Price')
```

In [79]:
```
plt.figure(figsize=(14,5))
plt.ylabel('Price', fontsize=20)
plt.title("Stock price prediction")
plt.plot(dframe)
plt.legend(['Real stock price', 'Predicted stock price'])
plt.show()
```