Report on:

Stock Market Prediction using Machine Learning Algorithms

-prepared by
Aanchal Soni (B190045EC)

Introduction

Stock Market Prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The entire idea of predicting stock prices is to gain significant profit. Estimating the stock market has a high demand for stock customers. Applying all the extracted rules at any time is a major challenge to estimate the future stock price with high accuracy. Accurate stock price prediction is an extremely challenging process because of multiple factors such as economic condition, a company's financial performances and so on. Plotting stock prices along a normal distribution can allow traders to see when the stock is overbought and oversold.

So in this project we will use a different Machine learning algorithm for early prediction of stock market price with high accuracy.

Libraries imported in our project:

Numpy
Numpy module offers an object called array where we perform mathematical operations.

Pandas
It is a Python Data Analysis library. It is used to provide the data frame and used to analyze the data.

Matplotlib

In Machine Learning, it helps us to understand the huge amount of data through visualization.

Fastai
Fastai library's goal is to make the training of deep neural networks as easy as possible and at the same time make it fast and accurate using modern best prices.

Sklearn
In Machine Learning, sklearn is probably the most useful library for ML in python. It contains a lot of efficient tools for machine learning and statistical modeling including classification, regression etc.

Keras
Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

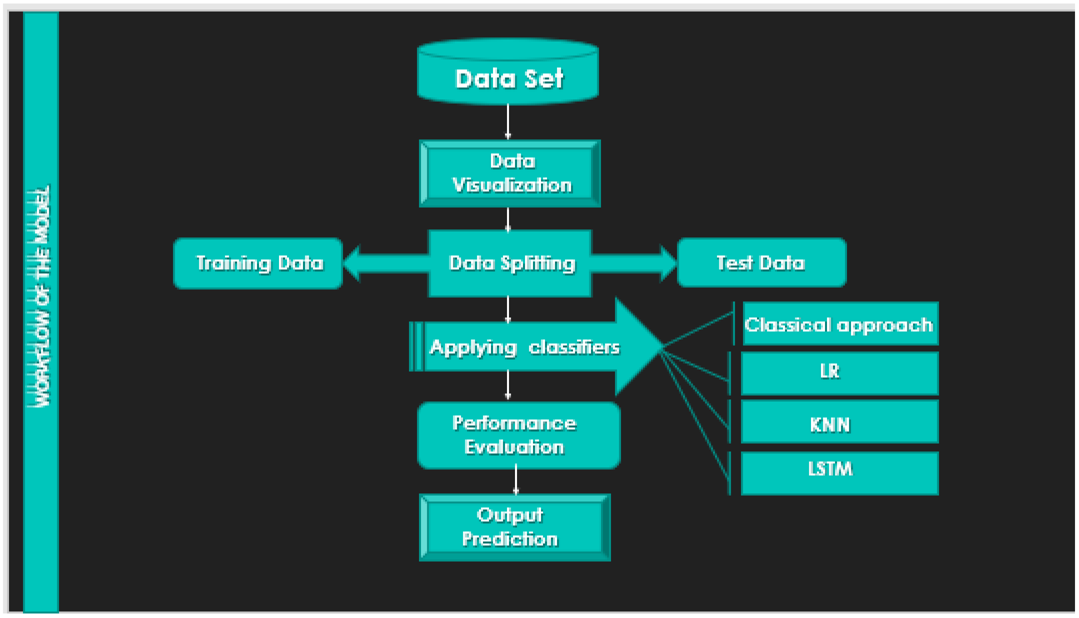
Grid Search CV
It is a technique to search through the best parameter values from the given set of grid of parameters.

Machine Learning Models Used:

1. Linear Regression
2. K - Nearest Neighbor
3. Long Term Short Memory

Proposed Methodology

The algorithm process proposed in the project is represented in the figure attached below.



- First we import the different libraries
- Then we visualize the data
- Then we split the data into training data and test data
- After that we apply different ML models i.e Classical Approach, Linear Regression, KNN and LSTM.
- Then we have evaluated the performance of the different models to check which model works best.
- The output is predicted.

A brief on LSTM

A recurrent neural network also known as RNN is used for persistent memory.

Long Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the problems faced by traditional RNN.

The shortcoming of RNN is, they can not remember Long term dependencies due to vanishing gradients. I.e, Networks are unable to backpropagate the gradient information to the input layers of the model. LSTMs are explicitly designed to avoid long-term dependency problems.

Program:

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

Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from fastai.tabular import *
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
%matplotlib inline
```

Setting Figure Size

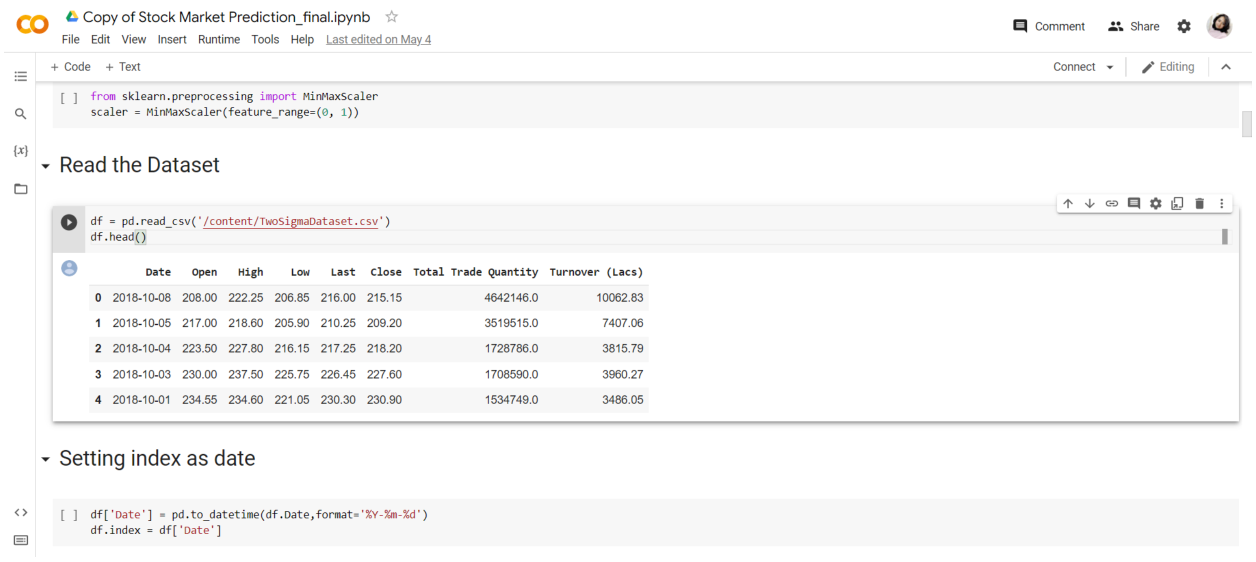
```
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20,10
```

For Normalizing Data

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
```

Reading the Dataset

```
df = pd.read_csv('/content/TwoSigmaDataset.csv')
df.head()
```


Copy of Stock Market Prediction_final.ipynb
File Edit View Insert Runtime Tools Help Last edited on May 4
Comment Share
+ Code + Text
Connect Editing

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
```

Read the Dataset

```
df = pd.read_csv('/content/TwoSigmaDataset.csv')
df.head()
```

```
         Date    Open    High     Low    Last   Close  Total Trade Quantity  Turnover (Lacs)
0  2018-10-08  208.00  222.25  206.85  216.00  215.15             4642146.0         10062.83
1  2018-10-05  217.00  218.60  205.90  210.25  209.20             3519515.0          7407.06
2  2018-10-04  223.50  227.80  216.15  217.25  218.20             1728786.0          3815.79
3  2018-10-03  230.00  237.50  225.75  226.45  227.60             1708590.0          3960.27
4  2018-10-01  234.55  234.60  221.05  230.30  230.90             1534749.0          3486.05
```

Setting index as date

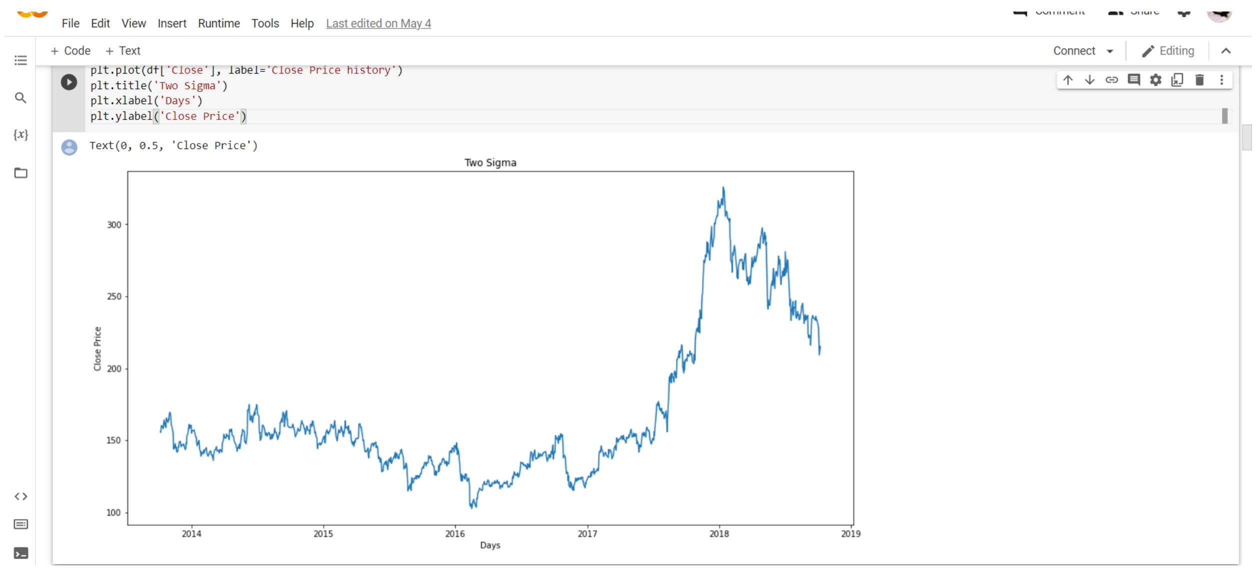
```
df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df['Date']
```

Setting index as date

```
df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df['Date']
```

Plotting the original dataset

```
plt.figure(figsize=(16,8))
plt.plot(df['Close'], label='Close Price history')
plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



```
plt.plot(df['Close'], label='Close Price history')
plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```

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

Ranging The Data

```
for i in range(0,len(data)):
     new_data['Date'][i] = data['Date'][i]
     new_data['Close'][i] = data['Close'][i]
```

Splitting into Train and Validation

```
train = new_data[:900]
valid = new_data[900:]
```

Shapes of Training Set

```
print('\n Shape of training set:')
print(train.shape)
Shape of training set:
(900, 2)
```

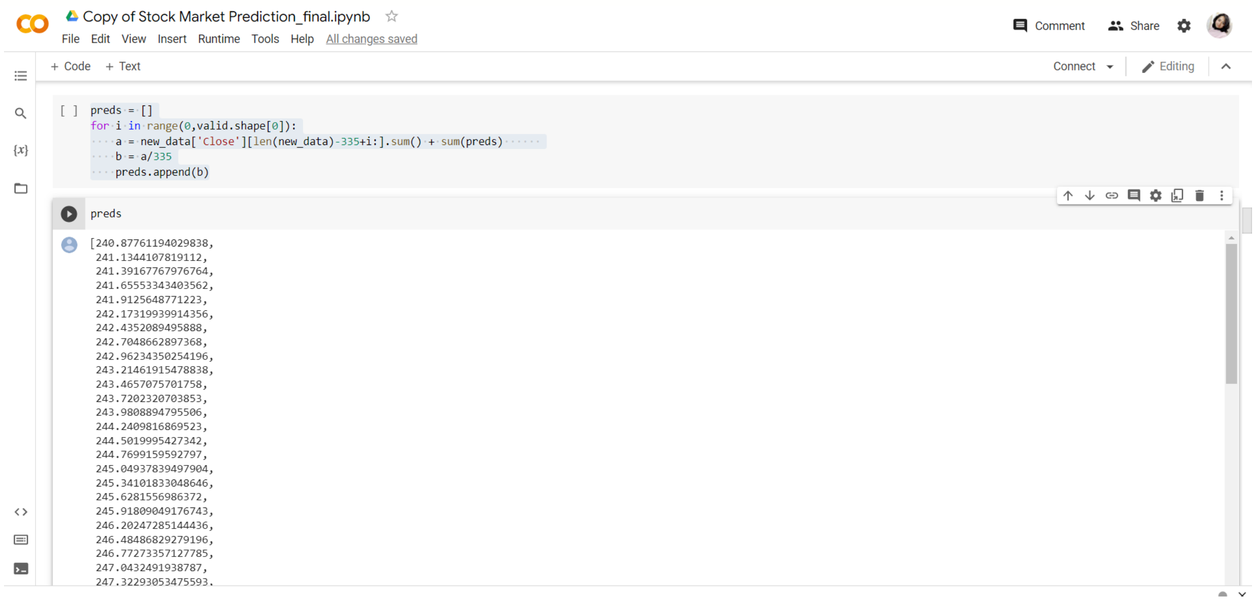
Shapes of Validation Set

```
print('\n Shape of validation set:')
print(valid.shape)
```

```
Shape of validation set:
(335, 2)
```

Making Predictions

```
preds = []
for i in range(0,valid.shape[0]):
    a = new_data['Close'][len(new_data)-335+i:].sum() + sum(preds)
    b = a/335
    preds.append(b)
```


Copy of Stock Market Prediction_final.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Editing

```
preds = []
for i in range(0,valid.shape[0]):
    a = new_data['Close'][len(new_data)-335+i:].sum() + sum(preds)
    b = a/335
    preds.append(b)
```

```
preds
```

```
[240.87761194029838,
 241.1344107819112,
 241.39167767976764,
 241.65553343403562,
 241.9125648771223,
 242.17319939914356,
 242.4352089495888,
 242.7048662897368,
 242.96234350254196,
 243.21461915478838,
 243.4657075701758,
 243.7202320703853,
 243.9808894795506,
 244.2409816869523,
 244.5019995427342,
 244.7699159592797,
 245.04937839497904,
 245.34101833048646,
 245.6281556986372,
 245.91809049176743,
 246.20247285144436,
 246.48486829279196,
 246.77273357127785,
 247.0432491938787,
 247.32293053475593,
```

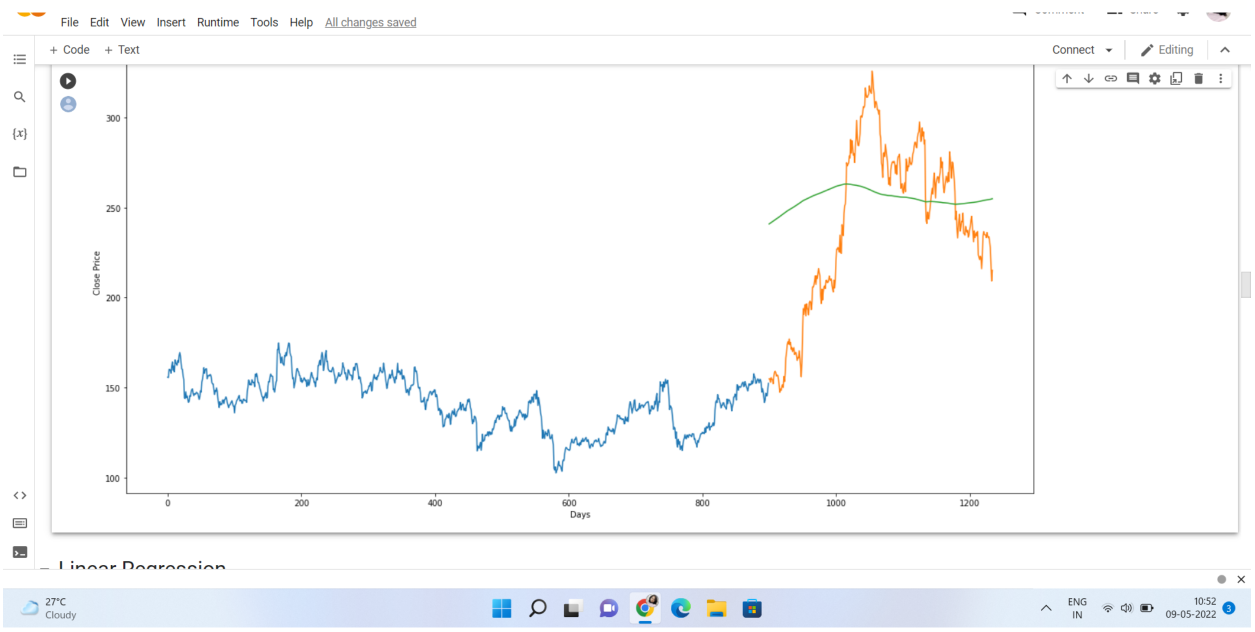
Checking The Results (RMSE value)

```
rms=np.sqrt(np.mean(np.power((np.array(valid['Close'])-preds),2)))
print('\n RMSE value on validation set:')
print(rms)
```

```
RMSE value on validation set:
45.239080104712116
```

```
#plot
valid['Predictions'] = 0
```

```
valid['Predictions'] = preds
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



Linear Regression

```
#setting index as date values
df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df['Date']

#creating a separate dataset
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

add_datepart(new_data, 'Date')
new_data.drop('Elapsed', axis=1, inplace=True)  #elapsed will be the time
stamp
```

```
from pandas._libs.hashtable import mode
#split into train and validation
train_lr = new_data[:900]
valid_lr = new_data[900:]

x_train_lr = train_lr.drop('Close', axis=1)
y_train_lr = train_lr['Close']
x_valid_lr = valid_lr.drop('Close', axis=1)
y_valid_lr = valid_lr['Close']

#implement linear regression

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train_lr,y_train_lr)
```

K-Nearest Neighbours

```
#split into train and validation

train_knn = new_data[:900]
valid_knn = new_data[900:]

x_train_knn = train_knn.drop('Close', axis=1)
y_train_knn = train_knn['Close']
x_valid_knn = valid_knn.drop('Close', axis=1)
y_valid_knn = valid_knn['Close']


#using gridsearch to find the best parameter
neighbor = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, neighbor, cv=6)

#fit the model and make predictions
```

```
model.fit(x_train_knn,y_train_knn)
preds_knn = model.predict(x_valid_knn)
```

Results

```
#make predictions and find the rmse
preds= model.predict(x_valid_lr)
rms=np.sqrt(np.mean(np.power((np.array(y_valid_lr)-np.array(preds)),2)))
rms
```

```
114.77000319913022
```

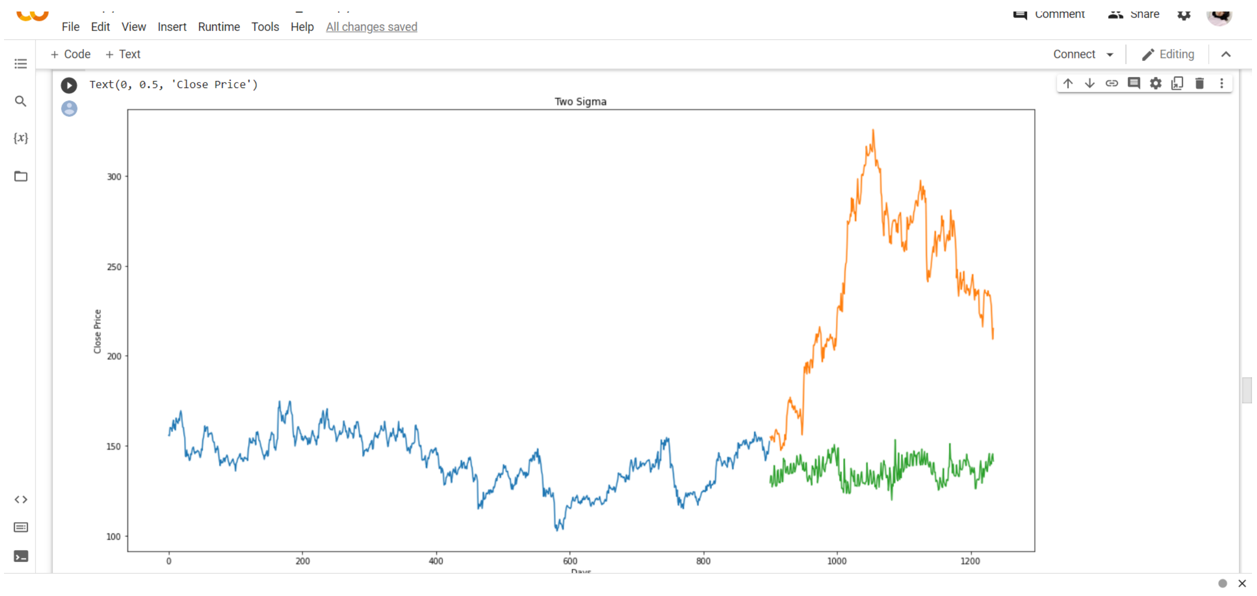
```
valid['Predictions'] = 0
valid['Predictions'] = preds

valid.index = new_data[900:].index
train.index = new_data[:900].index

plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])

plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```

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



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

K-Nearest Neighbours

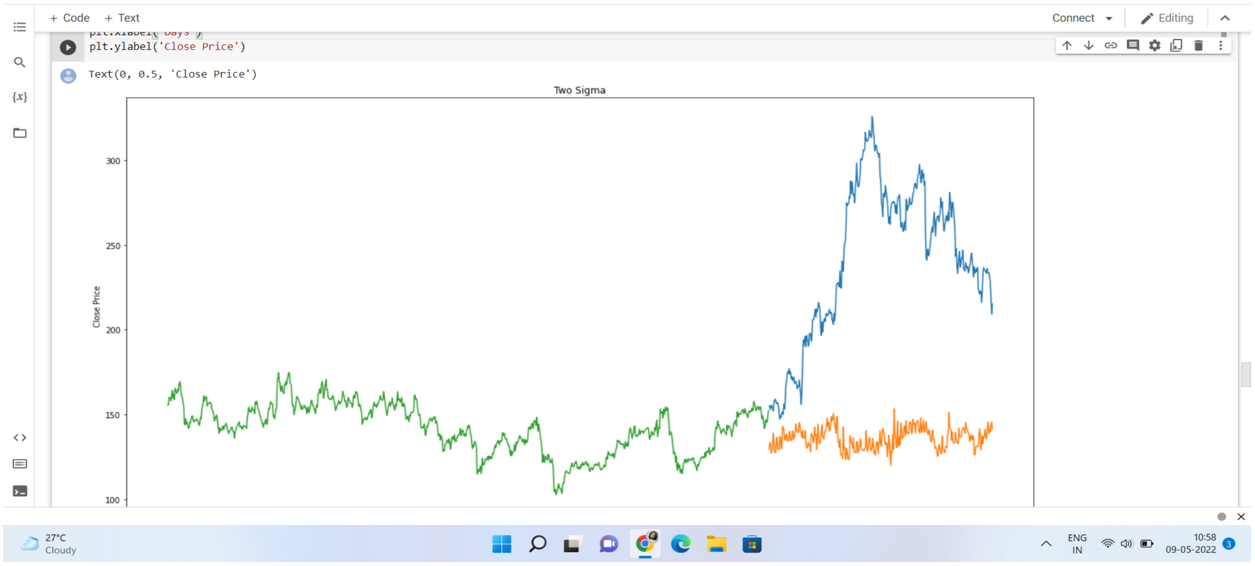
```
#rmse
rms=np.sqrt(np.mean(np.power((np.array(y_valid_knn)-np.array(preds_knn)),2
)))
rms

114.77000319913022



#plot
valid['Predictions'] = 0
valid['Predictions'] = preds_knn
plt.plot(valid[['Close', 'Predictions']])
plt.plot(train['Close'])

plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



```
plt.ylabel('Close Price')
```

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

Long Short Term Memory (LSTM)

```
#importing required libraries
from keras.models import Sequential    // Keras Library is used for deep learning Models
from keras.layers import Dense, Dropout, LSTM

#creating dataframe
#data = df.sort_index(ascending=True, axis=0)
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

#setting index
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)

#creating train and test sets
dataset = new_data.values

train = dataset[0:900,:]
valid = dataset[900:,:]
```

```
#converting dataset into x_train and y_train
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

x_train, y_train = [], []
for i in range(60,len(train)):
    x_train.append(scaled_data[i-60:i,0])
    y_train.append(scaled_data[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=5, batch_size=1, verbose=2)

#predicting 355 values, using past 60 from the train data
inputs = new_data[len(new_data) - len(valid) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)

X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

```
Epoch 1/5
840/840 - 24s - loss: 9.3870e-04 - 24s/epoch - 29ms/step
```

```
Epoch 2/5
840/840 - 21s - loss: 4.5876e-04 - 21s/epoch - 25ms/step
Epoch 3/5
840/840 - 21s - loss: 3.1944e-04 - 21s/epoch - 25ms/step
Epoch 4/5
840/840 - 21s - loss: 2.4578e-04 - 21s/epoch - 24ms/step
Epoch 5/5
840/840 - 21s - loss: 2.0917e-04 - 21s/epoch - 25ms/step
```

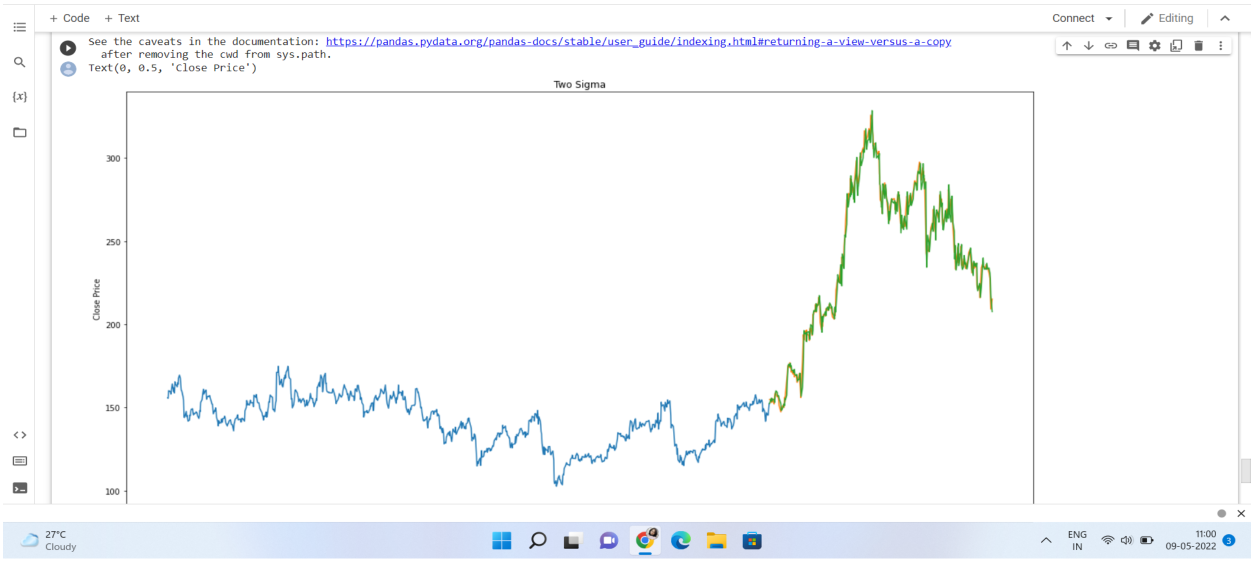
Result

```
rms=np.sqrt(np.mean(np.power((valid-closing_price),2)))
rms
```

```
5.77957835528417
```

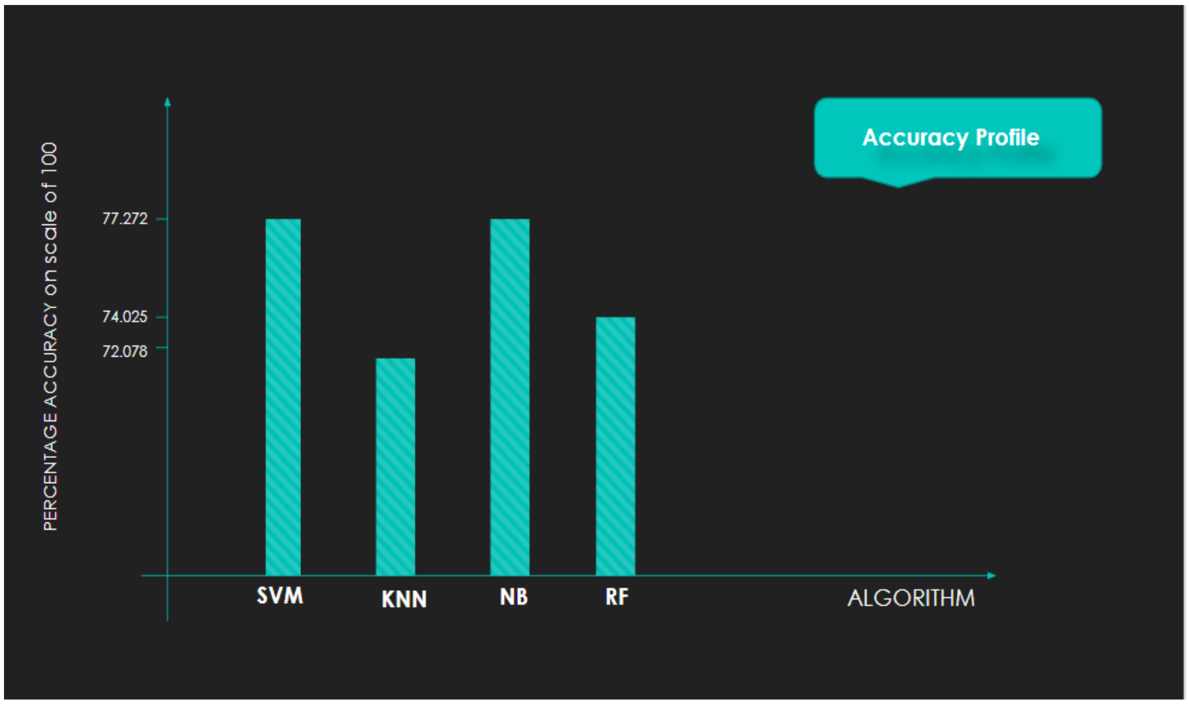
```
#for plotting
train = new_data[:900]
valid = new_data[900:]
valid['Predictions'] = closing_price
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])

plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
Text(0, 0.5, 'Close Price')
```

Result:



Conclusion

Plotting stock prices along a normal distribution can allow traders to see when the stock is overbought or oversold.

Using LSTM, time series forecasting models can predict future values based on previous, sequential data. This provides greater accuracy for demand forecasters which results in better decision making for the business.
So, LSTM is the best model for predicting stock prices.

Report on:

Stock Market Prediction using Machine Learning Algorithms

-prepared by
Group-2

- **Aanchal Soni (B190045EC)**
- **Milisha Mahapatra (B190058EC)**
- **Srishti Prasad (B190068EC)**

Date: May 9, 2022

Introduction

Stock Market Prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The entire idea of predicting stock prices is to gain significant profit. Estimating the stock market has a high demand for stock customers. Applying all the extracted rules at any time is a major challenge to estimate the future stock price with high accuracy. Accurate stock price prediction is an extremely challenging process because of multiple factors such as economic condition, a company's financial performances and so on. Plotting stock prices along a normal distribution can allow traders to see when the stock is overbought and oversold.

So in this project we will use a different Machine learning algorithm for early prediction of stock market price with high accuracy.

Libraries imported in our project:

Numpy

Numpy module offers an object called array where we perform mathematical operations.

Pandas
It is a Python Data Analysis library. It is used to provide the data frame and used to analyse the data.

Matplotlib
In Machine Learning, it helps us to understand the huge amount of data through visualisation.

Fastai
Fastai library's goal is to make the training of deep neural networks as easy as possible and at the same time make it fast and accurate using modern best prices.

Sklearn
In Machine Learning, sklearn is probably the most useful library for ML in python. It contains a lot of efficient tools for machine learning and statistical modeling including classification, regression etc.

Keras
Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

Grid Search CV
It is a technique to search through the best parameter values from the given set of grid of parameters.

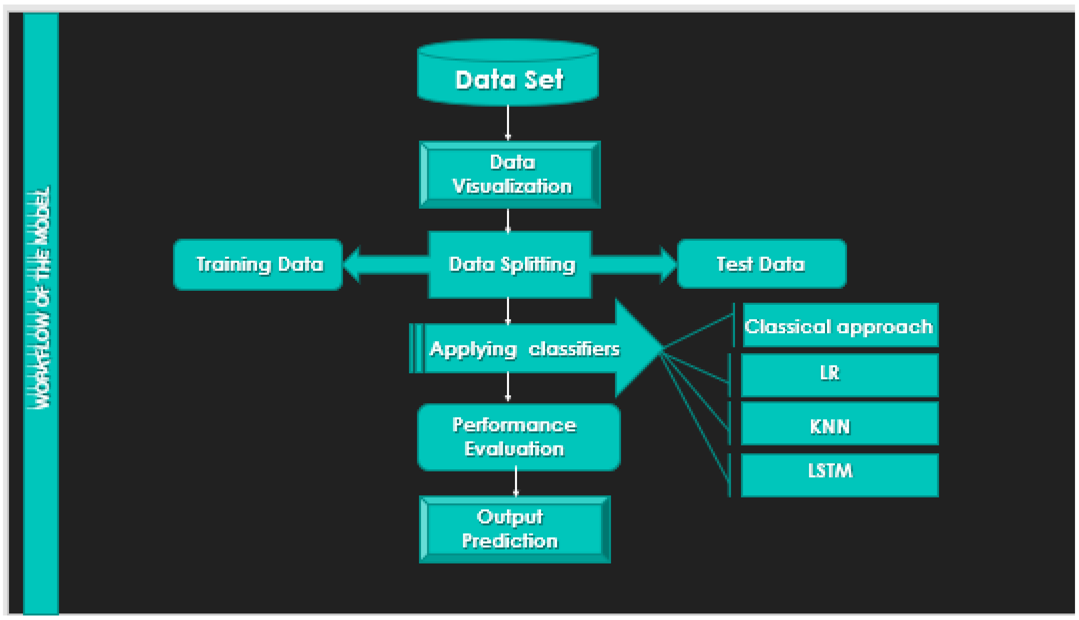
Machine Learning Models Used:

1. Linear Regression
2. K - Nearest Neighbour

3. Long Term Short Memory

Proposed Methodology

The algorithm process proposed in the project is represented in the figure attached below.



- First we import the different libraries
- Then we visualize the data
- Then we split the data into training data and test data
- After that we apply different ML models i.e Classical Approach, Linear Regression, KNN and LSTM.
- Then we have evaluated the performance of the different models to check which model works best.
- The output is predicted.

<u>A brief on LSTM</u>

A recurrent neural network also known as RNN is used for persistent memory.

Long Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the problems faced by traditional RNN.

The shortcoming of RNN is, they can not remember Long term dependencies due to vanishing gradients. I.e, Networks are unable to backpropagate the gradient information to the input layers of the model. LSTMs are explicitly designed to avoid long-term dependency problems.

Program:

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

Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from fastai.tabular import *
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
%matplotlib inline
```

Setting Figure Size

```
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20,10
```

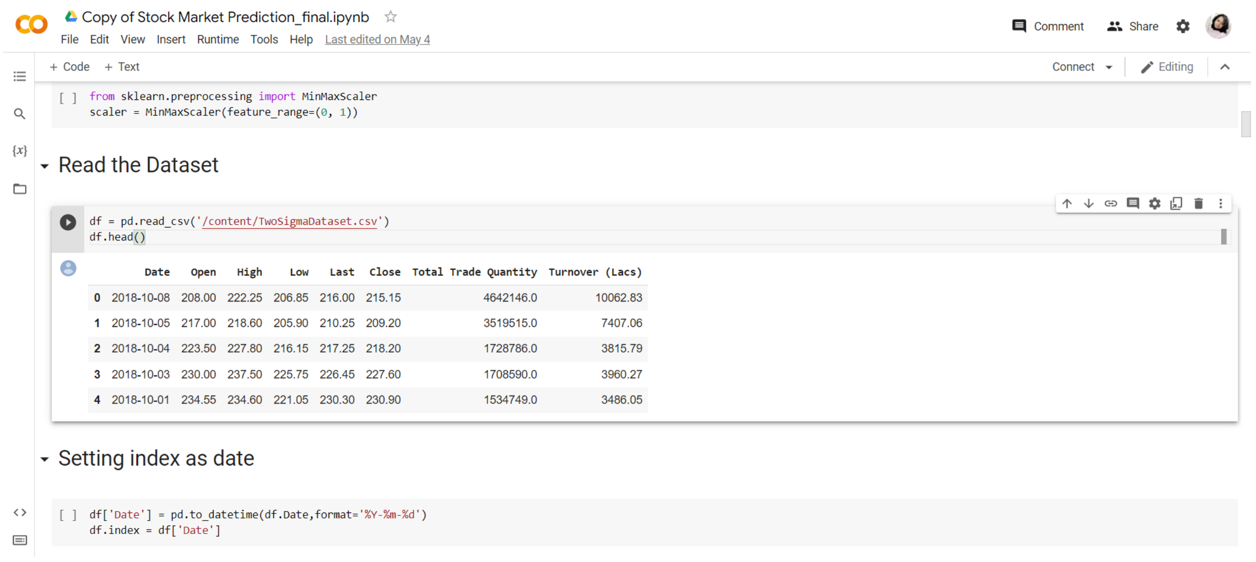
For Normalizing Data

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
```

Reading the Dataset

```
df = pd.read_csv('/content/TwoSigmaDataset.csv')
df.head()
```



Copy of Stock Market Prediction_final.ipynb
File Edit View Insert Runtime Tools Help Last edited on May 4
Comment
Share
+ Code + Text
Connect
Editing

```
[ ] from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler(feature_range=(0, 1))
```

Read the Dataset

```
df = pd.read_csv('/content/TwoSigmaDataset.csv')
df.head()
```

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146.0	10062.83
1	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515.0	7407.06
2	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786.0	3815.79
3	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590.0	3960.27
4	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749.0	3486.05

Setting index as date

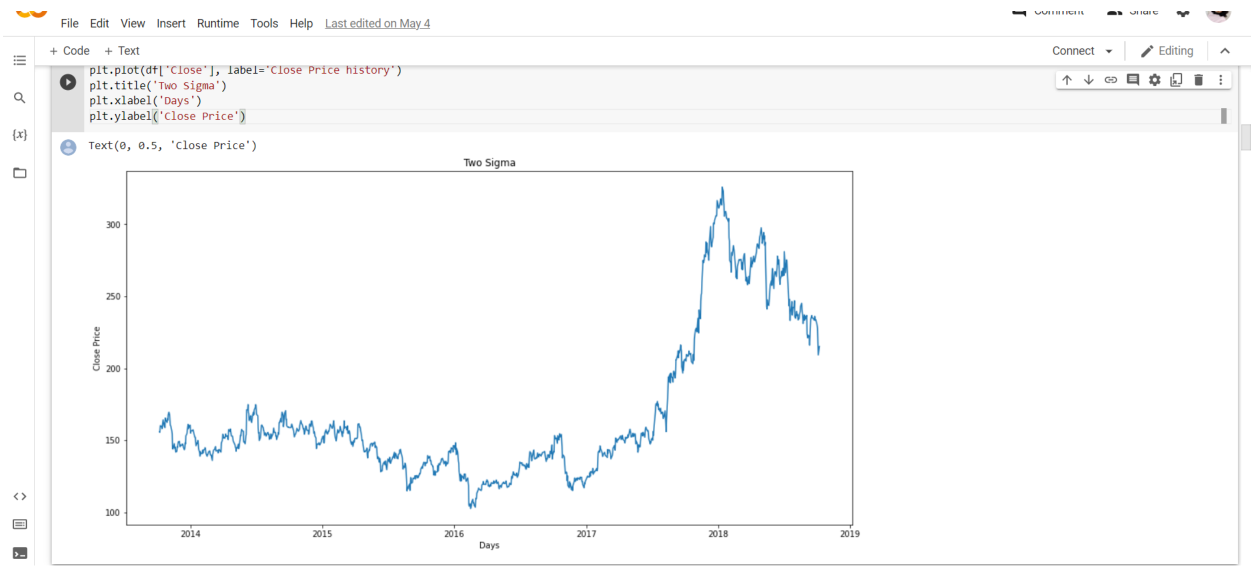
```
[ ] df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
    df.index = df['Date']
```

Setting index as date

```
df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df['Date']
```

Plotting the original dataset

```
plt.figure(figsize=(16,8))
plt.plot(df['Close'], label='Close Price history')
plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



```
plt.plot(df['Close'], label='Close Price history')
plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```

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

Ranging The Data

```
for i in range(0,len(data)):
     new_data['Date'][i] = data['Date'][i]
     new_data['Close'][i] = data['Close'][i]
```

Splitting into Train and Validation

```
train = new_data[:900]
valid = new_data[900:]
```

Shapes of Training Set

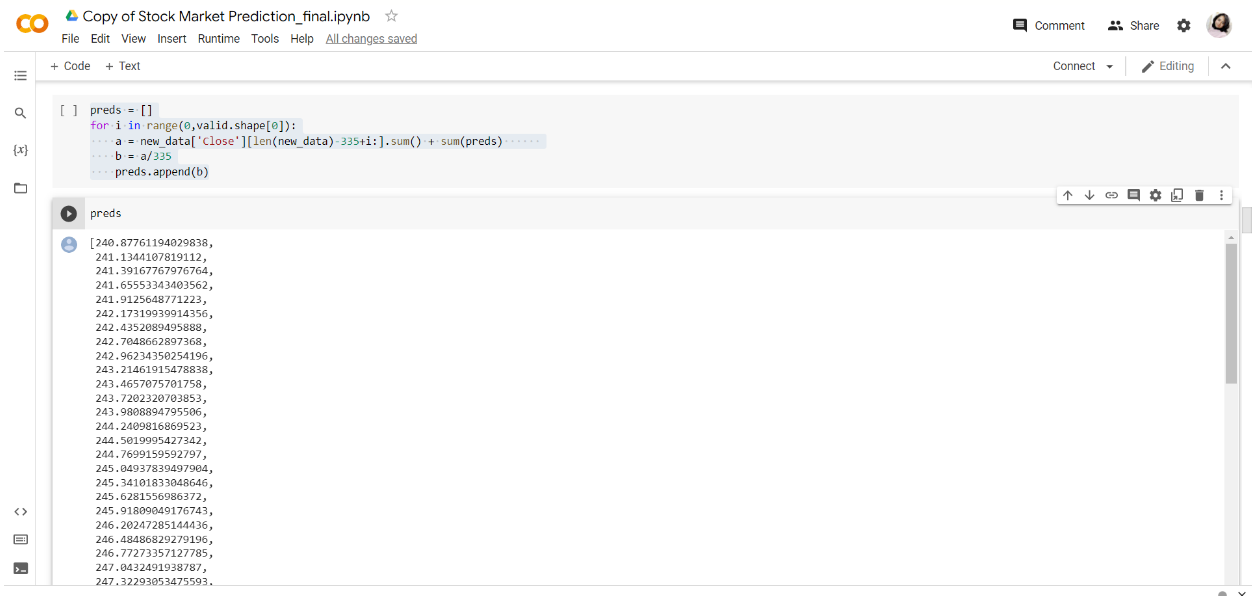
```
print('\n Shape of training set:')
print(train.shape)
Shape of training set:
(900, 2)
```

Shapes of Validation Set

```
print('\n Shape of validation set:')
print(valid.shape)
Shape of validation set:
(335, 2)
```

Making Predictions

```
preds = []
for i in range(0,valid.shape[0]):
    a = new_data['Close'][len(new_data)-335+i:].sum() + sum(preds)
    b = a/335
    preds.append(b)
```


Copy of Stock Market Prediction_final.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Editing

```
preds = []
for i in range(0,valid.shape[0]):
    a = new_data['Close'][len(new_data)-335+i:].sum() + sum(preds)
    b = a/335
    preds.append(b)
```

```
preds
```

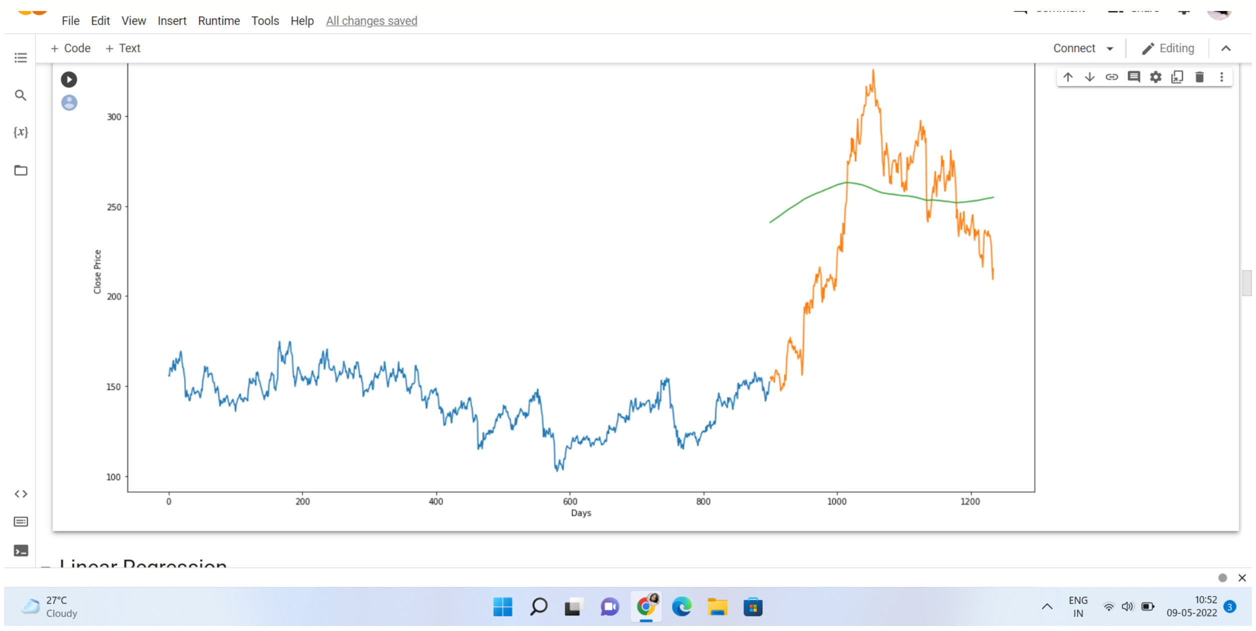
```
[240.87761194029838,
 241.1344107819112,
 241.39167767976764,
 241.65553343403562,
 241.9125648771223,
 242.17319939914356,
 242.4352089495888,
 242.7048662897368,
 242.96234350254196,
 243.21461915478838,
 243.4657075701758,
 243.7202320703853,
 243.9808894795506,
 244.2409816869523,
 244.5019995427342,
 244.7699159592797,
 245.04937839497904,
 245.34101833048646,
 245.6281556986372,
 245.91809049176743,
 246.20247285144436,
 246.48486829279196,
 246.77273357127785,
 247.0432491938787,
 247.32293053475593,
```

Checking The Results (RMSE value)

```
rms=np.sqrt(np.mean(np.power((np.array(valid['Close'])-preds),2)))
print('\n RMSE value on validation set:')
print(rms)
```

```
RMSE value on validation set:
45.239080104712116
```

```
#plot
valid['Predictions'] = 0
valid['Predictions'] = preds
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



Linear Regression

```
#setting index as date values
df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df['Date']

#creating a separate dataset
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]
```

```
add_datepart(new_data, 'Date')
new_data.drop('Elapsed', axis=1, inplace=True)  #elapsed will be the time
stamp



from pandas._libs.hashtable import mode
#split into train and validation
train_lr = new_data[:900]
valid_lr = new_data[900:]

x_train_lr = train_lr.drop('Close', axis=1)
y_train_lr = train_lr['Close']
x_valid_lr = valid_lr.drop('Close', axis=1)
y_valid_lr = valid_lr['Close']

#implement linear regression

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train_lr,y_train_lr)
```

K-Nearest Neighbours

```
#split into train and validation

train_knn = new_data[:900]
valid_knn = new_data[900:]

x_train_knn = train_knn.drop('Close', axis=1)
y_train_knn = train_knn['Close']
x_valid_knn = valid_knn.drop('Close', axis=1)
y_valid_knn = valid_knn['Close']


#using gridsearch to find the best parameter
neighbor = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
```

```
model = GridSearchCV(knn, neighbor, cv=6)

#fit the model and make predictions
model.fit(x_train_knn,y_train_knn)
preds_knn = model.predict(x_valid_knn)
```

Results

```
#make predictions and find the rmse
preds= model.predict(x_valid_lr)
rms=np.sqrt(np.mean(np.power((np.array(y_valid_lr)-np.array(preds)),2)))
rms
```

```
114.77000319913022
```

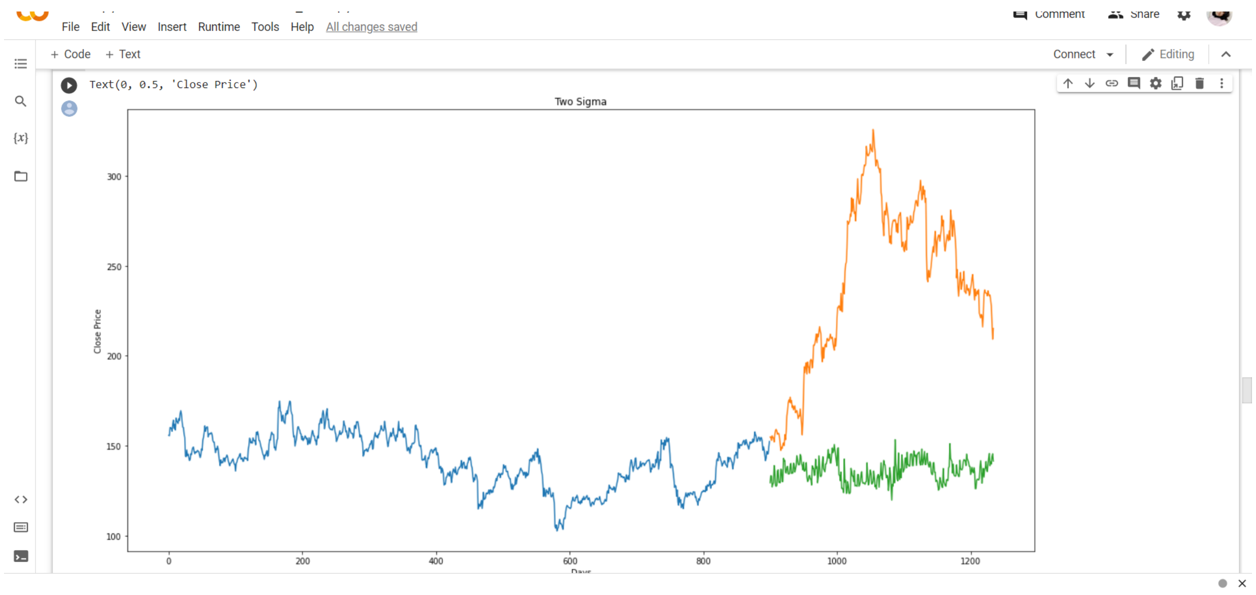
```
valid['Predictions'] = 0
valid['Predictions'] = preds

valid.index = new_data[900:].index
train.index = new_data[:900].index

plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])

plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```

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



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

K-Nearest Neighbours

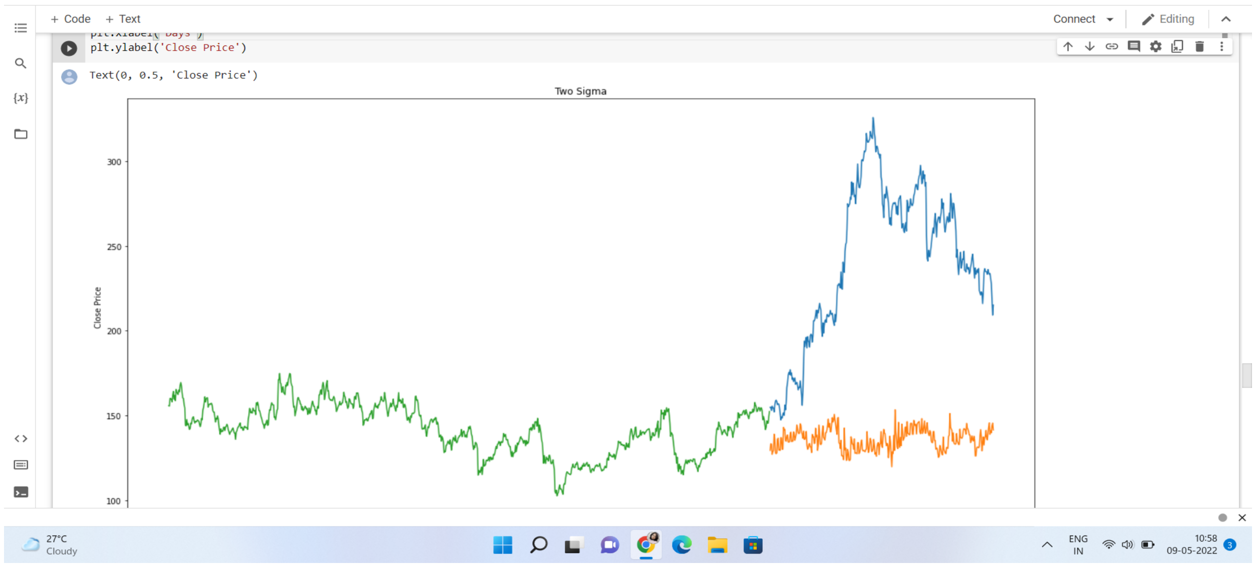
```
#rmse
rms=np.sqrt(np.mean(np.power((np.array(y_valid_knn)-np.array(preds_knn)),2
)))
rms

114.77000319913022



#plot
valid['Predictions'] = 0
valid['Predictions'] = preds_knn
plt.plot(valid[['Close', 'Predictions']])
plt.plot(train['Close'])

plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



```
plt.ylabel('Close Price')
```

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

Long Short Term Memory (LSTM)

```
#importing required libraries
from keras.models import Sequential    // Keras Library is used for deep learning Models
from keras.layers import Dense, Dropout, LSTM

#creating dataframe
#data = df.sort_index(ascending=True, axis=0)
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

#setting index
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)

#creating train and test sets
dataset = new_data.values

train = dataset[0:900,:]
valid = dataset[900:,:]
```

```
#converting dataset into x_train and y_train
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

x_train, y_train = [], []
for i in range(60,len(train)):
    x_train.append(scaled_data[i-60:i,0])
    y_train.append(scaled_data[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=5, batch_size=1, verbose=2)

#predicting 355 values, using past 60 from the train data
inputs = new_data[len(new_data) - len(valid) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)

X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

```
Epoch 1/5
840/840 - 24s - loss: 9.3870e-04 - 24s/epoch - 29ms/step
```

```
Epoch 2/5
840/840 - 21s - loss: 4.5876e-04 - 21s/epoch - 25ms/step
Epoch 3/5
840/840 - 21s - loss: 3.1944e-04 - 21s/epoch - 25ms/step
Epoch 4/5
840/840 - 21s - loss: 2.4578e-04 - 21s/epoch - 24ms/step
Epoch 5/5
840/840 - 21s - loss: 2.0917e-04 - 21s/epoch - 25ms/step
```

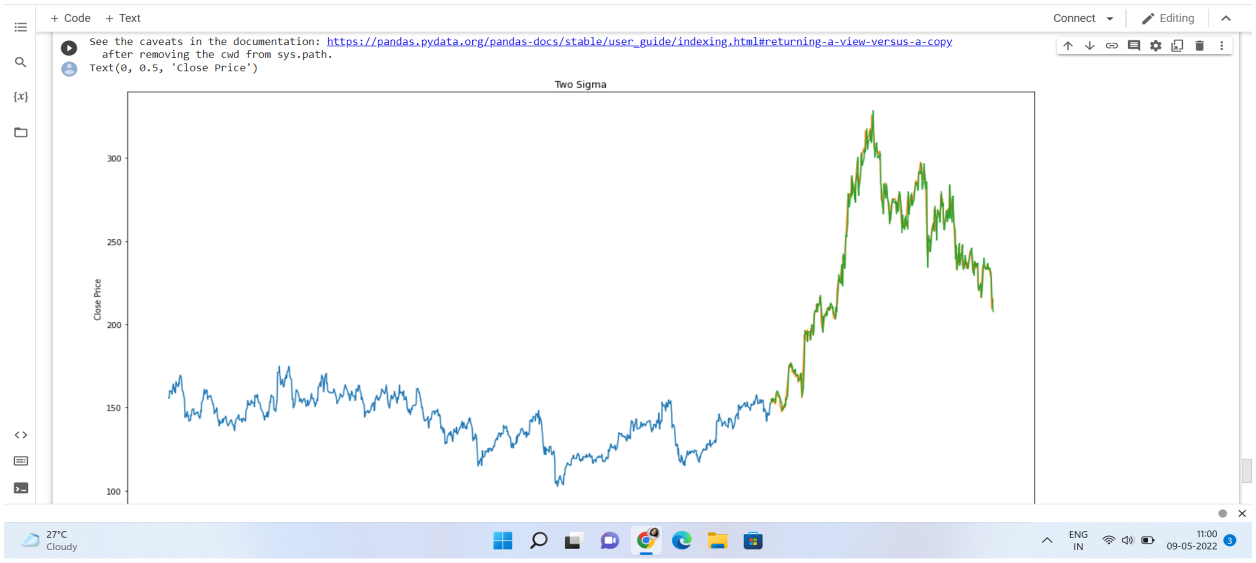
Result

```
rms=np.sqrt(np.mean(np.power((valid-closing_price),2)))
rms
```

```
5.77957835528417
```

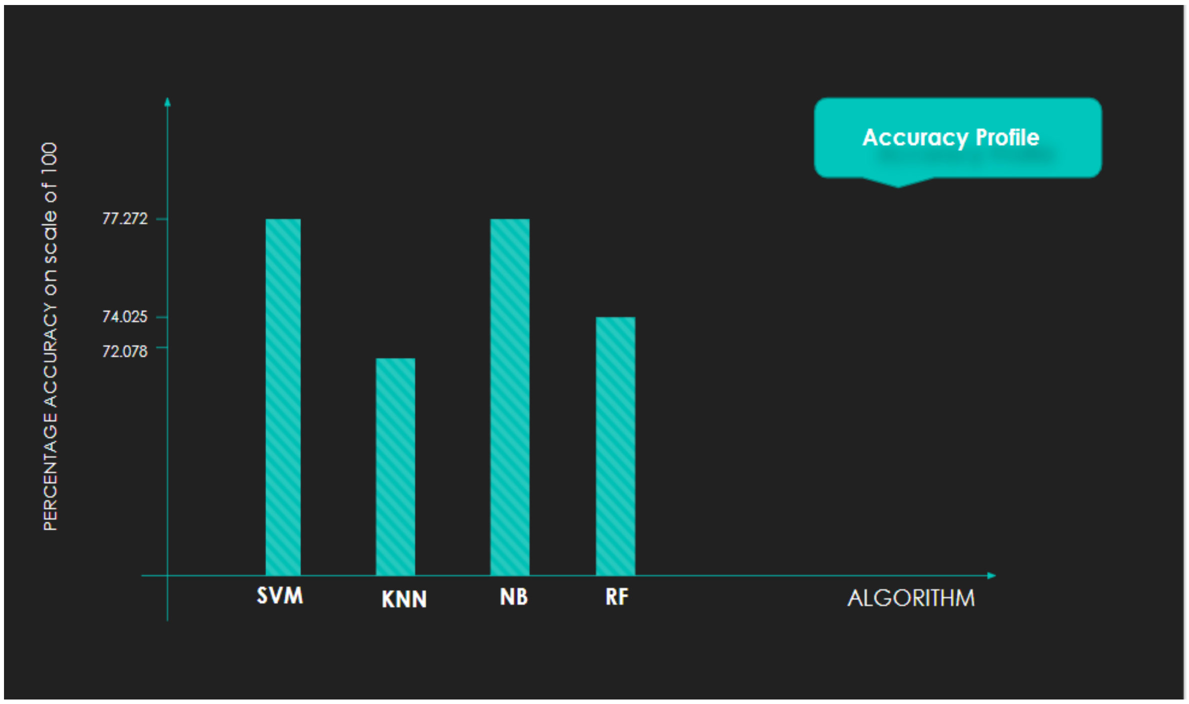
```
#for plotting
train = new_data[:900]
valid = new_data[900:]
valid['Predictions'] = closing_price
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])

plt.title('Two Sigma')
plt.xlabel('Days')
plt.ylabel('Close Price')
```



```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
Text(0, 0.5, 'Close Price')
```

Result:



Conclusion

Plotting stock prices along a normal distribution can allow traders to see when the stock is overbought or oversold.

Using LSTM, time series forecasting models can predict future values based on previous, sequential data. This provides greater accuracy for demand forecasters which results in better decision making for the business.
So, LSTM is the best model for predicting stock prices.