TECHNOCOLABS DATA ANALYSIS INTERNSHIP

PROJECT REPORT

TITLE: Intraday -Stock-Trading DST ML Project

AIM:

The principle focus of our project is to perform data analysis and train a model using the most popular Machine Learning algorithm – SVM in order to analyze the day-to-day price movements in stock market.

ABSTRACT:

Intraday trading, also called day trading, is the buying and selling of stocks and other financial instruments within the same day. In other words, intraday trading means all positions are squared-off before the market closes and there is no change in ownership of shares as a result of the trades.

Until recently, people perceived day trading to be the domain of financial firms and professional traders. But this has changed today, thanks to the popularity of electronic trading and margin trading.

Today, it is extremely easy to start day trading. If you want to start, read on to understand the basics of intraday trading:

PROJECT OBJECTIVE:

Building a model based on supervised learning which have ability to **Precisely predict the price movement of stocks** is the key to profitability in trading. In light of the increasing availability of financial data, prediction of price movement in the financial market with machine learning has become a topic of interests for both investors and researchers. Our group aims to predict whether the price in the next minute will go up or down, using the time series data of stock price, technical-analysis indicators, and trading volume

OVERVIEW:

- Data Segmentation and Data Cleaning
- Exploratory Data Analysis using python's data visualization
- Training the model based on the historical data available
- Deployment of model using Heroku Platform and Flask framework

DATASET:

There are two datasets:

1. MSFT-Stock-Trading
2. NIFTY-Stock-Trading

So, our data set comprises of 7 columns, out of which 6 are features and 1 is target.

Features

- Open — It is the opening price of a shares for that day
- High — It is the highest price the shares have touched in throughout day
- Low — It is the lowest price the shares have fallen to in throughout day
- Date — The date of the observation, mostly the index of our data
- Volume — The number of shares sold in throughout day

Target:

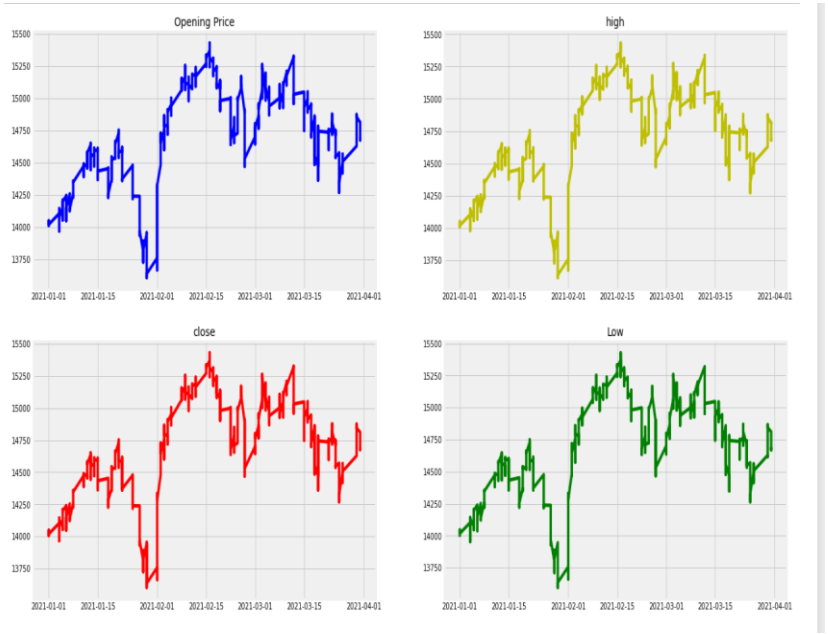
- Close — It is the closing price of the shares for that day

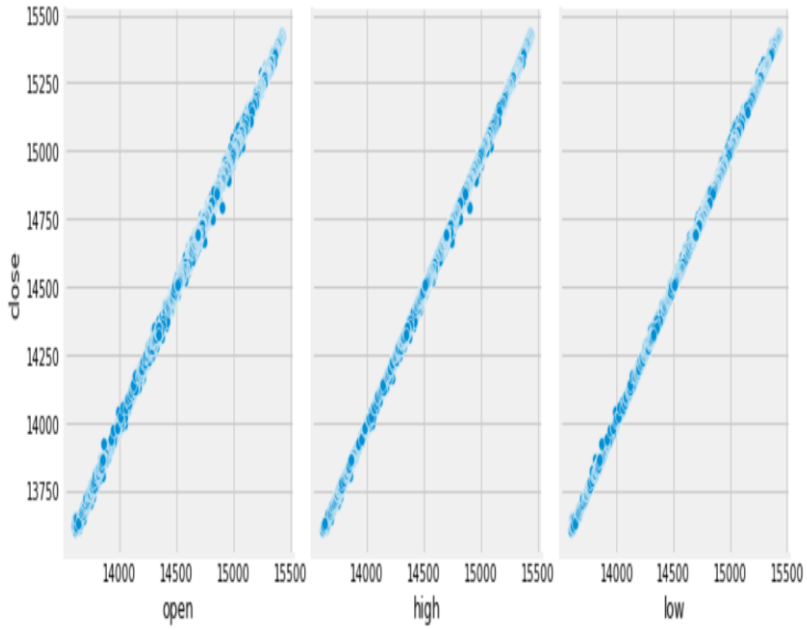
DATA SEGMENTATION AND DATA CLEANING:

- In this project, we have prepared a processed dataset by and collected the clear-cut data available online.
- Using panda's data frame, we have calculated the mean of every column.
- By using the fill-na we have filled all the cells with empty values.

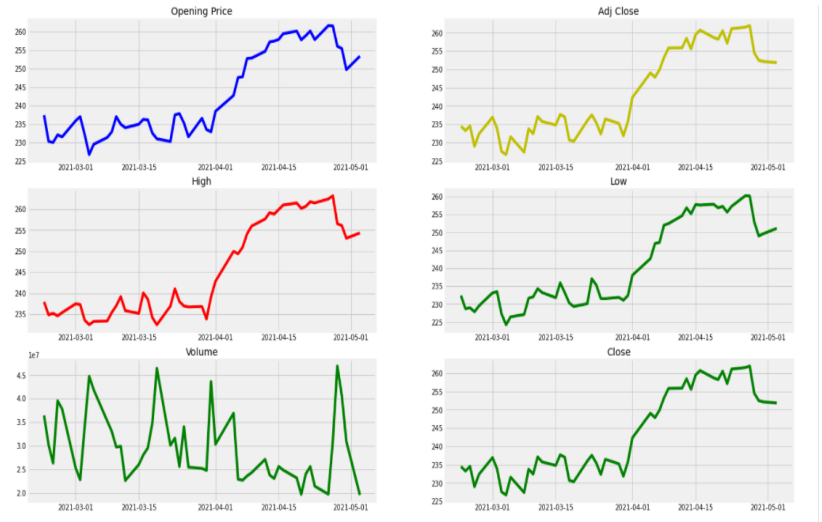
EXPLORATORY DATA ANALYSIS:

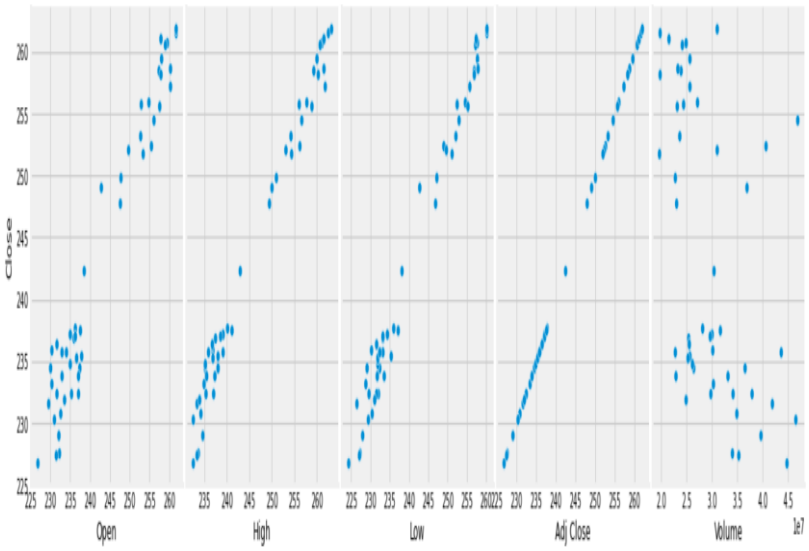
- **PLOT 1: CORRELATIONS (NIFTY)**





➢ PLOT 2 : CORRELATIONS (MSFT)

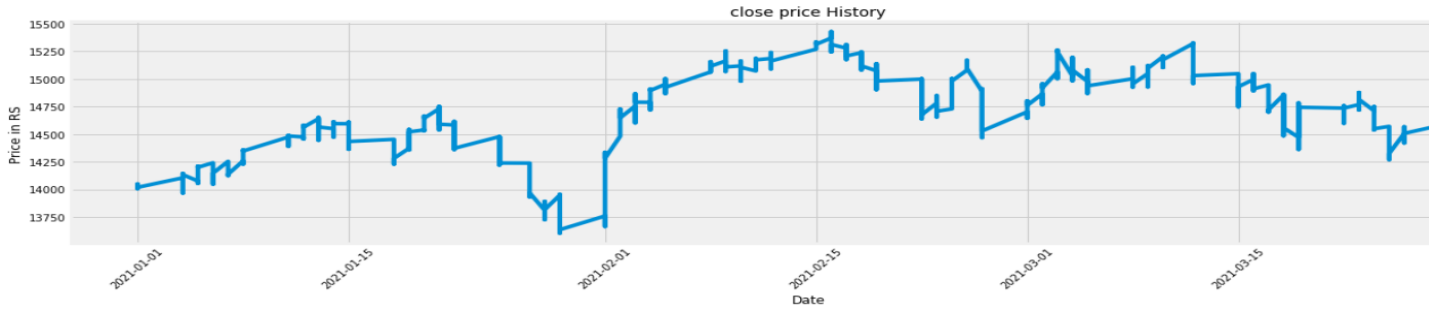




Insight: We have found that there is high variance between the all the variables with respect to target variables, so we decided to keep “Open Price” and remove the rest for better model

- **PLOT 3 : Close Price VS Date (MSFT)**

- **PLOT 4 : Close Price VS Date (NIFTY)**



Insights: The above interactive plot shows that Price growth over a period time

TRAINING THE MODEL:

NIFTY

Data View:

	Type	Date	Time	open	high	low	close
22800	NIFTY	20210331	15:27	14687.35	14694.35	14686.30	14690.60
22801	NIFTY	20210331	15:28	14690.90	14690.90	14686.05	14689.30
22802	NIFTY	20210331	15:29	14690.30	14694.10	14688.55	14691.30
22803	NIFTY	20210331	15:30	14689.85	14693.55	14689.45	14693.05
22804	NIFTY	20210331	15:31	14690.70	14690.70	14690.70	14690.70

Feature Engineering:

- **Added Extra Important features before model fit (Month, Time, Day)**

```
#DAte column Feature enginearing

df["Datetime"]=df["Date"].astype("str")+df["Time"]
df["Datetime"]=df["Datetime"].str.replace(":","")
df=df.drop(["Date","Time"],1)


m=[]
t=[]
d=[]

for i in df["Datetime"]:
    m.append(i[4:6])
    t.append(i[8:])
    d.append(i[6:8])

df.drop("Datetime",1,inplace=True)
df["month"]=m
df["time"]=t
df["day"]=d
df["month"]=pd.DataFrame(df["month"])
df["time"]=pd.DataFrame(df["time"])
df["day"]=pd.DataFrame(df["day"])
df.head()
```

	open	high	low	close	month	time	day
0	13997.90	14020.85	13991.35	14013.15	01	0916	01
1	14014.85	14018.55	14008.15	14009.05	01	0917	01
2	14008.05	14013.10	14005.05	14012.70	01	0918	01
3	14013.65	14019.10	14013.65	14016.20	01	0919	01
4	14015.45	14017.80	14011.95	14015.45	01	0920	01

Train(X) and Test(Y) Split

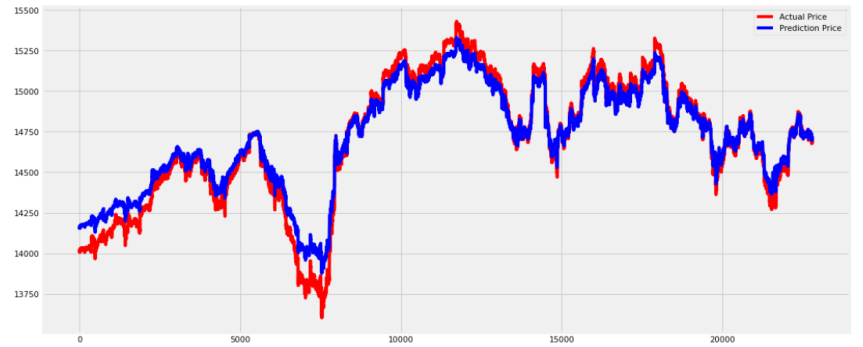
```
X=df[["day","month","time","open"]]
Y=df["close"]
```

Model Building

```
X=np.array(X)
rbf_svr=SVR(kernel="poly")
rbf_svr.fit(X,Y)
```

- **PLOT 5: Comparison between actual Close Price and Prediction Price after Model Building**

```
plt.figure(figsize=(16,8))
plt.plot(b.index,close_price,color="Red",label="Actual Price")
plt.plot(b.index,rbf_svr.predict(k),color="blue",label="Prediction Price")
plt.legend()
plt.show;
```



Model Accuracy

```
Accuracy=rbf_svr.score(X,Y)
```

```
Accuracy
```

```
0.9629549347141364
```

MSFT

Data View:

Date	Open	High	Low	Close	Adj Close	Volume
2021-02-22	237.419998	237.929993	232.399994	234.509995	234.509995	36446900
2021-02-23	230.330002	234.830002	228.729996	233.270004	233.270004	30228700
2021-02-24	230.009995	235.199997	229.000000	234.550003	234.550003	26339700
2021-02-25	232.080002	234.589996	227.880005	228.990005	228.990005	39542200
2021-02-26	231.529999	235.369995	229.539993	232.380005	232.380005	37819200

Feature Engineering:

- **Added Extra Important features before model fit (Month, Year, Day)**

```
m=[]
d=[]
y=[]

for i in df["Date"]:
    m.append(i[4:6])
    d.append(i[6:])
    y.append(i[:4])



df.drop("Date",1,inplace=True)
df["month"]=m
df["day"]=d
df["Year"]=y
df["month"]=pd.DataFrame(df["month"])
df["day"]=pd.DataFrame(df["day"])
df["Year"]=pd.DataFrame(df["Year"])
```

	Open	High	Low	Close	Adj Close	Volume	month	day	Year
0	0.088542	0.101563	0.088542	0.097222	0.061751	1031788800	03	13	1986
1	0.097222	0.102431	0.097222	0.100694	0.063956	308160000	03	14	1986
2	0.100694	0.103299	0.100694	0.102431	0.065059	133171200	03	17	1986
3	0.102431	0.103299	0.098958	0.099826	0.063405	67766400	03	18	1986
4	0.099826	0.100694	0.097222	0.098090	0.062302	47894400	03	19	1986

Train(X) and Test(Y) Split

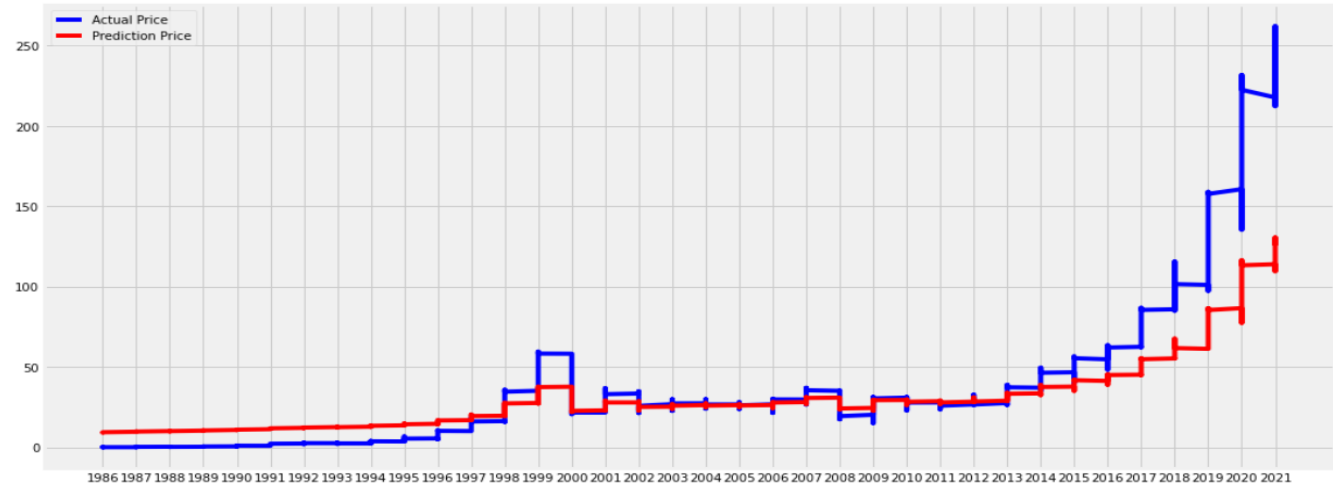
```
X=df[["day","month","Year","Open"]]
b=X.copy()
Y=df["Close"]
```

Model Building

```
X=np.array(X)
rbf_svr=SVR(kernel="poly")
rbf_svr.fit(X,Y)
```

- **PLOT 6: Comparison between actual Close Price and Prediction Price after Model Building**

```
plt.figure(figsize=(16,8))
plt.plot(b.Year,close_price,color="blue",label="Actual Price")
plt.plot(b.Year,rbf_svr.predict(k),color="red",label="Prediction Price")
plt.legend()
plt.show;
```



Model Accuracy

```
Accuracy=rbf_svr.score(k,close_price)
```

```
Accuracy
```

```
0.7034394060232005
```

OVERALL CONCLUSION:

1. **Added extra features to Achieve target variable as per business requirements like (Month, day, year, time)**
2. **Training Set includes 4 independent features Open, Day, Year, Time, Month**
3. **Test Set includes 1 target variable close Price**
4. **Used SVM ML model algorithm to predict the day-to-day price movements using kernel "Polynomial"**
5. **Accuracy Achieved for NIFTY – 0.96% and MSFT – 0.70 %**

MODEL DEPLOYEMENT

Preparing Pickle Files for Both Model

- **MSFT**
- **NIFTY**

```
with open('MSFT-SVM-all', 'wb') as picklefile:
    pickle.dump(rbf_svr,picklefile)
```

```
with open('MSFT-SVM-all', 'rb') as training_model:
    model2 = pickle.load(training_model)
```

```
with open('Nifty-SVM-all', 'wb') as picklefile:
    pickle.dump(rbf_svr,picklefile)
```

```
with open('Nifty-SVM-all', 'rb') as training_model:
    model1 = pickle.load(training_model)
```

Framework used “Flask”

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from flask import Flask,render_template,request
import pickle

model1=pickle.load(open("Nifty-SVM-all","rb"))
model2=pickle.load(open("MSFT-SVM-all","rb"))

app = Flask(__name__)

# Home Page is good

@app.route('/')
def home():
    return render_template("index.html")

#Nifty Page

@app.route('/Nifty',methods=['GET'])
def Nifty():

    return render_template('nifty.html')

#MSFT Page

@app.route('/MSFT',methods=['GET'])
def MSFT():

    return render_template('MSFT.html')
```

Rendering Inputs From “User” using request ( )

- **Prediction1 - NIFTY Prediction**
- **Prediction 2 - MSFT Prediction**

```
@app.route('/predict1',methods=['POST'])
def predict1():


        day = request.form.get("day")
        month =  request.form.get("Month")
        time =request.form.get("Time")
        open =request.form.get("Open")
        data1 =day
        data2 =month
        data3 =time
        data4 =float(open)

        data=pd.DataFrame({"day":data1,"month":data2,"Time":data3,"open":data4},index=[0])
        data=np.array(data)
        prediction=model1.predict(data)
        prediction = round(prediction[0], 2)
        return render_template('nifty.html', prediction_text='Price will be  {}'.format(prediction))



#Model2 Prediction


@app.route('/predict2',methods=["POST"])
def predict2():


        day = request.form.get("day")
        month =  request.form.get("Month")
        Year =request.form.get("Year")
        open =request.form.get("Open")
        data1 =day
        data2 =month
        data3 =Year
        data4 =float(open)

        data=pd.DataFrame({"day":data1,"month":data2,"Year":data3,"open":data4},index=[0])
        data=np.array(data)
        prediction=model2.predict(data)
        prediction = round(prediction[0], 2)
        return render_template('MSFT.html', prediction_text='Price will be  {}'.format(prediction))
```

Scripts/Language Used for deployment

- **Python: HTML: CSS**

App Details

- **Name: stockpriceapi**
- **Link: https://stockpriceapi.herokuapp.com/**

TEAM MEMBERS

1.Rakesh Yadav

2.Bhushan Dhamankar

3Vanita Padma

4.Vaibhav Sharma

5.Swapnil Agarwal