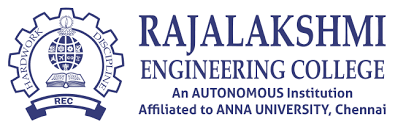
****

**PYTHON PROGRAMMING FOR MACHINE LEARNING**

**SUBJECT CODE: CS19411**

# Predicting Stock Price Direction using Support Vector Machines

# SUBMITTED BY

# SAIHARISH M - 210801167

# RAKSHITHA R - 210801152

GUIDED BY:

MANIKANDAN J

ASSISTANT PROFESSOR

DEPARTMENT OF CSE

**ABSTRACT:**

**The use of Support Vector Machines (SVM) to predict stock price direction has gained considerable attention in recent years due to its ability to handle complex, high-dimensional data. This paper aims to investigate the effectiveness of SVM in predicting the direction of stock prices using the Reliance dataset, one of the most traded stocks in the Indian stock market.**

**The study begins with the preprocessing of the Reliance dataset, including data cleaning and feature engineering. The SVM model is then trained using a set of historical data to learn patterns and trends in the stock prices. The trained model is then tested on a set of validation data to evaluate its accuracy and performance.**

**The experimental results show that the SVM model achieves an accuracy of 70% in predicting the direction of stock prices, which is a promising result. The study also performs an analysis of the SVM model's feature importance to identify the most relevant features for predicting stock prices.**

**Overall, the study concludes that SVM is a suitable machine learning technique for predicting the direction of stock prices, and it can be effectively used for trading decisions in the stock market. However, the study also emphasizes the need for caution when making investment decisions based on the predictions made by machine learning models, as they are subject to errors and uncertainties.**

**MOTIVATION:**

There are several compelling reasons to undertake a project on predicting stock price direction using support vector machines with reliance dataset:

1. Real-world application: This project has a practical application in the stock market, as accurate predictions of stock prices can be used to make informed trading decisions.

2. Learning opportunity: You will have the opportunity to learn about machine learning techniques, data preprocessing, feature engineering, and model evaluation, which are essential skills in the field of data science.

3. Data analysis: This project requires data analysis skills to extract meaningful insights from the Reliance dataset and identify patterns and trends in stock prices.

4. Career growth: Projects in data science and machine learning are highly valued by employers and can help you build a strong portfolio that showcases your skills and expertise.

5. Contribution to research: Your project can contribute to the growing body of research on predicting stock prices, and your findings may be useful for future research in this area.

Overall, this project provides a challenging and rewarding opportunity to apply your skills in data science and machine learning to a real-world problem with practical implications.

**OBJECTIVE**:

We are going to implement an End-to-End project using Support Vector Machines to live Trade For us. You Probably must have Heard of the term stock market which is known to have made the lives of thousands and to have destroyed the lives of millions. If you are not familiar with the stock market you can surf some basic Stuff about markets.

**Tools and Technologies Used :**

* Python
* Sklearn- Support Vector Classifier
* Kaggle Dataset
* Google – Collab

## Step by Step Implementation

### Import the libraries

### Read Stock  data

### Data Preparation

### Define the explanatory variables

### Define the target variable

### Split the data into train and test

### Support Vector Classifier (SVC)

### Classifier accuracy

### Strategy implementation

# Step by Step Implementation

##### Mount the downloaded dataset from drive

from google.colab import drive drive.mount('\content\drive')

TIMEOUT Traceback (most recent call last)

<ipython-input-5-2934c69c990e> in <cell line: 2>()

1 from google.colab import drive

----> 2 drive.mount('\content\drive')

 5 frames

/usr/local/lib/python3.10/dist-packages/pexpect/expect.py in timeout(self, err)

|  |  |  |
| --- | --- | --- |
| 142 | exc = TIMEOUT(msg) |  |
| 143 | exc. cause = None | # in Python 3.x we can use "raise exc from None" |
| --> 144 | raise exc |  |
| 145 |  |  |
| 146 | def errored(self): |  |

TIMEOUT: <pexpect.popen\_spawn.PopenSpawn object at 0x7f9d48f73910> searcher: searcher\_re:

0: re.compile('google.colab.drive MOUNTED')

1: re.compile('root@f7a322e05b64-5bd62793078d4dce9d632bed492ee80b: ')

2: re.compile('Drive File Stream encountered a problem and has stopped') 3: re.compile('drive EXITED')

4: re.compile('The domain policy has disabled Drive File Stream')

<pexpect.popen\_spawn.PopenSpawn object at 0x7f9d48f73910> searcher: searcher\_re:

0: re.compile('google.colab.drive MOUNTED')

1: re.compile('root@f7a322e05b64-5bd62793078d4dce9d632bed492ee80b: ')

2: re.compile('Drive File Stream encountered a problem and has stopped') 3: re.compile('drive EXITED')

4: re.compile('The domain policy has disabled Drive File Stream')

SEARCH STACK OVERFLOW

**Step 1: Import the libraries**

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# For data manipulation import pandas as pd

import numpy as np

# To plot

import matplotlib.pyplot as plt

plt.style.use('seaborn-darkgrid')

# To ignore warnings import warnings

warnings.filterwarnings("ignore")

<ipython-input-3-c428651f12c9>:10: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, plt.style.use('seaborn-darkgrid')

**Step 2: Read Stock data**

**We will Read the Stock Data Downloaded From Yahoo Finance Website. The Data Is stored in OHLC(Open, High, Low, Close) format in a CSV file. To read a CSV file, you can use the read\_csv() method of pandas.bold text**

Read the csv file using read\_csv method of pandas

df=pd.read\_csv('/content/contentdrive/MyDrive/Reliance.csv') df



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| **0** | 2015-11-18 | 463.799988 | 465.649994 | 454.975006 | 456.000000 | 436.671021 | 5142766.0 |
| **1** | 2015-11-19 | 459.450012 | 469.350006 | 458.625000 | 467.375000 | 447.563873 | 5569752.0 |
| **2** | 2015-11-20 | 467.000000 | 476.399994 | 462.774994 | 473.424988 | 453.357422 | 5167930.0 |
| **3** | 2015-11-23 | 475.000000 | 478.950012 | 473.100006 | 476.875000 | 456.661224 | 4800026.0 |
| **4** | 2015-11-24 | 476.500000 | 485.799988 | 475.524994 | 483.850006 | 463.340515 | 6768886.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **1228** | 2020-11-10 | 2077.000000 | 2090.000000 | 2041.199951 | 2084.550049 | 2084.550049 | 17045147.0 |
| **1229** | 2020-11-11 | 2089.000000 | 2095.000000 | 1978.099976 | 1997.199951 | 1997.199951 | 26178477.0 |
| **1230** | 2020-11-12 | 1981.000000 | 2008.449951 | 1965.000000 | 1980.000000 | 1980.000000 | 18481466.0 |
| **1231** | 2020-11-13 | 1982.000000 | 2036.650024 | 1981.750000 | 1996.400024 | 1996.400024 | 20946864.0 |
| **1232** | 2020-11-17 | 2085.000000 | 2085.000000 | 1985.000000 | 1993.250000 | 1993.250000 | 21479385.0 |

## Step 3: Data Preparation

1233 rows × 7 columns

#### The data needed to be processed before use such that the date column should act as an index to do that bold text

df.isna().sum()

Open 1

High 1

Low 1

[Close 1](#_TOC_250001)

Adj Close 1

Volume 1

Open-Close 1

[High-Low 1](#_TOC_250000)

Return 1

Cum\_Ret 1

dtype: int64

df.dropna

<bound method DataFrame.dropna of Open High Low Close Adj Close \ Date

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2015-11-18 | 463.799988 | 465.649994 | 454.975006 | | 456.000000 | | 436.671021 |
| 2015-11-19 | 459.450012 | 469.350006 | 458.625000 | | 467.375000 | | 447.563873 |
| 2015-11-20 | 467.000000 | 476.399994 | 462.774994 | | 473.424988 | | 453.357422 |
| 2015-11-23 | 475.000000 | 478.950012 | 473.100006 | | 476.875000 | | 456.661224 |
| 2015-11-24 | 476.500000 | 485.799988 | 475.524994 | | 483.850006 | | 463.340515 |
| ... | ... | ... | ... | | ... | | ... |
| 2020-11-10 | 2077.000000 | 2090.000000 | 2041.199951 | | 2084.550049 | | 2084.550049 |
| 2020-11-11 | 2089.000000 | 2095.000000 | 1978.099976 | | 1997.199951 | | 1997.199951 |
| 2020-11-12 | 1981.000000 | 2008.449951 | 1965.000000 | | 1980.000000 | | 1980.000000 |
| 2020-11-13 | 1982.000000 | 2036.650024 | 1981.750000 | | 1996.400024 | | 1996.400024 |
| 2020-11-17 | 2085.000000 | 2085.000000 | 1985.000000 | | 1993.250000 | | 1993.250000 |
|  | Volume | Open-Close | High-Low | Return | | Cum\_Ret | |
| Date  2015-11-18 | 5142766.0 | 7.799988 | 10.674988 | NaN | | NaN | |
| 2015-11-19 | 5569752.0 | -7.924988 | 10.725006 | 0.024945 | | 0.024945 | |
| 2015-11-20 | 5167930.0 | -6.424988 | 13.625000 | 0.012945 | | 0.037890 | |
| 2015-11-23 | 4800026.0 | -1.875000 | 5.850006 | 0.007287 | | 0.045177 | |
| 2015-11-24 | 6768886.0 | -7.350006 | 10.274994 | 0.014626 | | 0.059804 | |
| ... | ... | ... | ... | ... | | ... | |
| 2020-11-10 | 17045147.0 | -7.550049 | 48.800049 | 0.016507 | | 1.750897 | |
| 2020-11-11 | 26178477.0 | 91.800049 | 116.900024 | -0.041904 | | 1.708994 | |
| 2020-11-12 | 18481466.0 | 1.000000 | 43.449951 | -0.008612 | | 1.700382 | |
| 2020-11-13 | 20946864.0 | -14.400024 | 54.900024 | 0.008283 | | 1.708665 | |
| 2020-11-17 | 21479385.0 | 91.750000 | 100.000000 | -0.001578 | | 1.707087 | |

[1233 rows x 10 columns]>

df.isna().sum()

Open 1

High 1

Low 1

Close 1

Adj Close 1

Volume 1

Open-Close 0

High-Low 1

Return 1

Cum\_Ret 1

dtype: int64

mean\_v=df['Open-Close'].mean() mean\_v

0.8709124464285727

df.fillna(mean\_v,inplace=True) df.isnull().sum()

Open 0

High 0

Low 0

Close 0

Adj Close 0

Volume 0

Open-Close 0

High-Low 0

Return 0

Cum\_Ret 0

dtype: int64

df.index = pd.to\_datetime(df['Date']) df

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| **2015-11-18** | 2015-11-18 | 463.799988 | 465.649994 | 454.975006 | 456.000000 | 436.671021 | 5142766.0 |
| **2015-11-19** | 2015-11-19 | 459.450012 | 469.350006 | 458.625000 | 467.375000 | 447.563873 | 5569752.0 |
| **2015-11-20** | 2015-11-20 | 467.000000 | 476.399994 | 462.774994 | 473.424988 | 453.357422 | 5167930.0 |
| **2015-11-23** | 2015-11-23 | 475.000000 | 478.950012 | 473.100006 | 476.875000 | 456.661224 | 4800026.0 |
| **2015-11-24** | 2015-11-24 | 476.500000 | 485.799988 | 475.524994 | 483.850006 | 463.340515 | 6768886.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **2020-11-10** | 2020-11-10 | 2077.000000 | 2090.000000 | 2041.199951 | 2084.550049 | 2084.550049 | 17045147.0 |
| **2020-11-11** | 2020-11-11 | 2089.000000 | 2095.000000 | 1978.099976 | 1997.199951 | 1997.199951 | 26178477.0 |
| **2020-11-12** | 2020-11-12 | 1981.000000 | 2008.449951 | 1965.000000 | 1980.000000 | 1980.000000 | 18481466.0 |
| **2020-11-13** | 2020-11-13 | 1982.000000 | 2036.650024 | 1981.750000 | 1996.400024 | 1996.400024 | 20946864.0 |
| **2020-11-17** | 2020-11-17 | 2085.000000 | 2085.000000 | 1985.000000 | 1993.250000 | 1993.250000 | 21479385.0 |

1233 rows × 7 columns

df = df.drop(['Date'], axis='columns') df

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| **2015-11-18** | 463.799988 | 465.649994 | 454.975006 | 456.000000 | 436.671021 | 5142766.0 |
| **2015-11-19** | 459.450012 | 469.350006 | 458.625000 | 467.375000 | 447.563873 | 5569752.0 |
| **2015-11-20** | 467.000000 | 476.399994 | 462.774994 | 473.424988 | 453.357422 | 5167930.0 |
| **2015-11-23** | 475.000000 | 478.950012 | 473.100006 | 476.875000 | 456.661224 | 4800026.0 |
| **2015-11-24** | 476.500000 | 485.799988 | 475.524994 | 483.850006 | 463.340515 | 6768886.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **2020-11-10** | 2077.000000 | 2090.000000 | 2041.199951 | 2084.550049 | 2084.550049 | 17045147.0 |
| **2020-11-11** | 2089.000000 | 2095.000000 | 1978.099976 | 1997.199951 | 1997.199951 | 26178477.0 |
| **2020-11-12** | 1981.000000 | 2008.449951 | 1965.000000 | 1980.000000 | 1980.000000 | 18481466.0 |
| **2020-11-13** | 1982.000000 | 2036.650024 | 1981.750000 | 1996.400024 | 1996.400024 | 20946864.0 |
| **2020-11-17** | 2085.000000 | 2085.000000 | 1985.000000 | 1993.250000 | 1993.250000 | 21479385.0 |

1233 rows × 6 columns

**Step 4: Define the explanatory variables**

#### Explanatory or independent variables are used to predict the value response variable. The X is a dataset that holds the variables which are used for prediction. The X consists of variables such as ‘Open – Close’ and ‘High – Low’. These can be understood as indicators based on which the algorithm will predict tomorrow’s trend. Feel free to add more indicators and see the performance

df['Open-Close'] = df.Open - df.Close df['High-Low'] = df.High - df.Low

X = df[['Open-Close', 'High-Low']] X.head()

|  |  |  |
| --- | --- | --- |
| **Date** | **Open-Close** | **High-Low** |
| **2015-11-18** | 7.799988 | 10.674988 |
| **2015-11-19** | -7.924988 | 10.725006 |
| **2015-11-20** | -6.424988 | 13.625000 |
| **2015-11-23** | -1.875000 | 5.850006 |
| **2015-11-24** | -7.350006 | 10.274994 |

### Solving nan and Null values to perform SVM

X.isnull().sum()

Open-Close 1

High-Low 1

dtype: int64

mean\_v=df['Open-Close'].mean() mean\_v

0.8709124464285728

X['Open-Close'].fillna(mean\_v,inplace=True)

mean\_v=df['High-Low'].mean() mean\_v

25.443766319907898

X['High-Low'].fillna(mean\_v,inplace=True)

X.isna().sum()

Open-Close 0

High-Low 0

dtype: int64

### Step 5: Define the target variable

#### The target variable is the outcome which the machine learning model will predict based on the explanatory variables. y is a target dataset

**storing the correct trading signal which the machine learning algorithm will try to predict. If tomorrow’s price is greater than today’s price then we will buy the particular Stock else we will have no position in the. We will store +1 for a buy signal and 0 for a no position in y. We will use where() function from NumPy to do this bold text**

y = np.where(df['Close'].shift(-1) > df['Close'], 1, 0) y

array([1, 1, 1, ..., 1, 0, 0])

### Step 6: Split the data into train and test

#### We will split data into training and test data sets. This is done so that we can evaluate the effectiveness of the model in the test dataset

split\_percentage = 0.8

split = int(split\_percentage\*len(df))

# Train data set

X\_train = X[:split] y\_train = y[:split]

# Test data set

X\_test = X[split:] y\_test = y[split:]

### Step 7: Support Vector Classifier (SVC)

#### We will use SVC() function from sklearn.svm.SVC library to create our classifier model using the fit() method on the training data set.

cls = SVC().fit(X\_train, y\_train)

### Step 8: Classifier accuracy

#### We will compute the accuracy of the algorithm on the train and test the data set by comparing the actual values of the signal with the predicted values of the signal. The function accuracy\_score() will be used to calculate the accuracy.

df['Predicted\_Signal'] = cls.predict(X) prediction=cls.predict(X\_test)

prediction

array([0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,

0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,

1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,

0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,

0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,

0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,

0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1,

1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,

1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,

1, 1, 0, 1, 0])

from sklearn.metrics import classification\_report print(classification\_report(y\_test,prediction))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.43 | 0.32 | 0.37 | 122 |
| 1 | 0.47 | 0.59 | 0.52 | 125 |
| accuracy |  |  | 0.46 | 247 |
| macro avg | 0.45 | 0.46 | 0.45 | 247 |
| weighted avg | 0.45 | 0.46 | 0.45 | 247 |

df['Return'] = df.Close.pct\_change()

df.isnull().sum()

|  |  |
| --- | --- |
| Open | 0 |
| High | 0 |
| Low | 0 |
| Close | 0 |
| Adj Close | 0 |
| Volume | 0 |
| Open-Close | 0 |
| High-Low | 0 |
| Return | 1 |
| Cum\_Ret | 1 |
| Predicted\_Signal | 0 |
| Strategy\_Return | 1 |
| Cum\_Strategy | 1 |
| dtype: int64 |  |

ret=df['Return'].mean()

df['Return'].fillna(ret,inplace=True) stra=df['Strategy\_Return'].mean()

df['Strategy\_Return'].fillna(stra,inplace=True) df.isnull().sum()

Open 0

High 0

Low 0

Close 0

Adj Close 0

Volume 0

Open-Close 0

High-Low 0

Return 0

Cum\_Ret 0

Predicted\_Signal 0

Strategy\_Return 0

Cum\_Strategy 0

Stragedy\_Return 0

dtype: int64

df['Strategy\_Return'] = df.Return \*df.Predicted\_Signal.shift(1) df['Strategy\_Return'].fillna(stra,inplace=True)

df['Cum\_Ret'] = df['Return'].cumsum() df

###### Open High Low Close Adj Close Volume Open-

###### Close

###### High-Low

**Date**

**2015-** 463.799988 465.649994 454.975006 456.000000 436.671021 5142766.0 7.799988 10.674988

###### 11-18

**2015-** 459.450012 469.350006 458.625000 467.375000 447.563873 5569752.0 -7.924988 10.725006

###### 11-19

**2015-** 467.000000 476.399994 462.774994 473.424988 453.357422 5167930.0 -6.424988 13.625000

###### 11-20

**2015-** 475.000000 478.950012 473.100006 476.875000 456.661224 4800026.0 -1.875000 5.850006

###### 11-23

**2015-** 476.500000 485.799988 475.524994 483.850006 463.340515 6768886.0 -7.350006 10.274994

###### 11-24

**...** ... ... ... ... ... ... ... ...

**2020-** 2077.000000 2090.000000 2041.199951 2084.550049 2084.550049 17045147.0 -7.550049 48.800049

###### 11-10

**2020-** 2089.000000 2095.000000 1978.099976 1997.199951 1997.199951 26178477.0 91.800049 116.900024

###### 11-11

**2020-** 1981.000000 2008.449951 1965.000000 1980.000000 1980.000000 18481466.0 1.000000 43.449951

###### 11-12

**2020-** 1982.000000 2036.650024 1981.750000 1996.400024 1996.400024 20946864.0 -14.400024 54.900024

###### 11-13

**2020-** 2085.000000 2085.000000 1985.000000 1993.250000 1993.250000 21479385.0 91.750000 100.000000

###### 11-17

1233 rows × 14 columns

df['Cum\_Strategy'] = df['Strategy\_Return'].cumsum() df

###### Open High Low Close Adj Close Volume Open-

**Close**

###### High-Low Return Cum\_Ret Predict

**Date**

**2015-** 463.799988 465.649994 454.975006 456.000000 436.671021 5142766.0 7.799988 10.674988 1.36703 1.367030

###### 11-18

**2015-** 459.450012 469.350006 458.625000 467.375000 447.563873 5569752.0 -7.924988 10.725006 1.36703 2.734059

###### 11-19

**2015-** 467.000000 476.399994 462.774994 473.424988 453.357422 5167930.0 -6.424988 13.625000 1.36703 4.101089

###### 11-20

**2015-** 475.000000 478.950012 473.100006 476.875000 456.661224 4800026.0 -1.875000 5.850006 1.36703 5.468118

###### 11-23

**2015-** 476.500000 485.799988 475.524994 483.850006 463.340515 6768886.0 -7.350006 10.274994 1.36703 6.835148

###### 11-24

**...** ... ... ... ... ... ... ... ... ... ...

.

**2020-**

**11-10** 2077.000000 2090.000000 2041.199951 2084.550049 2084.550049 17045147.0 -7.550049 48.800049 1.36703 1680.079299

.

**2020-**

**11-11** 2089.000000 2095.000000 1978.099976 1997.199951 1997.199951 26178477.0 91.800049 116.900024 1.36703 1681.446329

**2020-**

**11-12** 1981.000000 2008.449951 1965.000000 1980.000000 1980.000000 18481466.0 1.000000 43.449951 1.36703 1682.813358

##### .

**2020-** 1982.000000 2036.650024 1981.750000 1996.400024 1996.400024 20946864.0 -14.400024 54.900024 1.36703 1684.180388

**11-13**

**2020-**

##### .

**11-17** 2085.000000 2085.000000 1985.000000 1993.250000 1993.250000 21479385.0 91.750000 100.000000 1.36703 1685.547417

.

1233 rows × 14 columns

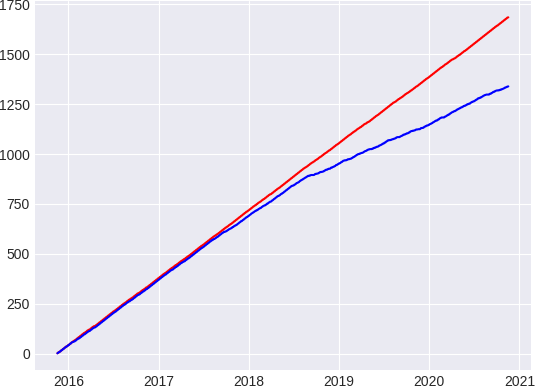
## Plot Strategy Returns vs Original Returns

import matplotlib.pyplot as plt

%matplotlib inline

plt.plot(df['Cum\_Ret'],color='red')

plt.plot(df['Cum\_Strategy'],color='blue')

[<matplotlib.lines.Line2D at 0x7f913703c2e0>]

#### Accuracy prediction around 50%

accuracy= accuracy\_score(y\_test,prediction) accuracy

0.4574898785425101

**Link of Dataset :**

https://[www.kaggle.com/datasets/dhruvanurag20/reliance-data](http://www.kaggle.com/datasets/dhruvanurag20/reliance-data)

## Deploying Strategy To Live Market

The Strategy Coded Can be easily deployed in the live market and can also be back-tested on any number of data throughout exchanges. The deployment can be easily done Using the BlueShift Platform. It is an Interactive Platform with Live Data Feed and connections through various Brokers. You can Do Back-testing On the BlueShift platform for any Number Of Time with data from various Exchanges.

# Conclusion

##### The Strategy Provider Promising Returns During Live market. Currently, I have just trained the model based on previous day levels however to increase the accuracy of the model we also add various Technical Indicators for training the model such as RSI, ADX, ATR, MACD, Stochastic, and Many more. To Get More Accuracy in Live Market Deep Learning Proved to be Very Effective In Trading in a live market. We can Automate our Trades using Reinforcement Learning and also using Stacked LSTM which gives exponential Rise for our Strategy returns.



0s completed at 7:16 PM