Visualizing and Forecasting of Stocks(SBIN)(NSE)

Master of Technology
In
Data and Computational Science

Submitted by

Rakesh Reddy Kadapala – M22AI608

Under the Guidence of
Dr.Sandeep Yadav

Department of Data and Computational Science

Indian Institute of Technology, Jodhpur

Introduction:

Investment:

An investment is an asset or item acquired with the goal of generating income or appreciation. Appreciation refers to an increase in the value of an asset over time.

"Investing" we mean buying an asset for making a profit by selling it in the future, after it appreciates in value.

Project Goal: Visualizing and Forecasting of Stocks (NSE)

Objective: Maximize the Investment Returns

Constraints: Minimize the Investment Risk

NSEpy

NSEpy is a library to extract historical and realtime data from NSE's website. This Library aims to keep the API very simple.

Python is a great tool for data analysis along with the scipy stack and the main objective of NSEpy is to provide analysis ready data-series for use with scipy stack. NSEpy can seamlessly integrate with Technical Analysis library This library would serve as a basic building block for automatic/semi-automatic algorithm trading systems or backtesting systems for Indian markets.

NSE:

National Stock Exchange

Installing NSEpy:

Pip install nsepy

Data Collection:

Data Source:

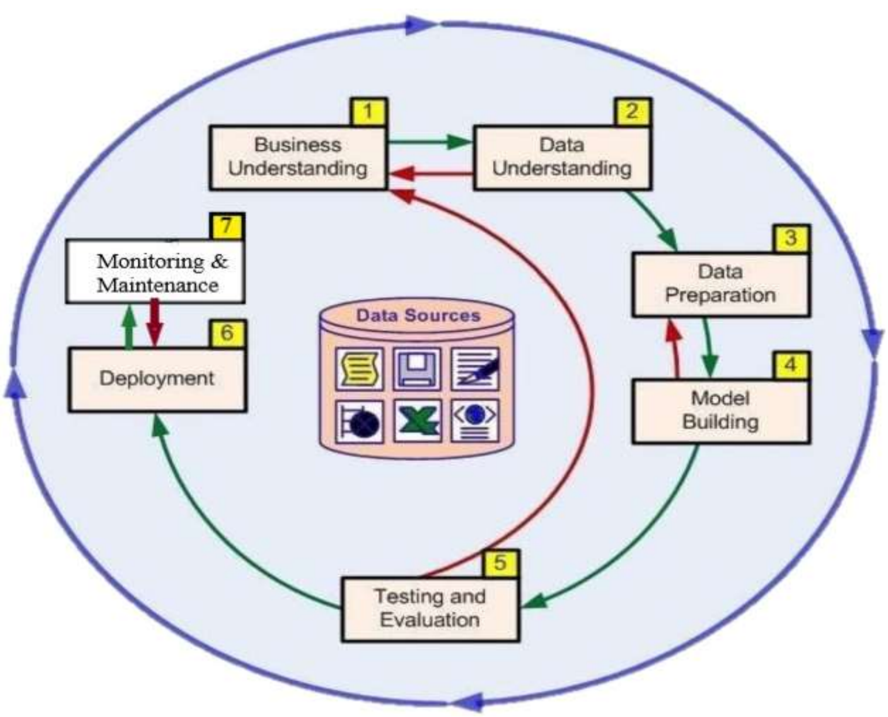
https://www.nseindia.com/get-quotes/equity?symbol=SBIN

Technical Stacks:

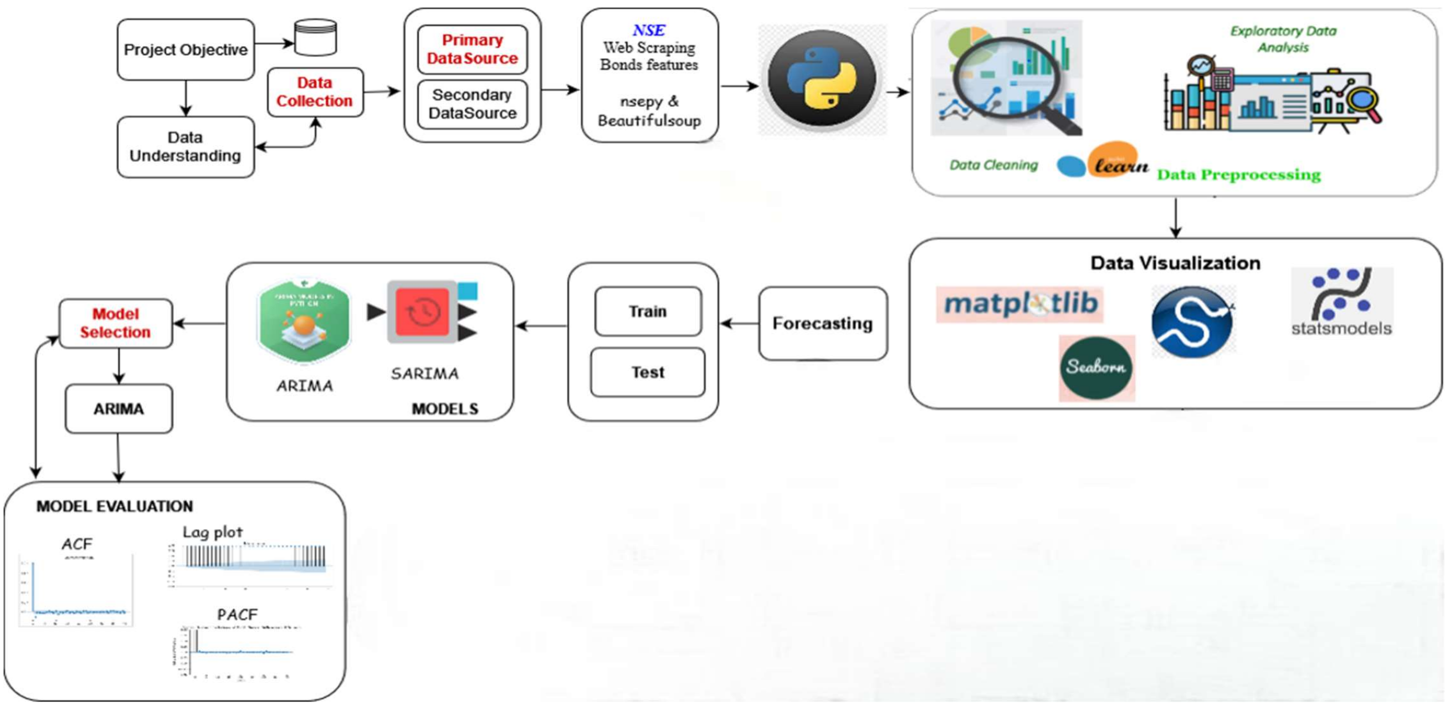
- Python is a general-purpose programming language. We used it for Data Cleaning, EDA, Model Building and Visualization.
- pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python
- Statsmodels is a Python package that allows users to explore data, estimate statistical models, and perform statistical tests.

Data Pre-processing

CRISP-ML Methodology:



Project Architecture/Data Pipeline:



Data Understanding:

Name of Feature	Description
Date(Index)	Date format(YYYY/MM/DD)
Open / Price	Opening price of the bond
High	Highest Price of the bond
Low	Low Price of the bond
Close	Closing price of the bond
Change Pct	Change in the Open Price of Present-Day Price and Close Price of Previous day Price

EDA: Exploratory Data Analysis:

- Exploratory Data Analysis (EDA) is **an approach to analyze the data using visual techniques**. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

```
symbol = "SBIN"
start = date(2015, 1, 1)
end = date.today()
sbin = get_history(symbol=symbol, start=start, end=end)
sbin
```

- Extracting data from 2015/1/1 to present date
- Initial dataset was having 1941 rows and 15 columns.
- The main constrain is stocks available for Monday to Friday, So last day of week prices are considering to next 2 days (Saturday, Sunday)

Following steps were taken to perform Exploratory Data Analysis using Python:

important packages

```
from nsepy import get_history
from datetime import date
import pandas as pd
import numpy as np
import dtale as dt
from statsmodels.tsa.seasonal import seasonal_decompose
from dateutil.parser import parse
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

Data Analysis:

1. Getting column data
2. Resetting Index
3. Sorting data a/c to Date
4. Duplicates
5. Null Values

```
In [18]: sbin.columns
Out[18]: Index(['Symbol', 'Series', 'Prev Close', 'Open', 'High', 'Low', 'Last',
                'Close', 'VWAP', 'Volume', 'Turnover', 'Trades', 'Deliverable Volume',
                '%Deliverble'],
               dtype='object')
```

```
In [19]: sbin = sbin.reset_index()
         sbin.head(3)
Out[19]:
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
0	2015-01-01	SBIN	EQ	311.85	312.45	315.0	310.70	314.0	314.00	313.67	6138488	1.925489e+14	58688	1877677	0.3059
1	2015-01-02	SBIN	EQ	314.00	314.35	318.3	314.35	315.6	315.25	316.80	9935094	3.147389e+14	79553	4221685	0.4249
2	2015-01-05	SBIN	EQ	315.25	316.25	316.8	312.10	312.8	312.75	313.84	9136716	2.867432e+14	88236	3845173	0.4208

```
In [21]: sbin.sort_values(
           by="Date",
           ascending=False
          )
Out[21]:
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
1940	2022-11-04	SBIN	EQ	584.90	586.00	596.95	580.40	595.70	593.95	587.64	25759031	1.513705e+15	302456	10018968	0.3889
1939	2022-11-03	SBIN	EQ	573.85	569.00	587.95	568.00	584.15	584.90	581.58	17557998	1.021139e+15	244825	5121059	0.2917
1938	2022-10-31	SBIN	EQ	570.75	574.95	577.45	568.40	573.50	573.80	572.79	9894639	5.667560e+14	139535	3759140	0.3799
			EQ	579.65	579.95	582.65	567.00	571.80	570.75	572.65	10043644	5.751516e+14	144061	3054910	0.4319

```
#Duplicate Values
print("There are", sbin.duplicated().sum(),'duplicated values in dataset')
```

```
There are 0 duplicated values in dataset
```

Adding missing days:

```
# Adding saturday and sunday dates
```

```
sbin_data = pd.date_range(start=sbin1.Date.min(), end=sbin1.Date.max())
```

```
sbin_data = pd.DataFrame(sbin_data)
```

```
sbin_data['Price'] = ""
```

```
sbin_data.head(2)
```

	0	Price
0	2015-01-01	
1	2015-01-02	

Creating Data frame for Price and Date columns:

```
for i in range(len(sbin1['Date'])):
    for j in range(len(sbin_data[0])):
        if sbin1['Date'][i] == sbin_data[0][j]:
            sbin_data['Price'][j] = sbin1['Close'][i]
```

```
sbin_data.head(10)
```

	0	Price
0	2015-01-01	314.0
1	2015-01-02	315.25
2	2015-01-03	
3	2015-01-04	
4	2015-01-05	312.75

Replacing Null Values in Empty Rows:

```
#replacing nan in empty places
sbin_data['Price'] = sbin_data['Price'].replace('',np.nan)
sbin_data.head(5)
```

	0	Price
0	2015-01-01	314.00
1	2015-01-02	315.25
2	2015-01-03	NaN
3	2015-01-04	NaN
4	2015-01-05	312.75

```
#adding previous day prices to sat and sunday
sbin_data['Price'] = sbin_data['Price'].ffill()
sbin_data.head(5)
```

	0	Price
0	2015-01-01	314.00
1	2015-01-02	315.25
2	2015-01-03	315.25
3	2015-01-04	315.25

Count of Null Values:

```
#Null values
sbin_data.isna().sum()
```

```
0        0
Price    0
dtype: int64
```

Counting Rows and Columns:

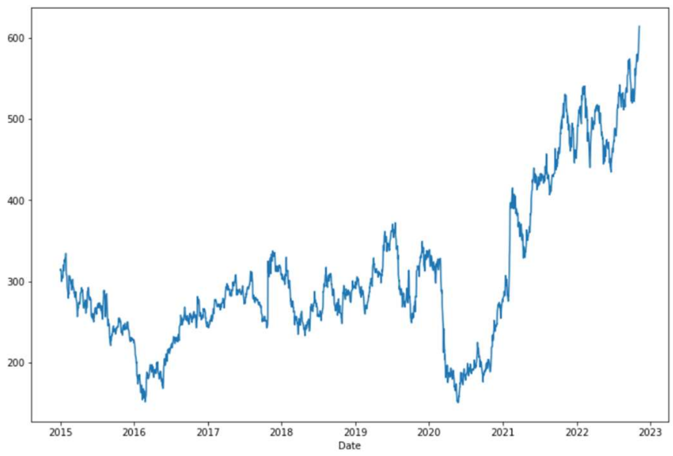
```
sbin_data.shape
```

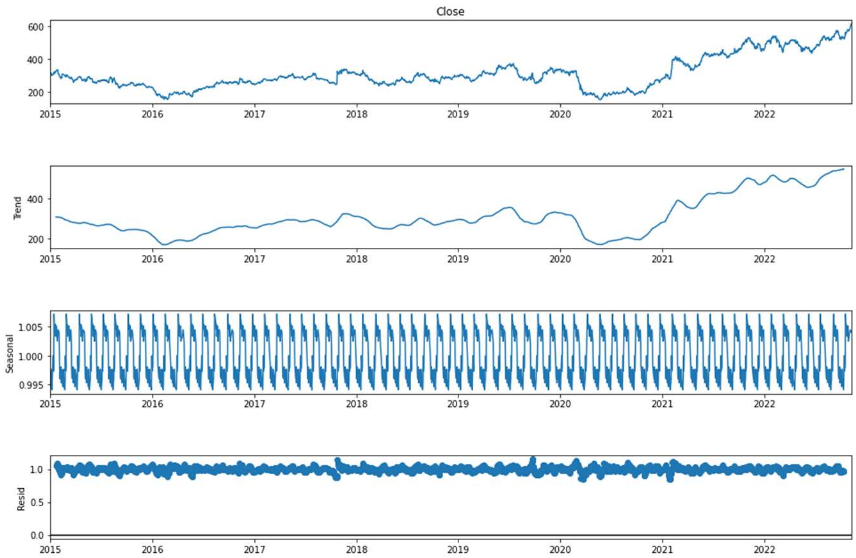
```
(2865, 2)
```

Visualization:

```
sbin['Close'].plot(figsize=(12,8))
```

<AxesSubplot:xlabel='Date'>





Plotting Price values a/c to year wise,

- Trend, as its name suggests, is the overall direction of the data.
- Seasonality is a periodic component
- Residual is what's left over when the trend and seasonality have been removed. Residuals are random fluctuations. You can think of them as a noise component.

- By observing Plot Data is not stationary it is seasonal. We need to use the Seasonal ARIMA (SARIMA) model for Time Series Forecasting.

We need to use the Seasonal ARIMA (SARIMA) model for Time Series Forecasting on this data.

Stationary time series is one whose properties do not depend on the time

Properties:

- Mean -- constant mean
- Variance -- variance should be constant with time
- Auto correlation -- correlation b/w to points depends on distance b/w 2points (lags b/w 2 points)

Checking for stationary with Dickey-fuller Test

- low Pvalue(lower than 0.05) implies series is stationary
- High PValue(greater than 0.05)implies not stationary

Import Adfuller Library

- from statsmodels.tsa.statstools import adfuller
- from statsmodels.tsa.stattools import adfuller#library for finding d

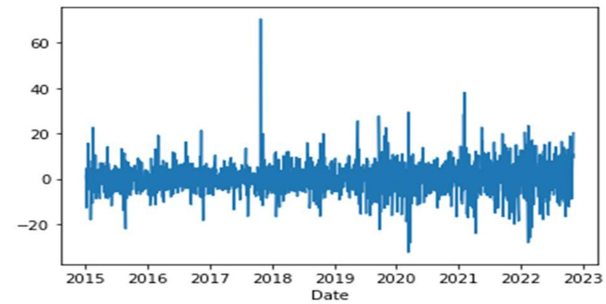
```
# ADF Test
result = adfuller(series, autolag='AIC')
#Extracting the values from the results:

print('ADF Statistic: %f' % result[0])

print('p-value: %f' % result[1])

print('Critical Values:')

for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
if result[0] < result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

```
ADF Statistic: 0.153092
p-value: 0.969504
Critical Values:
        1%: -3.434
        5%: -2.863
        10%: -2.568
Failed to Reject Ho - Time Series is Non-Stationary
```

```
from numpy import sqrt,mean,log,diff
series1=sbin['Close'].diff()
```

```
series1.plot()
```

```
<AxesSubplot:xlabel='Date'>
```




```
adf1=adfuller(series1.dropna())
print("Pvalue of ADF test is:",adf1[1])
```

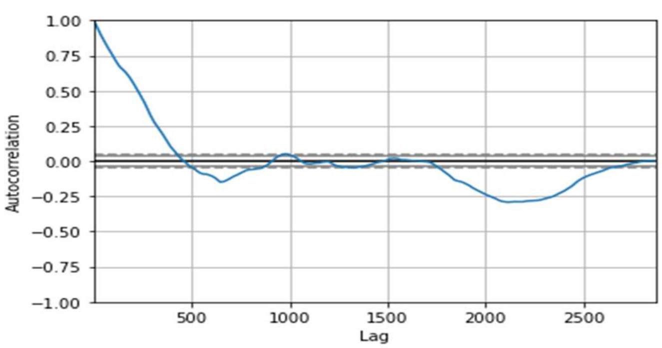
```
Pvalue of ADF test is: 6.719950430072117e-30
```

From Augmented Dickey-Fuller unit root test, P value is 0.96<0.05 implies stationary due to trend. For making stationary differentiating ADF we can remove trend.

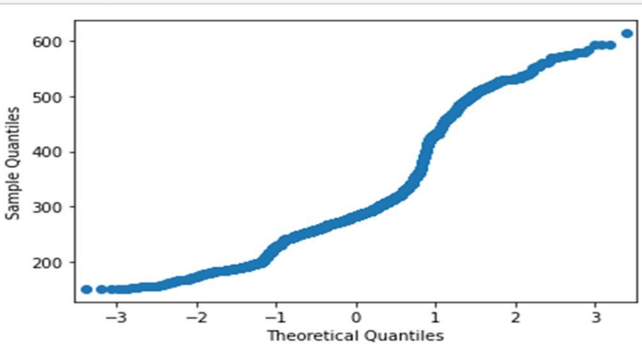
Autocorrelation is the correlation between two observations at different points in a time series.

```
pd.plotting.autocorrelation_plot(sbin_data["Price"])
```

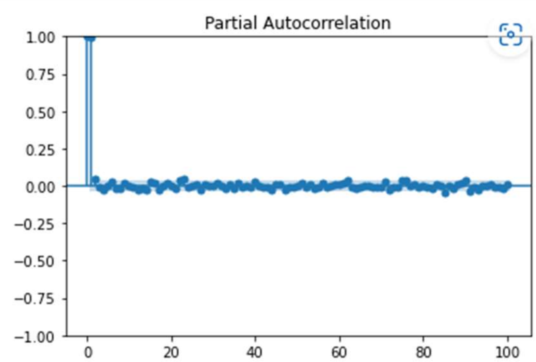
```
<AxesSubplot:xlabel='Lag', ylabel='Autocorrelation'>
```

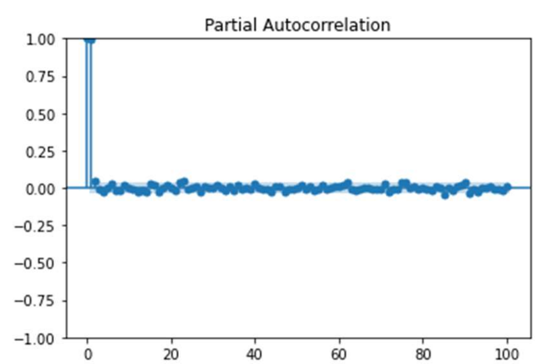



```
from statsmodels.graphics.gofplots import qqplot

qqplot(sbin_data['Price'])
plt.show()
```




```
#Arima
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(sbin_data["Price"], lags = 100)
```





- In the above autocorrelation plot, the curve is moving down after the 5th line of the first boundary. That is how to decide the p-value. Hence the value of p is 5.
- In the above partial autocorrelation plot, we can see that only two points are far away from all the points. That is how to decide the q value. Hence the value of q is 2.

ARIMA(Autoregressive Integrated Moving Average)

- Time Series Forecasting means analyzing and modeling time-series data to make future decisions.
- Arima is one of the statistical method for forecasting Time series data
- ARIMA models have three parameters like ARIMA(p, d, q).
- p is the number of lagged values that need to be added or subtracted from the values (label column). It captures the autoregressive part of ARIMA.
- d represents the number of times the data needs to differentiate to produce a stationary signal. If it's stationary data, the value of d should be 0, and if it's seasonal data, the value of d should be 1. d captures the integrated part of ARIMA.
- q is the number of lagged values for the error term added or subtracted from the values (label column). It captures the moving average part of ARIMA.

```
p, d, q = 5, 1, 2
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(sbin_data["Price"], order=(p,d,q))
fitted = model.fit()
print(fitted.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Price   No. Observations:                 2868
Model:                 ARIMA(5, 1, 2)   Log Likelihood               -8918.929
Date:                Mon, 07 Nov 2022   AIC                          17853.858
Time:                        19:44:14   BIC                          17901.546
Sample:                             0   HQIC                         17871.051
                               - 2868
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3236      0.830      0.390      0.697      -1.303       1.950
ar.L2          0.4548      0.571      0.796      0.426      -0.665       1.575
ar.L3          0.0532      0.029      1.850      0.064      -0.003       0.110
ar.L4          0.0044      0.035      0.127      0.899      -0.064       0.072
ar.L5         -0.0270      0.026     -1.048      0.295      -0.078       0.024
ma.L1         -0.3517      0.831     -0.423      0.672      -1.980       1.277
ma.L2         -0.4344      0.591     -0.735      0.462      -1.593       0.724
sigma2        29.4855      0.287    102.729      0.000      28.923      30.048
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             25185.11
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.37   Skew:                             0.90
Prob(H) (two-sided):                  0.00   Kurtosis:                        17.41
===================================================================================
```

```
predictions = fitted.predict()
print(predictions)
```

```
0         0.000000
1       313.999754
2       315.216970
3       315.263195
4       315.306308
           ...
2863    573.966490
2864    584.740594
2865    593.856663
2866    594.623270
2867    594.676464
Name: predicted_mean, Length: 2868, dtype: float64
```

he predicted values are wrong because the data is seasonal.

ARIMA model will never perform well on seasonal time series data. So, here's how to build a SARIMA model:

SARIMA model

SARIMA stands for Seasonal-ARIMA and it includes seasonality contribution to the forecast. The importance of seasonality is quite evident and ARIMA fails to encapsulate that information implicitly.

The Autoregressive (AR), Integrated (I), and Moving Average (MA) parts of the model remain as that of ARIMA. The addition of Seasonality adds robustness to the SARIMA model. It's represented as:

$$\text{ARIMA} \quad \underbrace{(p, d, q)}_{\substack{\uparrow \\ \text{Non-seasonal part} \\ \text{of the model}}} \quad \underbrace{(P, D, Q)_m}_{\substack{\uparrow \\ \text{Seasonal part of} \\ \text{of the model}}}$$

where m is the number of observations per year. We use the uppercase notation for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model.

Similar to ARIMA, the P,D,Q values for seasonal parts of the model can be deduced from the ACF and PACF plots of the data.

```python
#sarima
import statsmodels.api as sm
import warnings
model=sm.tsa.statespace.SARIMAX(sbin_data['Price'],
                                order=(p, d, q),
                                seasonal_order=(p, d, q, 12))
model=model.fit()
print(model.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                               Price   No. Observations:                 2868
Model:             SARIMAX(5, 1, 2)x(5, 1, 2, 12)   Log Likelihood               -8904.124
Date:                             Mon, 07 Nov 2022   AIC                          17838.248
Time:                                     19:47:22   BIC                          17927.601
Sample:                                          0   HQIC                         17870.469
                                            - 2868
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8062      2.126     -0.379      0.705      -4.973       3.361
ar.L2         -0.3088      1.206     -0.256      0.798      -2.672       2.055
ar.L3          0.0483      0.022      2.230      0.026       0.006       0.091
ar.L4          0.0591      0.118      0.501      0.616      -0.172       0.290
ar.L5          0.0138      0.102      0.136      0.892      -0.186       0.214
ma.L1          0.7802      2.125      0.367      0.714      -3.385       4.946
ma.L2          0.3015      1.154      0.261      0.794      -1.961       2.564
ar.S.L12      -0.9520      0.062    -15.384      0.000      -1.073      -0.831
ar.S.L24      -0.0052      0.028     -0.185      0.854      -0.060       0.050
ar.S.L36      -0.0467      0.026     -1.827      0.068      -0.097       0.003
ar.S.L48      -0.0352      0.028     -1.249      0.212      -0.090       0.020
ar.S.L60      -0.0011      0.021     -0.053      0.958      -0.042       0.040
ma.S.L12      -0.0269      0.056     -0.479      0.632      -0.137       0.083
ma.S.L24      -0.9603      0.057    -16.922      0.000      -1.071      -0.849
sigma2        29.3450      0.304     96.486      0.000      28.749      29.941
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):             24729.54
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               2.39   Skew:                             0.88
Prob(H) (two-sided):                  0.00   Kurtosis:                        17.31
===================================================================================
```

```python
predictions = model.predict(len(sbin_data), len(sbin_data)+10)
print(predictions)
```

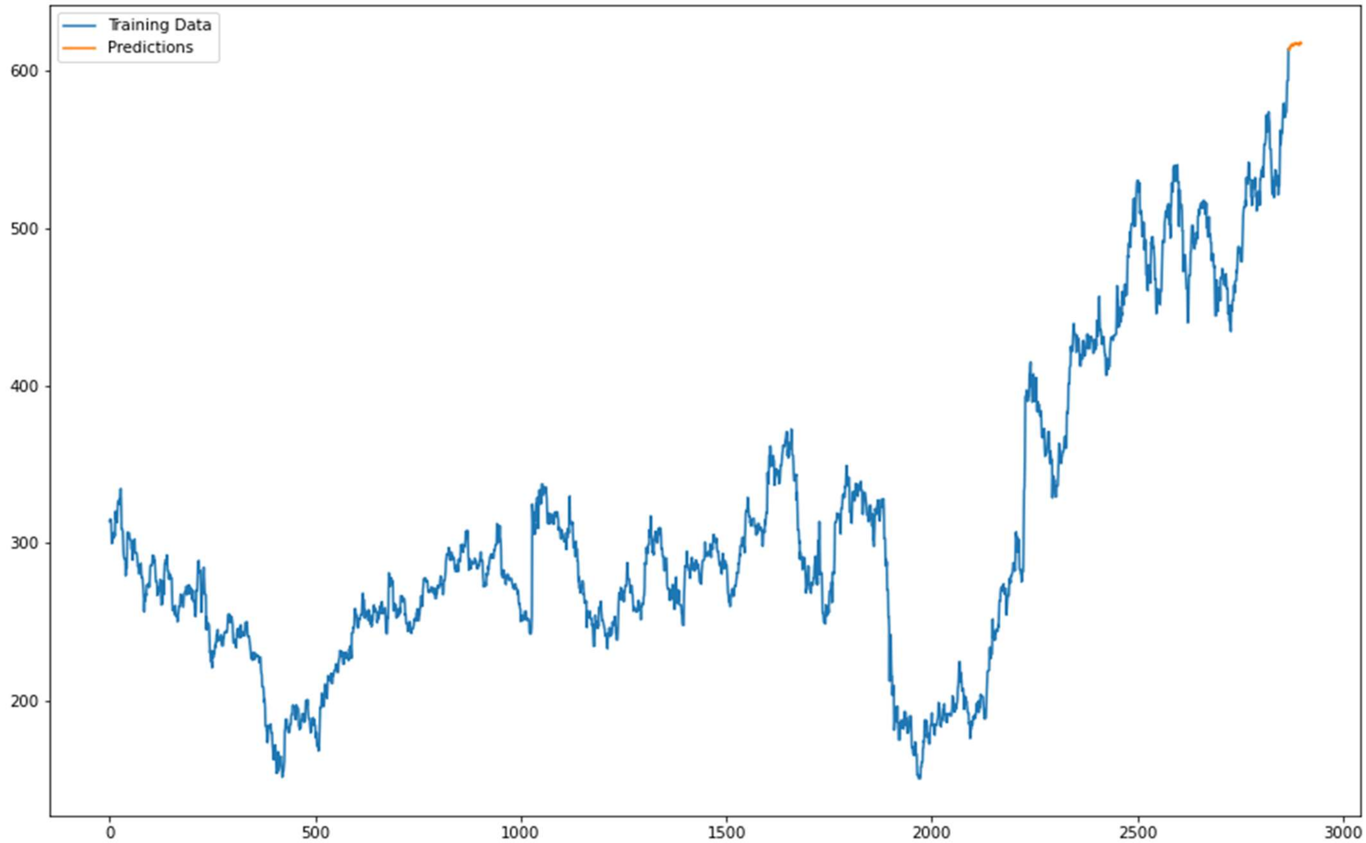
```
2868    613.837972
2869    613.231356
2870    615.062595
2871    615.673296
2872    616.150422
2873    615.985802
2874    615.540617
2875    616.433601
2876    617.053005
2877    616.089526
2878    617.134454
Name: predicted_mean, dtype: float64
```

Predicted values for next 10days with SARIMA model.

plotting the predicted values:

```
sbin_data["Price"].plot(legend=True, label="Training Data", figsize=(15, 10))
predictions.plot(legend=True, label="Predictions")
```

```
<AxesSubplot:>
```



Conclusion: ARIMA is an algorithm used for forecasting Time Series Data. If the data is stationary, we need to use ARIMA, if the data is seasonal, we need to use Seasonal ARIMA (SARIMA).
By using ARIMA(SARIMA) model forecasted next 10days values
The ARIMA algorithm will be a great asset for brokers and investors for investing money in the stock market since it is trained on a vast collection of historical data and has been chosen after being tested on a trial data.
The project demonstrates the machine learning model to predict the stock price with more accuracy as compared to other machine learning models.

LSTM model

LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better.

```
]: ### LSTM are sensitive to the scale of the data. so we apply MinMax scaler

]: sbin_data1=sbin_data.reset_index()['Price']

]: sbin_data1.shape

]: (2868,)

]: from sklearn.preprocessing import MinMaxScaler
   scaler=MinMaxScaler(feature_range=(0,1))
   sbin_data1=scaler.fit_transform(np.array(sbin_data1).reshape(-1,1))

]: print(sbin_data1)

   [[0.35214764]
    [0.35484567]
    [0.35484567]
    ...
    [0.95639974]
    [0.95639974]
    [1.        ]]

]: ##splitting dataset into train and test split
   training_size=int(len(sbin_data1)*0.65)
   test_size=len(sbin_data1)-training_size
   train_data,test_data=sbin_data1[0:training_size,:],sbin_data1[training_size:len(sbin_data1),:1]

]: training_size,test_size

]: (1864, 1004)
```

```
: train_data

: array([[0.35214764],
         [0.35484567],
         [0.35484567],
         ...,
         [0.34567235],
         [0.36930714],
         [0.36628534]])

: import numpy
  # convert an array of values into a dataset matrix
  def create_dataset(dataset, time_step=1):
      dataX, dataY = [], []
      for i in range(len(dataset)-time_step-1):
          a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
          dataX.append(a)
          dataY.append(dataset[i + time_step, 0])
      return numpy.array(dataX), numpy.array(dataY)

: # reshape into X=t,t+1,t+2,t+3 and Y=t+4
  time_step = 300
  X_train, y_train = create_dataset(train_data, time_step)
  X_test, ytest = create_dataset(test_data, time_step)

: print(X_train.shape), print(y_train.shape)

  (1563, 300)
  (1563,)

: (None, None)

: print(X_test.shape), print(ytest.shape)

  (703, 300)
  (703,)

: (None, None)
```

Splitting data set into training and testing, and reshaping the input to be samples, time steps, features required for LSTM.

```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(300,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 300, 50)           10400

 lstm_1 (LSTM)               (None, 300, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

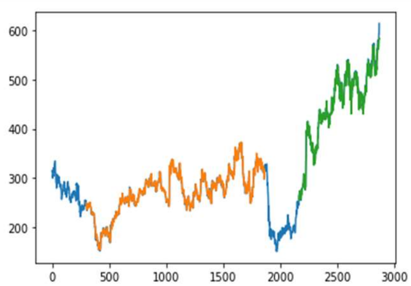
```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
274.87385694175975
```

```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

```
451.35118367292546
```

```
### Plotting
# shift train predictions for plotting
look_back=300
trainPredictPlot = numpy.empty_like(sbin_data1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(sbin_data1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(sbin_data1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(sbin_data1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```




```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=100,batch_size=64,verbose=1)
```

```
25/25 [==============================] - 12s 463ms/step - loss: 1.2999e-04 - val_loss: 3.7645e-04
Epoch 83/100
25/25 [==============================] - 12s 463ms/step - loss: 1.5543e-04 - val_loss: 6.7999e-04
Epoch 84/100
25/25 [==============================] - 12s 462ms/step - loss: 1.2321e-04 - val_loss: 9.1586e-04
Epoch 85/100
25/25 [==============================] - 12s 469ms/step - loss: 1.1834e-04 - val_loss: 0.0013
Epoch 86/100
25/25 [==============================] - 12s 468ms/step - loss: 1.5125e-04 - val_loss: 4.3061e-04
Epoch 87/100
25/25 [==============================] - 12s 473ms/step - loss: 1.2326e-04 - val_loss: 3.5192e-04
Epoch 88/100
25/25 [==============================] - 12s 481ms/step - loss: 1.1895e-04 - val_loss: 8.9335e-04
Epoch 89/100
25/25 [==============================] - 12s 467ms/step - loss: 1.2202e-04 - val_loss: 3.7168e-04
Epoch 90/100
25/25 [==============================] - 11s 462ms/step - loss: 1.2692e-04 - val_loss: 2.9339e-04
Epoch 91/100
25/25 [==============================] - 11s 461ms/step - loss: 1.3699e-04 - val_loss: 2.6570e-04
```

```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.7.0'
```

```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

```
# demonstrate prediction for next 30 days
from numpy import array

lst_output=[]
n_steps=300
i=0
while(i<30):

    if(len(temp_input)>300):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

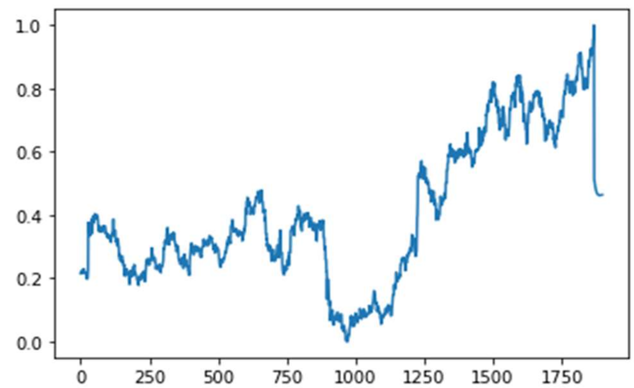
```
0 day input [0.81750486 0.81750486 0.81750486 0.75663717 0.8071444  0.78966113
 0.78156702 0.78663933 0.78663933 0.78663933 0.77919275 0.75016188
 0.75080941 0.69458234 0.71681416 0.71681416 0.71681416 0.71735377
 0.71735377 0.69803583 0.68325059 0.67148716 0.67148716 0.67148716
 0.62475718 0.62475718 0.64936326 0.68605655 0.68961796 0.68961796
 0.68961796 0.7215627  0.7230736  0.73796676 0.75771638 0.75771638
 0.75771638 0.75771638 0.73332614 0.74001727 0.73343406 0.72577164
```

Predicting[close] Price values for next 30 days

Visualizing Forecasted data:

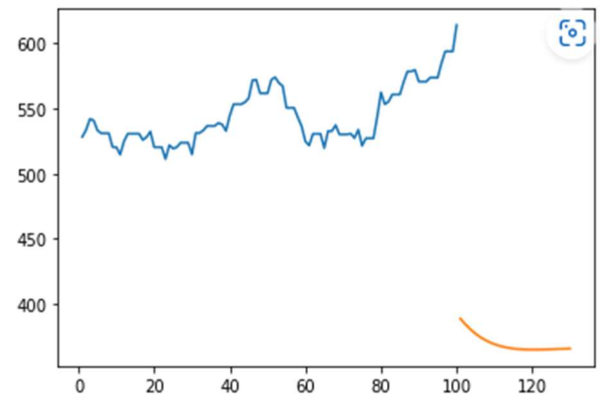
```
df3=sbin_data1.tolist()
df3.extend(lst_output)
plt.plot(df3[1000:])
```

```
[<matplotlib.lines.Line2D at 0x2bb0cb5d760>]
```




```
plt.plot(day_new,scaler.inverse_transform(sbin_data1[2768:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x2bb0c9dd700>]
```



Conclusion:

Forecasted Future 30 days Close(price) by using LSTM model.

Predicting stock market returns is a challenging task due to consistently changing stock values which are dependent on multiple parameters which form complex patterns.

Conclusion

By comparing LSTM and ARIMA(SARIMA) models, ARIMA model predicted good accurate price values, So for SBIN(State Bank of INDIA)(NSE) stocks concluded ARIMA is finalized for SBIN Stocks.

Thank You