Currency Exchange Rate Forecasting using Machine Learning Techniques

(by Yaswanth Phani Kommineni)

Contents

Problem Definition

Given the currency exchange rates from late 20th century till present year, the task is to predict the exchange rates for next 50 days.

Datasets

Dataset – 1:

- Source: https://www.kaggle.com/kianwee/foreign-exchange-rate-1994-2020
- The above data tells us the currency exchange rate (with respect to USD) for 53 currencies during every day between years 1994 – 2020.
- There are only two features (for each of 53 currencies), they are date and exchange rate with respect to USD. It is observed the second feature increases gradually according to the date.

Dataset – 2:

- Source: https://www.kaggle.com/brunotly/foreign-exchange-rates-per dollar-20002019
- The data tells us about the currency exchange rates between 2000 to 2029 for 24 different countries with respect to USD.
- The given dataset has only two features for all the 24 countries. They are the date and the exchange rate with respect to USD. While taking a closer look at the data it is observed that the exchange rate increases with Time.

Dataset – 3:

- SOURCE: https://www.kaggle.com/federalreserve/exchange-rates
- The Federal Reserve's H.10 statistical release provides data on exchange rates between the US dollar, 23 other currencies, and three benchmark indexes. The data extend back to 1971 for several of these.
- The Given Dataset has only 2 features (for 24 currencies including the USA). The features are date and exchange rate with respect to US Dollar. If we observe We can Say that there is increase in exchange rate with respect to Time.

Data Pre-processing

- We have 3 datasets and the pre-processing that we did on those 3 datasets is the same.
- In all the three datasets we converted the DataFrame of combined currencies into array of DataFrames with each element of the array containing the details of exchange rates for individual currency to US dollar.

```
data = []
for x in originaldata.columns[1:len(originaldata.columns)]:
    data.append(originaldata[[originaldata.columns[0],x]])
```

- Dealing with null values:
    - In all the three datasets, we have some null values (NaN) in the data because, on some date, the currency exchange rate for some currency may not be available.

```
print(data[0])
```

```
            Date  Algerian dinar (DZD)
0     3-Jan-1994                   NaN
1     4-Jan-1994                   NaN
2     5-Jan-1994                   NaN
3     6-Jan-1994                   NaN
4     7-Jan-1994                   NaN
...          ...                   ...
6732         NaN                   NaN
6733         NaN                   NaN
6734         NaN                   NaN
6735         NaN                   NaN
6736         NaN                   NaN

[6737 rows x 2 columns]
```

```
originaldata.isnull().sum()
```

```
Time Serie                                      0
AUSTRALIA - AUSTRALIAN DOLLAR/US$             198
EURO AREA - EURO/US$                          198
NEW ZEALAND - NEW ZELAND DOLLAR/US$           198
UNITED KINGDOM - UNITED KINGDOM POUND/US$     198
BRAZIL - REAL/US$                             198
CANADA - CANADIAN DOLLAR/US$                  198
CHINA - YUAN/US$                              197
HONG KONG - HONG KONG DOLLAR/US$              198
INDIA - INDIAN RUPEE/US$                      199
KOREA - WON/US$                               198
MEXICO - MEXICAN PESO/US$                     198
SOUTH AFRICA - RAND/US$                       198
SINGAPORE - SINGAPORE DOLLAR/US$              198
DENMARK - DANISH KRONE/US$                    198
JAPAN - YEN/US$                               198
MALAYSIA - RINGGIT/US$                        198
NORWAY - NORWEGIAN KRONE/US$                  198
SWEDEN - KRONA/US$                            198
SRI LANKA - SRI LANKAN RUPEE/US$              198
SWITZERLAND - FRANC/US$                       198
TAIWAN - NEW TAIWAN DOLLAR/US$                201
THAILAND - BAHT/US$                           198
dtype: int64
```

```
originaldata.isnull().sum()
```

```
Series Description                                                    0
SPOT EXCHANGE RATE - EURO AREA                                     7305
UNITED KINGDOM -- SPOT EXCHANGE RATE, US$/POUND (1/RXI_N.B.UK)        0
SPOT EXCHANGE RATE - BRAZIL                                        6260
CHINA -- SPOT EXCHANGE RATE, YUAN/US$ P.R.                         2609
DENMARK -- SPOT EXCHANGE RATE, KRONER/US$                             0
INDIA -- SPOT EXCHANGE RATE, RUPEES/US$                             521
JAPAN -- SPOT EXCHANGE RATE, YEN/US$                                  0
KOREA -- SPOT EXCHANGE RATE, WON/US$                               2680
MALAYSIA -- SPOT EXCHANGE RATE, RINGGIT/US$                           0
MEXICO -- SPOT EXCHANGE RATE, PESOS/US$                            5960
NORWAY -- SPOT EXCHANGE RATE, KRONER/US$                              0
SWEDEN -- SPOT EXCHANGE RATE, KRONOR/US$                              0
SOUTH AFRICA -- SPOT EXCHANGE RATE, RAND/$US                       2347
SINGAPORE -- SPOT EXCHANGE RATE, SINGAPORE $/US$                   2609
SWITZERLAND -- SPOT EXCHANGE RATE, FRANCS/US$                         0
TAIWAN -- SPOT EXCHANGE RATE, NT$/US$                              3325
THAILAND -- SPOT EXCHANGE RATE, BAHT/US$                           2609
SPOT EXCHANGE RATE - VENEZUELA                                     6260
Nominal Broad Dollar Index                                         6262
Nominal Major Currencies Dollar Index                               521
Nominal Other Important Trading Partners Dollar Index              6262
AUSTRALIA -- SPOT EXCHANGE RATE US$/AU$ (RECIPROCAL OF RXI_N.B.AL)    0
NEW ZEALAND -- SPOT EXCHANGE RATE, US$/NZ$ RECIPROCAL OF RXI_N.B.NZ   0
CANADA -- SPOT EXCHANGE RATE, CANADIAN $/US$                          0
HONG KONG -- SPOT EXCHANGE RATE, HK$/US$                           2609
SRI LANKA -- SPOT EXCHANGE RATE, RUPEES/US$                         521
dtype: int64
```

- In the first dataset, we iterated through all the data values and removed null values. This was done after forming array of

DataFrames as mentioned earlier.

```
for i in range(len(data)):
    columns = data[i].columns
    removalRows = []
    for j in range(data[i].shape[0]):
        if(math.isnan(data[i][columns[1]].values[j])):
            removalRows.append(j)
    temp = data[i].drop(removalRows,axis = 0)
    data[i] = deepcopy(temp)
```

```
print(data[0])
```

```
             Date  Algerian dinar (DZD)
4253  14-May-2010                74.9996
4259  24-May-2010                75.2979
4260  25-May-2010                75.6136
4261  26-May-2010                75.4628
4262  27-May-2010                75.5750
...           ...                    ...
6695  24-Feb-2020               120.9095
6696  25-Feb-2020               120.7883
6697  26-Feb-2020               120.7145
6698  27-Feb-2020               120.5406
6699  28-Feb-2020               120.0978

[2287 rows x 2 columns]
```

- In second and the third dataset, we used inbuilt function to remove the rows that contain null values. This was done before converting the main data into array of DataFrames unlike what we did on the first dataset.

```
originaldata.interpolate(inplace = True)
originaldata.isnull().sum()
```

```
Time Serie                                  0
AUSTRALIA - AUSTRALIAN DOLLAR/US$           0
EURO AREA - EURO/US$                        0
NEW ZEALAND - NEW ZELAND DOLLAR/US$         0
UNITED KINGDOM - UNITED KINGDOM POUND/US$   0
BRAZIL - REAL/US$                           0
CANADA - CANADIAN DOLLAR/US$                0
CHINA - YUAN/US$                            0
HONG KONG - HONG KONG DOLLAR/US$            0
INDIA - INDIAN RUPEE/US$                    0
KOREA - WON/US$                             0
MEXICO - MEXICAN PESO/US$                   0
SOUTH AFRICA - RAND/US$                     0
SINGAPORE - SINGAPORE DOLLAR/US$            0
DENMARK - DANISH KRONE/US$                  0
JAPAN - YEN/US$                             0
MALAYSIA - RINGGIT/US$                      0
NORWAY - NORWEGIAN KRONE/US$                0
SWEDEN - KRONA/US$                          0
SRI LANKA - SRI LANKAN RUPEE/US$            0
SWITZERLAND - FRANC/US$                     0
TAIWAN - NEW TAIWAN DOLLAR/US$              0
THAILAND - BAHT/US$                         0
dtype: int64
```

- In our dataset the dates are of ‘str’ datatype. But, we need to convert them to numpy.datetime64 to make them usable. On all the three datasets, this was done similarly.

```
print(type(data[0][data[0].columns[0]].values[0]))
```

```
<class 'str'>
```

```
pd.options.mode.chained_assignment = None
for i in range(len(data)):
    col = data[i].columns[0]
    data[i][col] = pd.to_datetime(data[i][col].values)
```

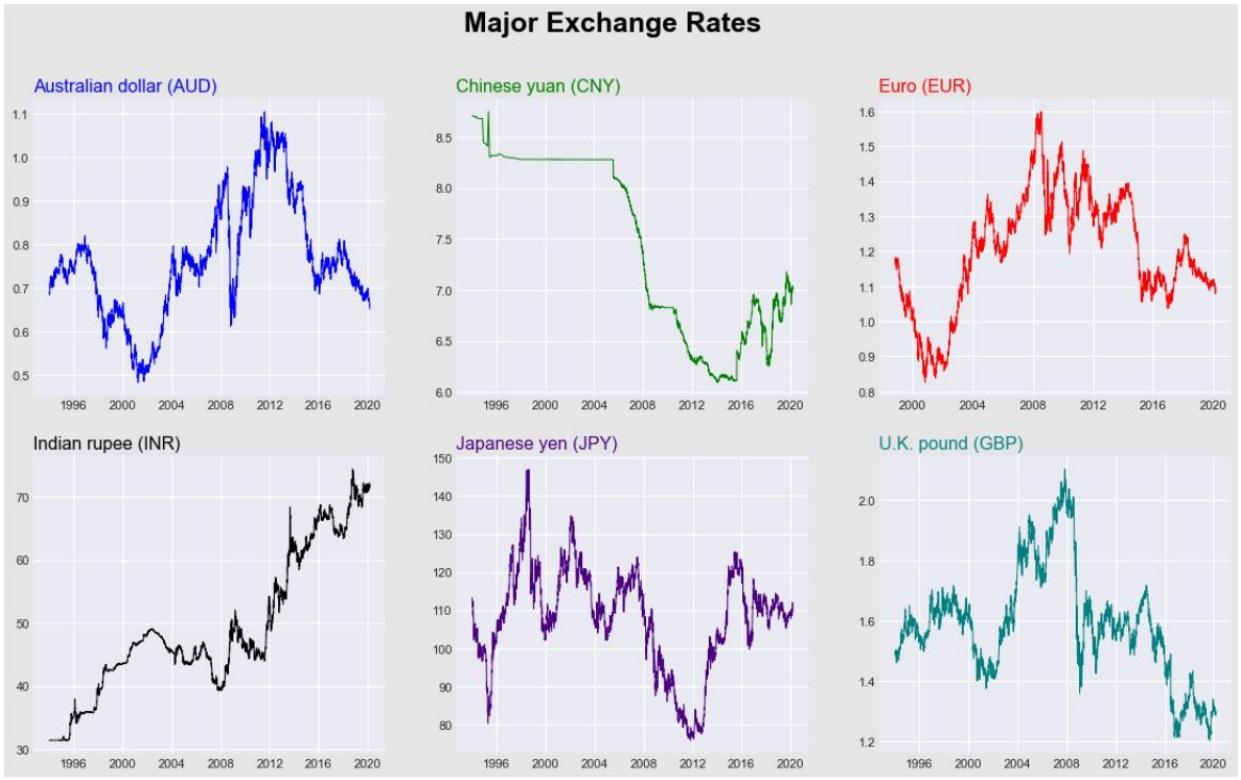
```
print(type(data[0]['Time Serie'].values[0]))
```

```
<class 'numpy.datetime64'>
```

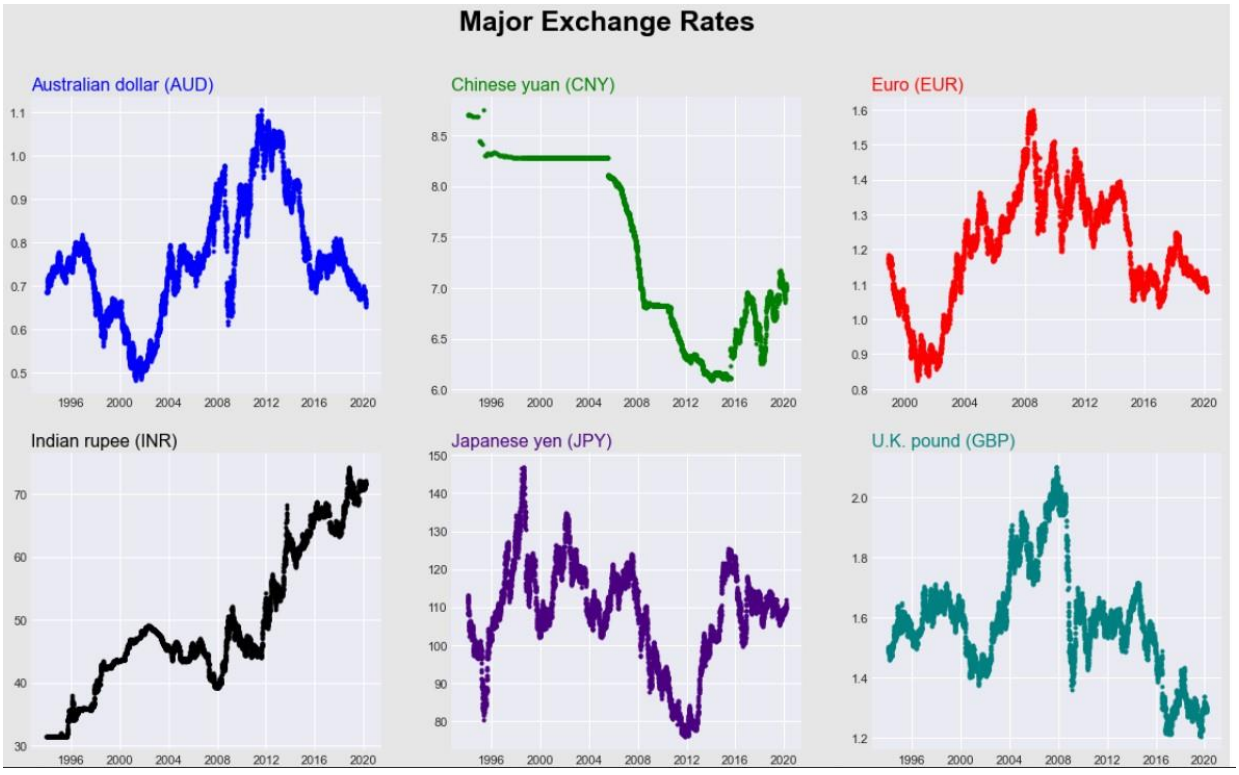
```
for i in range(len(data)):
    data[i].set_index('Time Serie',inplace = True)
```

Data Visualization

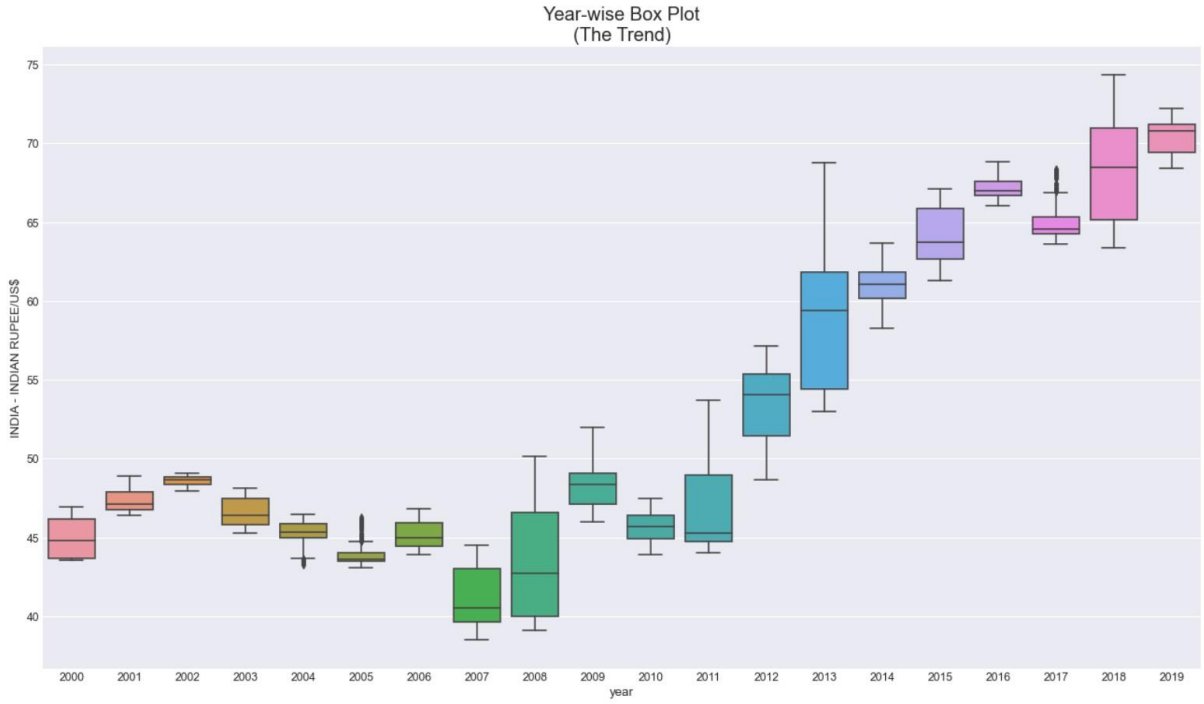
- Line Chart



- Scatter Plot

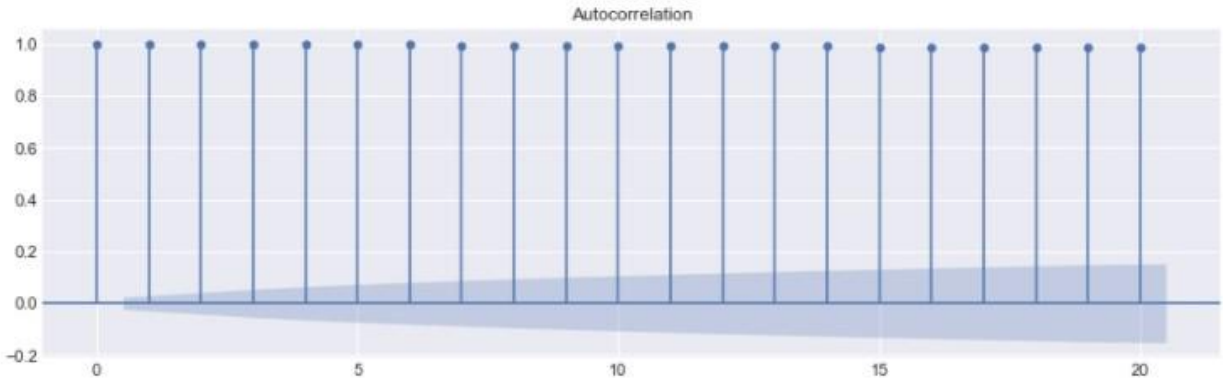


- Box Plot

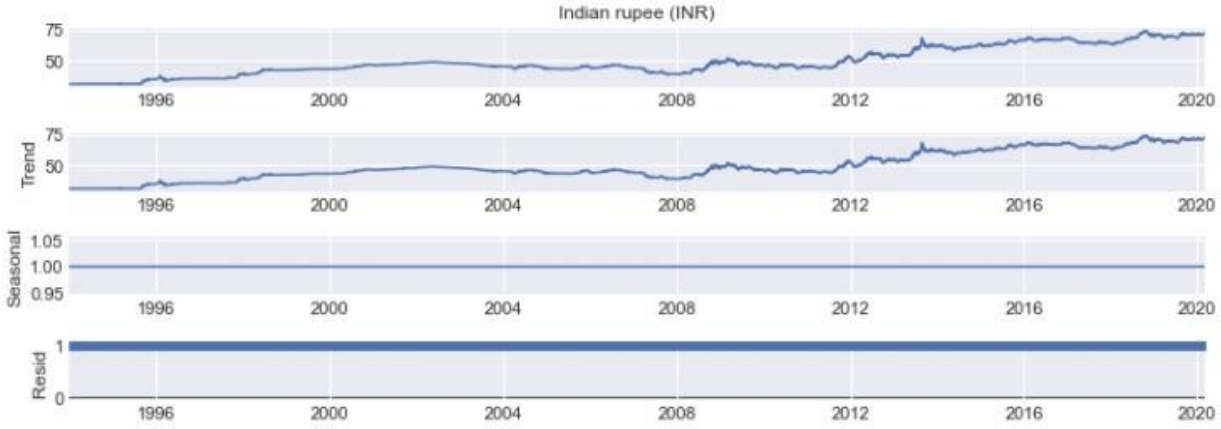

Year-wise Box Plot
(The Trend)
INDIA - INDIAN RUPEE/US$
75
70
65
60
55
50
45
40
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
year

Data Summarization

- Auto-correlation: The auto-correlation function has a slow decay, which indicates that the future values and its past values are highly correlated.



- Seasonal Decomposition: We can see that the residual plot shows zero. The decomposition was not able to separate the noise that we added from the linear trend.



- Our dataset is time series dataset. So, we only have one attribute which is date.

Python Packages

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import math
from copy import deepcopy
import seaborn as sns
import xgboost as xgb
from xgboost import XGBRegressor
from bayes_opt import BayesianOptimization
from sklearn.preprocessing import MinMaxScaler
import statsmodels.api as sm
from statsmodels.tsa.statespace import sarimax
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import torch
import torch.nn as nn
from datetime import datetime
%matplotlib inline
```

pandas – To make use of DataFrames that are provided by pandas

numpy – To make use of inbuilt computational operations and numpy.datetime64 datatype which is provided by numpy.

matplotlib.pyplot – to plot useful graphs for data visualization and summarization.

deepcopy – to create copies of some data.

seaborn – To make graphs better as we can customize our graphs.

xgboost – Provides inbuilt function for gradient boosting algorithm.

bayes_opt – makes parameter tuning much easier.

MinMaxScaler – converts numerical values to range [-1,1] relative to the minimum and the maximum values.

statsmodes.api – A convenience interface for specifying models using formula strings and DataFrames.

sarimax – provides inbuilt ARIMA algorithm.

LSTM – to make use of inbuilt algorithm LSTM algorithm.

mean_squared_error – inbuilt function to calculate MSE.

seasonal_decompose – to plot seasonal decompose graph.

r2_score – inbuilt function to calculate R2 score.

plot_pacf,plot_acf – to plot auto-correlation graph.

torch, torch.nn – to make use of PyTorch deep neural networks (used in LSTM).

%matplotlib inline – sets the backend of matplotlib to the 'inline' backend, With this backend, the output of plotting commands is displayed inline within frontends like the Jupyter notebook, directly below the code cell that produced it.

Machine Learning Algorithms

In all three datasets, we have split the data into test data and training data with test data being the exchange rate values being the last 50 days.

```
test_size = 50
x_train, x_test = x[:-test_size], x[-test_size:]
y_train, y_test = y[:-test_size], y[-test_size:]
```

Note:

- In sequential algorithms, the test data should be the last instances. Because, if we know the past and the future values, it will be easy to predict the present values which lead to an improper measure of the performance. Hence, no k-fold validation required.
- We have not used validation split because there is some complicated issue. Let us say I have 100 sequential data samples of exchange rate for some currency. If I split the data into train (60), validation (20), and test (20), I should train the model on the first train data (60) samples and change the parameters according to the accuracy of validation data. Then, if we apply that on test data, the results will not be accurate. Because, the last few days of test data is validation data and we did not have this crucial information as the results highly depend on this information.

Polynomial Regression

- Polynomial regression is almost same as linear regression. In linear regression, we use multiple features whereas in polynomial regression, we use only single feature and convert it to multiple features by squaring, cubing etc.

  Linear Regression: y = a*x + b*y + c
  Polynomial Regression: y = a + b*x + c*(x^2) + d*(c^3) ….

  Here, y is the target value. We just find the optimal values of a,b,c .. (weights) by using gradient descent algorithm.

- By series of parameter tunings, we arrived at the conclusion that the polynomial with degree 4 is the best fit for our data.

  degree = 1

  degree = 2

  degree = 3

degree = 4

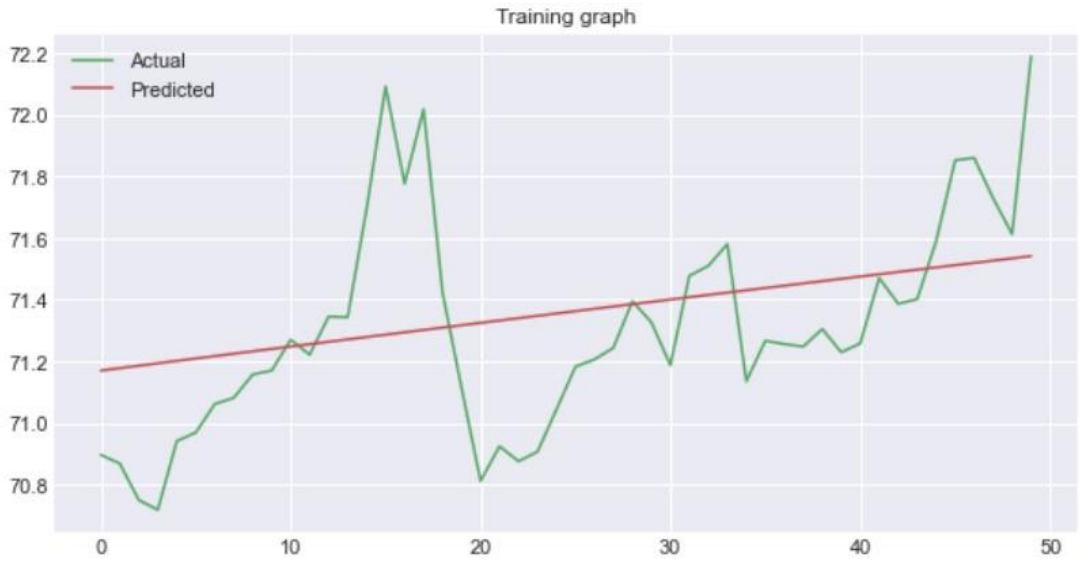
degree = 5

degree = 6

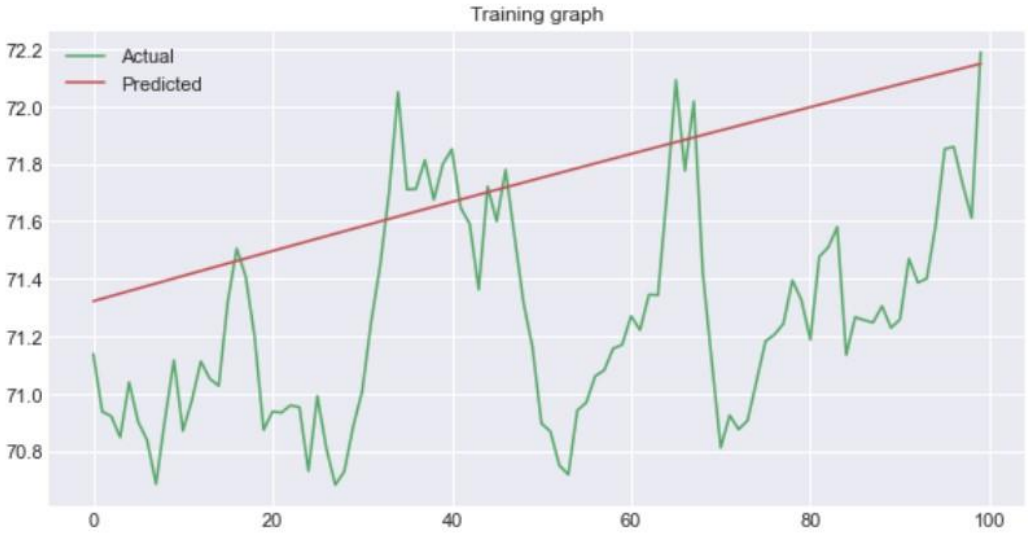
degree = 7

- Prediction graphs on test data
  test size = 20

  test size = 50



test size = 100



test size = 500

- Root Mean Square Error (for 50 test size): 0.3018312293288945
- R2- Score(for 50 test size): 0.21473658422994912

ARIMA

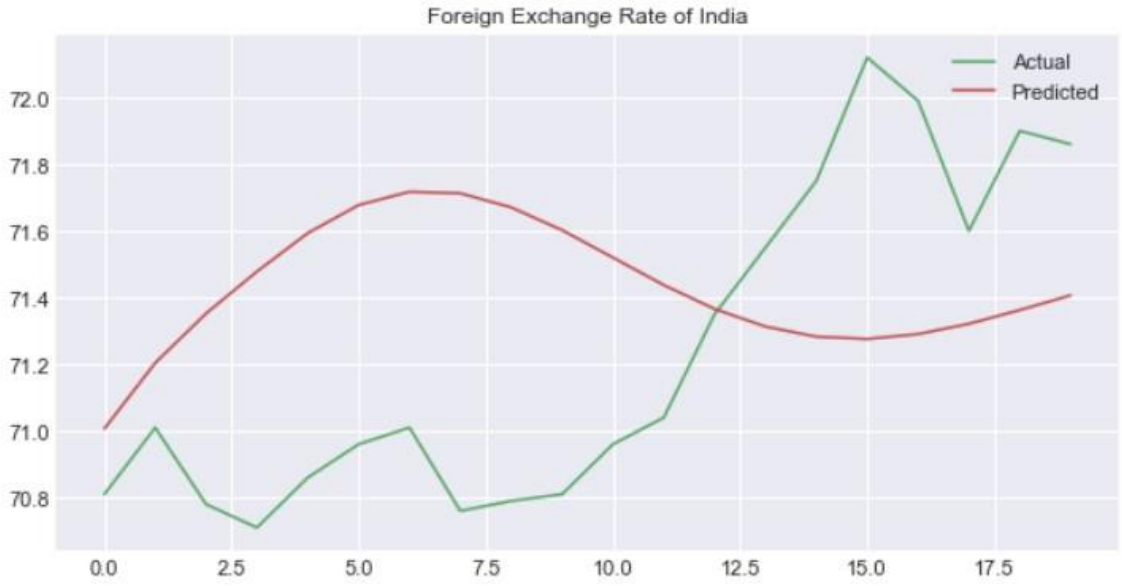
- ARIMA stands for Auto Regressive Integrated Moving Average. The main objective of the ARIMA model is for forecasting (predicting future values of the Time Series). The model is referred to as ARIMA (p, d, q), where p, d and q are non-negative numerical values. ARIMA models are defined for stationary Time Series.
- Prediction graph

- Root mean squared error: 0.306372523975768

- R2- score: 0.19092897441906298

LSTM

- Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more.
- Prediction graph



- Root mean squared error: 0.6073244704926808
- R2- score: -0.6783213849961363

XGBoost

- XGBoosting or extreme gradient boosting is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data Such as images, text, etc. It is extremely popular for its speed in tree-based machine learning algorithms.

- Prediction graph

- Root mean squared error: 0.19266318256774767
- R2- score: 0.5763280597098182

Best Algorithm

Among these four algorithms, XGBoost has less RMSE and a high R2-score which means it outperformed the other three algorithms.

The below graph shows the RMSE values for all four algorithms when applied on different test sizes. We can see that both the implemented Polynomial regression and ARIMA almost have the same accuracy because both come from the same regression family. If we look at the overall performance, XGBoost has the lower RMSE. Hence, we can say that XGBoost performs better than the other three algorithms.