Endsem part 2
Time series analysis of stock price data

Anik Bhowmick
Inter Disciplinary Dual Degree Data-Science
Indian Institute of Technology Madras
ae20b102@smail.iitm.ac.in

***Abstract*—The objective of this paper, as a part of the final exam, is to explore various relevant statistical techniques for time series analysis. The datasets given for this purpose are the financial data/ stock prices for various companies and banks like Cognizant, Infosys, HDFC, SBI, etc. Because of the dynamic nature of the stock market, it often poses a challenging task to predict future stock prices. Machine learning has emerged as a powerful tool to mitigate this issue. So, to overcome this challenge, this paper will discuss the applicability of machine-learning techniques, including some techniques not taught in class. This paper will also uncover the key difference between conventional regression techniques and time-series predictions in addition to numerous visualization techniques.**

***Index Terms*—time series, python, visualization, predictive modelling, financial data.**

I. Introduction

In our daily lives, we frequently encounter a significant amount of temporal data characterized by an inherent order that other types of data lack. This intrinsic ordering is crucial for analysis, adding complexity to the process. While various techniques are available for handling such data, machine learning has become increasingly popular in recent years for temporal data analysis, particularly in the context of time series data.

Financial markets provide a well-known example of temporal data, known for their complexity and dynamic nature, resulting in the generation of substantial data. Extracting valuable insights from financial data is essential, and this is often manifested in the form of stock market prices and currency exchange rates.

The focus here is on leveraging machine learning techniques to analyze data from financial markets, specifically employing time series analysis methods. Although no specific objective is outlined, the aim is to identify stocks that offer the best return on investment, a key consideration for investors. Additionally, the exploration includes building a model for predicting stock prices.

II. Time series and models

A time series is a sequential collection of data points measured or recorded over successive and equally spaced intervals of time. Each data point in a time series corresponds to a specific moment in time, forming a chronological sequence. Time series analysis is a branch of statistics and data analysis that focuses on understanding the patterns, trends, and behaviours inherent in time-ordered data. Common applications of time series analysis include forecasting future values, identifying underlying patterns, and capturing seasonality or cyclicality in the data. Time series data is prevalent in various fields, such as finance, economics, meteorology, and signal processing, where the temporal aspect is a critical factor in understanding and making predictions.

A. Terminologies

1) *Time Point or Timestamp*: An individual moment in time corresponding to a specific data observation.
2) *Temporal Interval*: The time duration between two consecutive time points in a time series.
3) *Temporal Resolution*: The frequency or granularity of observations in a time series, indicating how often data is collected.
4) *Trend*: A long-term systematic pattern or tendency in the time series data that indicates a general direction of movement.
5) *Seasonality*: Regular, repetitive fluctuations or patterns in the data that occur at fixed intervals.
6) *Cyclic Patterns*: Repeating, but not necessarily fixed, patterns that occur over longer time spans than seasonality.
7) *Stationarity*: A desirable property of time series data where statistical properties remain constant over time. In fact, all the models of time series work on this principle; if in case the data is non-stationary, it is very important to convert the data to stationary and then use the model. But ARIMA has a feature that automatically makes the data stationary and then performs the training.
8) *Autocorrelation*: The correlation of a time series with a lagged version of itself, indicating the degree of similarity between observations at different time points.
9) *Lag*: The time interval between two observations used in autocorrelation analysis.
10) *Moving Average*: A statistical calculation that smoothens fluctuations in data by averaging values over a specified window or period. The typical moving average used in stock markets is 50 days. This moving average helps in

understanding the trend of the time series without being affected by noise.

11) *Exponential Smoothing*: A time series forecasting method that assigns exponentially decreasing weights to past observations.
12) *ARIMA (AutoRegressive Integrated Moving Average)*: A popular time series forecasting model that combines autoregression, differencing, and moving averages.
13) *Seasonal Decomposition of Time Series (STL)*: A technique to decompose a time series into its trend, seasonality, and residual components.
14) *Forecasting*: The process of predicting future values of a time series based on historical data and patterns.

B. Trend, Seasonality, Noise and Level

A given time series is comprised of three systematic components: level, trend, and seasonality, along with one non-systematic component known as noise. There is no series free from level and noise. The trend and seasonality need not be present necessarily. These components are present in the true data in the form of addition or multiplication.

$$y(t) = \text{Level} + \text{Trend} + \text{Seasonality} + \text{Noise}$$

$$y(t) = \text{Level} \times \text{Trend} \times \text{Seasonality} \times \text{Noise}$$

C. Moving-Average, Auto Correlation, Partial Correlation

In financial applications, a simple moving average (SMA) is the unweighted mean of the previous t data points. These t data points can be stock closing prices. Say we are using the past t days data to calculate this average. Mathematically,

$$SMA_t = \frac{X_1 + X_2 + \ldots + X_t}{t}$$

This is a t-day estimated moving average. This can be used as an estimate for the t+1 th day prediction. However, the error introduced by the MA is very high because it does not look at the trend or seasonality of the data. Instead, it should be looked at as an indicator of the trend free from noise. This process is also called as smoothing.

In autoregression, the data at the current timestamp is assumed to depend on the past data point/ points. If depends on one single past data point, it is called autoregression of order 1. It's given mathematically as,

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t$$

Autoregression of order 2 is given as,

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \epsilon_t$$

Similarly, it can be extended for order k. this k is called Lag. But what order the real data set follows is a very difficult task to know beforehand. Auto-ARIMA has a feature to decide what order the true distribution of the data for autoregression is following.

Autocorrelation is a method to check the goodness of autoregressive model assumption. It basically finds the correlation between the data at a particular time stamp and data that is past by some lag. Mathematically,

$$\text{Autocorrelation at lag } k = \frac{\sum_{t=k+1}^{T}(X_t - \bar{X})(X_{t-k} - \bar{X})}{\sum_{t=1}^{T}(X_t - \bar{X})^2}$$

where:

- X_t is the value of the time series at time t,
- $\bar{X}$ is the mean of the time series,
- T is the total number of observations in the time series, and
- k is the lag for which autocorrelation is being calculated.

The partial autocorrelation at lag k for a time series is denoted by $\phi_{k,k}$ and is calculated using the Yule-Walker equations. The formula for the partial autocorrelation function (PACF) is recursive and is expressed as follows:

For $k = 1$:

$$\phi_{1,1} = \rho_1$$

For $k > 1$:

$$\phi_{k,k} = \frac{\rho_k - \sum_{i=1}^{k-1} \phi_{k-1,i} \cdot \rho_{k-i}}{1 - \sum_{i=1}^{k-1} \phi_{k-1,i} \cdot \rho_i}$$

where:

- ρ_k is the autocorrelation at lag k,
- $\phi_{k,i}$ is the partial autocorrelation at lag k with i lags excluded.

The PACF helps identify the direct relationship between observations at different lags without taking the correlation with the intermediate lags. It is particularly useful in determining the appropriate order for an autoregressive model. In some sense, it measures a direct correlation between two time-step data.

D. ARIMA, LSTM

An earlier version of this model was ARMA, which worked well for stationary time series. In this model, it was essential to perform differencing beforehand manually to eliminate seasonality and trends from the data. Next came the ARIMA.

The ARIMA (AutoRegressive Integrated Moving Average) model is a time series forecasting method that combines three main components: autoregressive (**AR**), differencing (**I**), and moving average (**MA**).

1. **Autoregressive Component (AR):** The AR component models the relationship between the current observation and its past observations using autoregressive coefficients $\phi_1, \phi_2, \ldots, \phi_p$:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \varepsilon_t$$

2. **Integrated Component (I):** The integrated component involves differencing the time series to make it stationary. The differencing order is denoted by d, and it is applied as:

$$Y_t = X_t - X_{t-d}$$

3. **Moving Average Component (MA):** The MA component models the relationship between the current observation and past forecast errors (ε) using moving average coefficients $\theta_1, \theta_2, \ldots, \theta_q$:

$$X_t = \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

The ARIMA model combines these components into the formula:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \theta_1 \varepsilon_{t-1}$$
$$\theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

The suitable values for p, d, and q are determined through model diagnostics, autocorrelation, partial autocorrelation analysis, and information criteria like AIC or BIC.

In many cases, it is found that ARIMA prediction is not up to the mark. In such cases, we can take advantage of the Recurrent Neural networks for sequential modelling. In such cases, LSTM comes in very handy. But we won't go into too much regarding forecasting and, hence, details of LSTMs. As the name suggests, long Short-Term Memory (LSTM) is designed for sequential data processing and is celebrated for its proficiency in capturing long-term dependencies. It employs memory cells and gating mechanisms—forget, input, and output gates—to regulate information flow. LSTMs excel in tasks such as time series prediction, natural language processing, and speech recognition, where understanding and remembering patterns over extended sequences are crucial. The gating mechanism empowers LSTMs to selectively update and retrieve information from memory cells, overcoming challenges like the vanishing gradient problem. LSTMs are trained using backpropagation through time (BPTT), with hyperparameters fine-tuned for optimal performance in specific applications.

Basic structure:

The output of LSTM at any time step depends on 3 inputs primarily.

- Cell-state, which maintains a long-term memory of the network.
- Output of the previous time step is also called a hidden state.
- Input of the current time step.

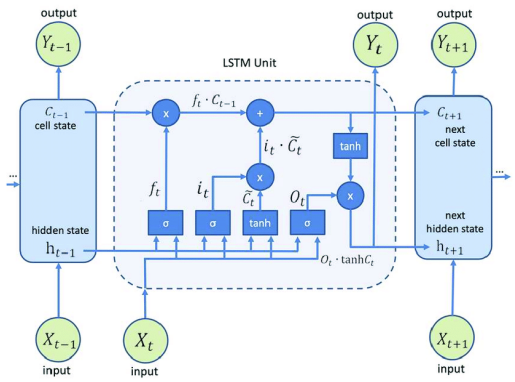


Fig. 1. LSTM cell

III. The Data

There are a total of 7 datasets given, 6 of which are stock prices of HDFC, SBI, ICICI banks, Infosys, HCL and Cognizant company. The last dataset is a USD-INR currency exchange dataset. The range of dates in this dataset is more or less from 2019 to 2021. Each of the datasets has 'Date', 'Open', 'Close', 'Volume', 'High', and 'Low' columns, as usual in any stock dataset. The Closing price is the most essential column for any person who is willing to invest in the stock.

A. Some terminologies related to stock data

- **Open Price:** The stock's price on a given day or time period at the beginning.
- **High Price:** The highest price reached by the stock during the same day or time period.
- **Low Price:** The lowest price reached by the stock during the same day or time period.
- **Close Price:** The price of the stock on a given day or time period at the end.
- **Adjusted Close Price:** The closing price is adjusted for factors such as dividends, stock splits, or other corporate actions. This is often used to account for stock price changes due to events unrelated to its actual performance.

IV. Exploratory Data Analysis

A. Missing Value imputation

Some of the stock data had two missing entries. One was on 27-10-2019, and the other one on 14-11-2020. But when we checked the corresponding days of the week, they were found to be Saturday and Sunday. The stock market operates from Monday to Friday. So, these two data points will have no meaning if they are imputed by traditional techniques such as forward or backward filling. So, we dropped these two rows. For the currency exchange dataset, we also dropped the 21 missing entries.

B. Plots for insight

With these many datasets, we decided to perform a comparative analysis as to which company/ bank will be safer to invest in stock purchases. Below are the individual line plots with moving averages.

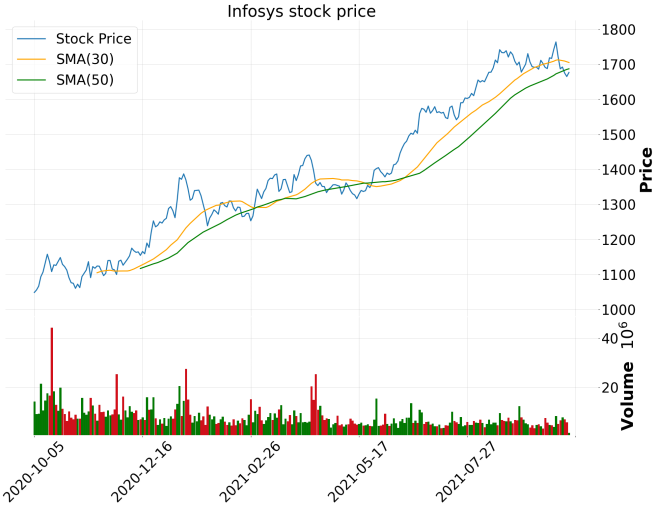


Fig. 2. Infosys closing price with moving average

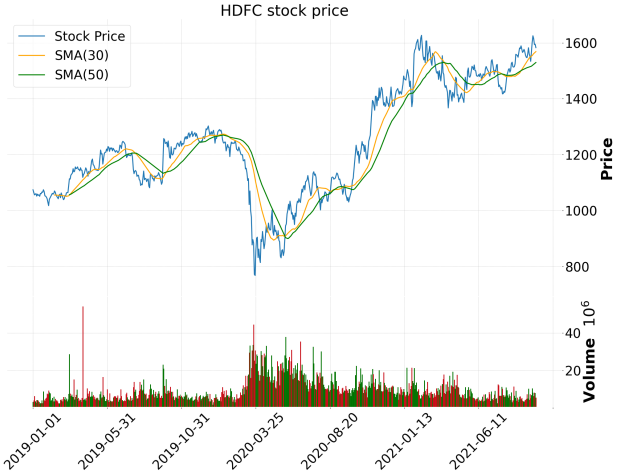


Fig. 5. HDFC closing price with moving average

Fig. 3. HCL closing price with moving average

Fig. 6. SBI closing price with moving average

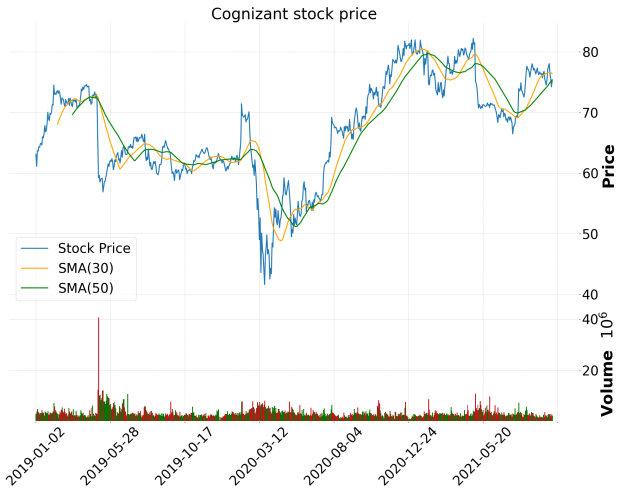


Fig. 4. Cognizant closing price with moving average

Fig. 7. SBI closing price with moving average

The common characteristic of all these plots is that there is a

sharp dip in the stock closing price around 2020. Our intuition tells us that it's simply because of the Pandemic Covid19.

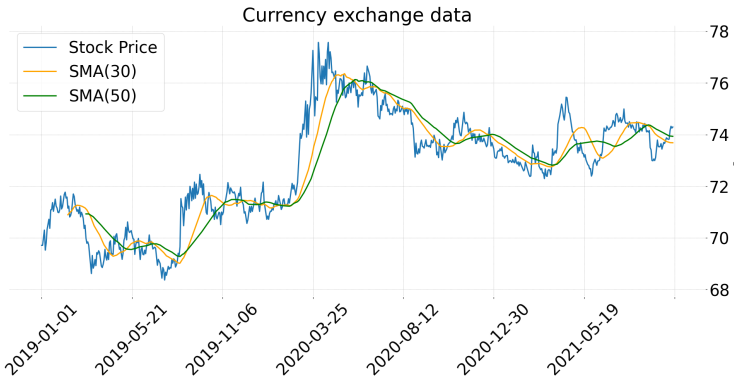


Fig. 8. USD to INR exchange rate

The exchange rate shows an overall increase with time. This is obviously true. In past years, we have seen the exchange rate go up. This currency exchange rate will be used while calculating the daily return rate, and this will help us perform the necessary calculation in the dollar price. Further, we know that currency exchange rates depend on various factors such as interest rates, inflation, market expectations, etc. The closing price of a stock does not consider these effects. So, by conversion to USD, we can account for these effects in the Daily return value. The formula for Daily return is given as follows:

$$\text{DR} = \frac{\text{Today's Closing Price} - \text{Yesterday's Closing Price}}{\text{Yesterday's Closing Price}}$$

Here, DR denotes the daily return. the currency values will be taken in USD. To do so, we have to divide that day's closing price by the closing value of the currency exchange rate of that same day.

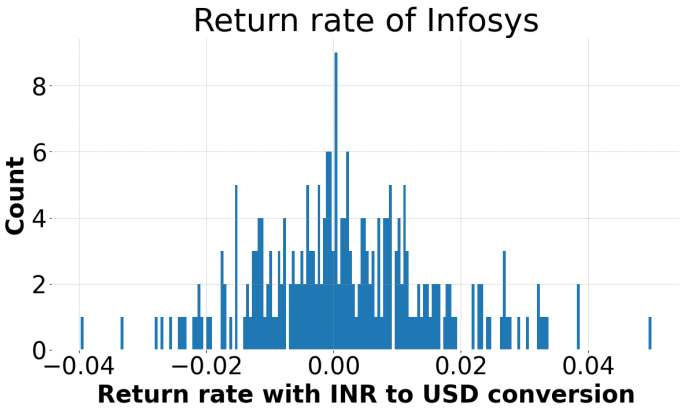


Fig. 9. Daily return of Infosys

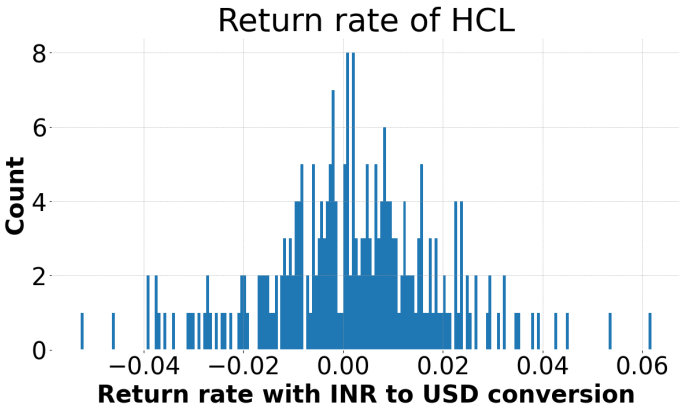


Fig. 10. Daily return of HCL

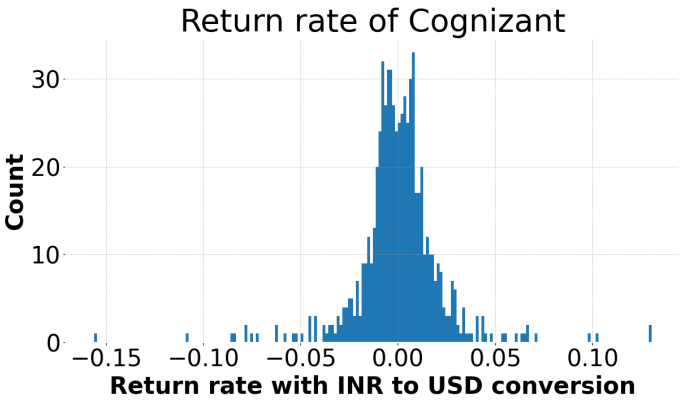


Fig. 11. Daily return of Cognizant

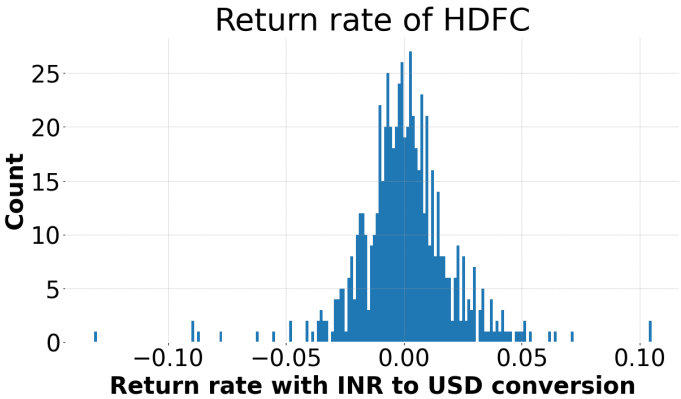


Fig. 12. Daily return of HDFC

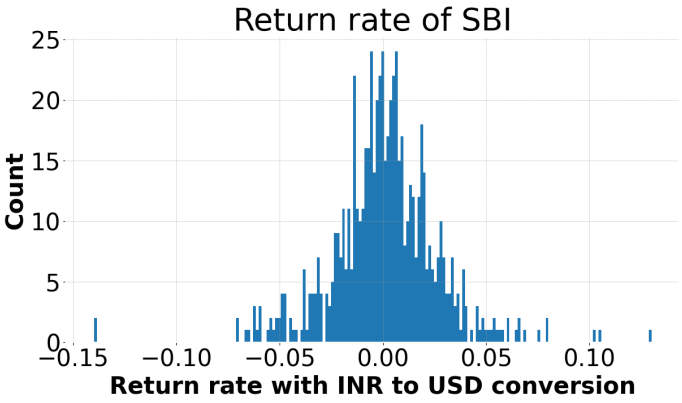


Fig. 13. Daily return of SBI

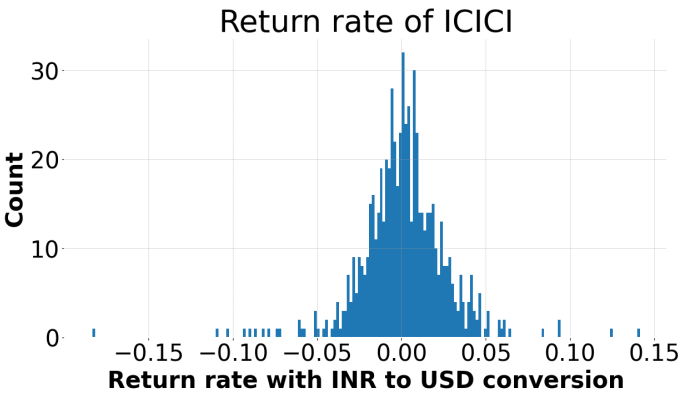


Fig. 14. Daily return of ICICI

The X-axis represents the spread/ fluctuation of the daily return. This spread is the least for Infosys. This is very useful for investors as it directly hints towards the risk associated with investment. The less the spread, the lesser the risk. Below is a plot indicating the risk associated with investment in each organization. The standard deviation of daily return calculates this risk.

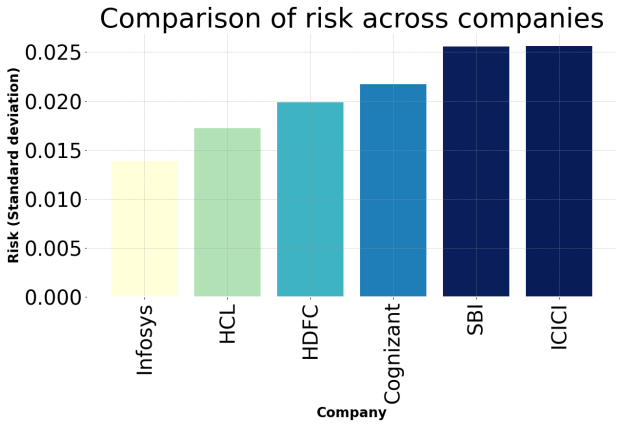


Fig. 15. Risk associated with each investment

Clearly, the risk is the least for Infosys.

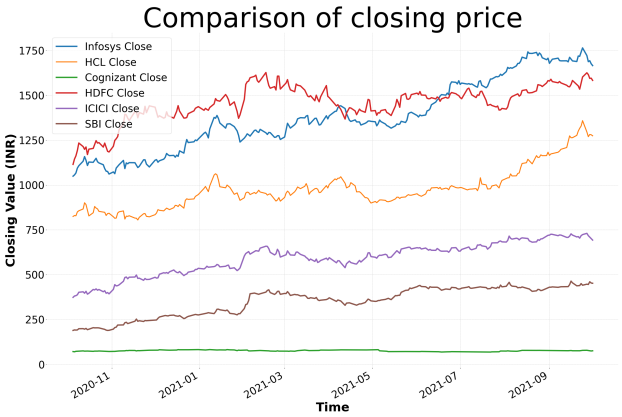


Fig. 16. Combined close price

Also, from the above comparison, Infosys has a higher closing price towards the end of the time series. So, Investors are more likely to invest in Infosys because of its lower risk and higher closing price.

1) Correlation among the closing prices: Two features are correlated if they have an absolute correlation coefficient close to 1. They are uncorrelated if their correlation coefficient is closer to 0. The Pearson's correlation coefficient is given as follows:

$$r_{xy} = \frac{\sum_i x_i y_i - n\bar{x}\bar{y}}{\sqrt{\sum_i x_i^2 - n\bar{x}^2}\ \sqrt{\sum_i y_i^2 - n\bar{y}^2}}.$$

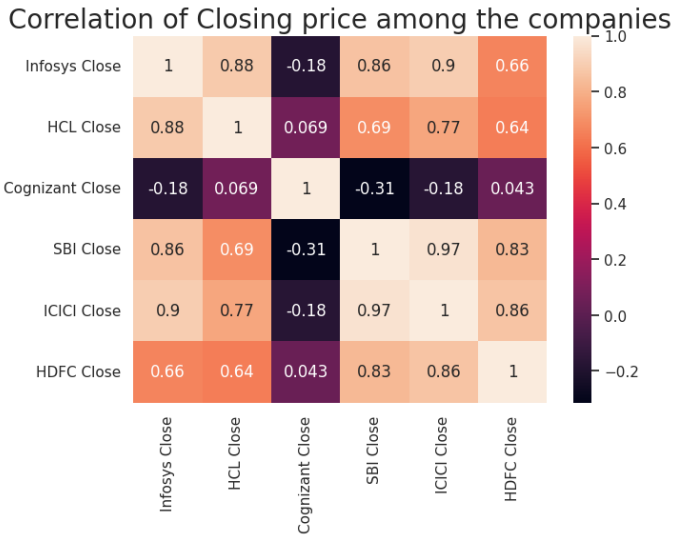


Fig. 17. Correlation heatmap before among the closing prices of the organizations

Here, we can find there is a significant correlation between closing prices among organizations of the same kind. I.e. SBI is highly correlated to HDFC and ICICI. ICICI and HDFC, too, have a high correlation among them. The commonality between these three is all of them are banks. The same is true for the 2 companies, Infosys and HCL. Cognizant has an exceptionally low and negative correlation with the other

two companies. The reason why it is so is not fully clear to us. It may happen that this company operates completely independently from the other two.

C. Autocorrelation analysis

Because Infosys emerged as the best company for investment, we will conduct a predictive and forecasting analysis on Infosys closing price only. The simplest reason is that investors will be more interested in knowing the future stock prediction of a company they are willing to invest in. So, the subsequent part of this paper will discuss necessary preprocessing and model training only on the Infosys dataset.

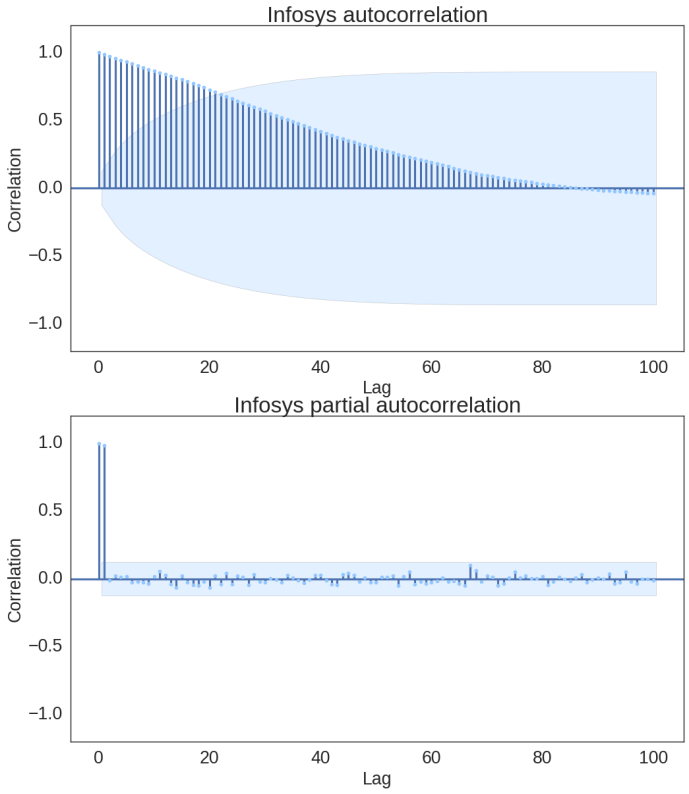


Fig. 18. Train set correlation heatmap after dropping features

To know whether a model is AR, MA or ARMA type, the following rule of thumb can be used:

- If tail-off is in ACF, then it's AR model. The location of the cutoff in PACF will give the order of AR(p).
- If tail-off is in PACF, then it's MA model. The location of the cutoff in ACF will give the order of MA(q).
- If tails-off is present in both, then it's an ARMA model. So need to consider both.

As per this general guideline, we have a spike in PACF at lag=1 and smooth decay of ACF. So, we can conclude it's an AR(1) process. Regarding the order of differencing, we can say roughly it will be 1st order differencing by the nature of ACF and PACF. However, relying on mere intuition based on plots and training a model is not a good idea. So we will automatically determine all the values p,q, and d by a library called 'pmdarima' at the time of training. We will discuss about it in the modelling section.

V. Statistical Model (ARIMA)

For forecasting purposes, we decided to use automated ARIMA, which can come up with p,d, and q values on its own. It's somewhat similar to the grid-search technique. We pass a range of p and q values; the model chooses the best one by minimizing criteria such as the Akaike Information criterion, Bayesian Information Criterion, etc. The model has its confidence level set at 95%, which is the most common for assessing the goodness of a statistical model. For this purpose, we used a library called 'pmdarima'. We will discuss the goodness of this model in the validation section.

VI. Deep Learning Model (LSTM)

We used the TensorFlow library to implement the LSTM neural network. Because the data set is quite small, we used a comparatively simpler model architecture, as shown below. The model was trained on training data and validated on the test data. Test data has about 50 data points.

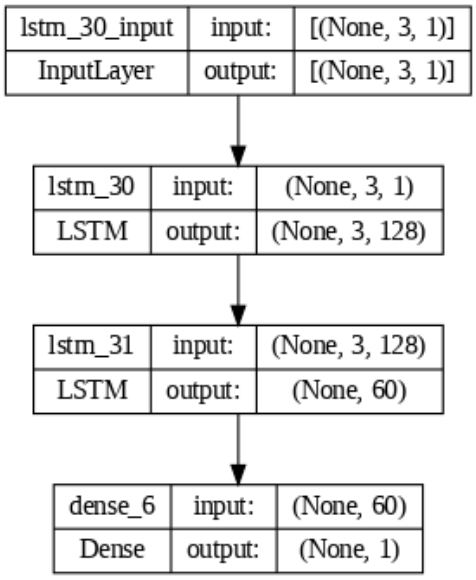


Fig. 19. LSTM architecture

VII. Preprocessing and Splitting the Data

The data is split into 80%-20% train test ratio. For time series, we can't use a large test size because as the time steps increase, the error made by the model increases. Also, we should never shuffle the data while splitting because this is sequential data, so the order in which the model sees this data directly influences its training.

We need a special kind of data preprocessing before training an LSTM model. An RNN model expects a sequence of data. Based on that sequence, it makes one prediction (Depending on the architecture used). So, we generated a sequence of data in a small 3-day window frame. So basically, the model sees 3-day data and then makes a prediction for the 4th day. It keeps

on repeating for all triplets in the test set in a sequence. This special kind of data structure is generated by the Tensorflow window module. Further, we need to add the last training window to the beginning of the test data to get the prediction of the first data in the test set. The reason for keeping such a small 3-day window is to capture as much variation of the data as possible on a daily basis. We could keep a larger window, but the model needs proper hyperparameters to predict the test data correctly. For a 3-day windowed data, the hyperparameters that we chose turned out to be correct by looking at the loss plot. The plot of loss with epoch is given below; sometimes, this helps us assess the model by proper hyperparameter tuning. By looking at this plot, we see both train and validation errors are in the same order; this clearly indicates that the model is performing well and is free from bias and variance.

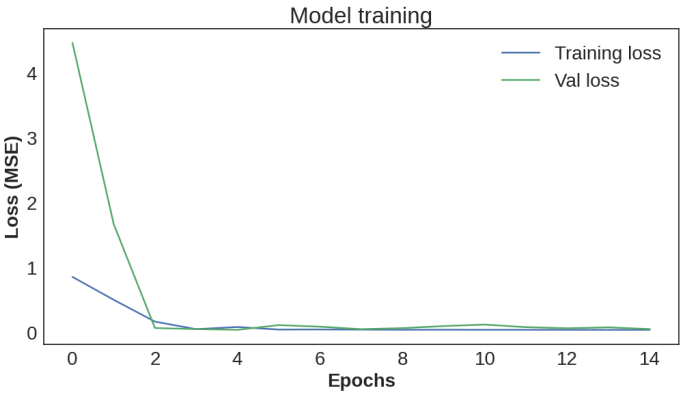


Fig. 20. Model evolution with epochs

VIII. Validation of Models

Since time series prediction is a perfect regression problem, the only difference is that the target is dependent on the same feature data of the past. So conventional metrics such as Mean absolute error and mean squared errors will work fine. We used the Root mean squared error metric for the comparison of the two models.

Models	Root mean squared error
ARIMA	48.20
LSTM	27.09

TABLE I
Comparison of Models

Clearly LSTM model is superior to ARIMA.

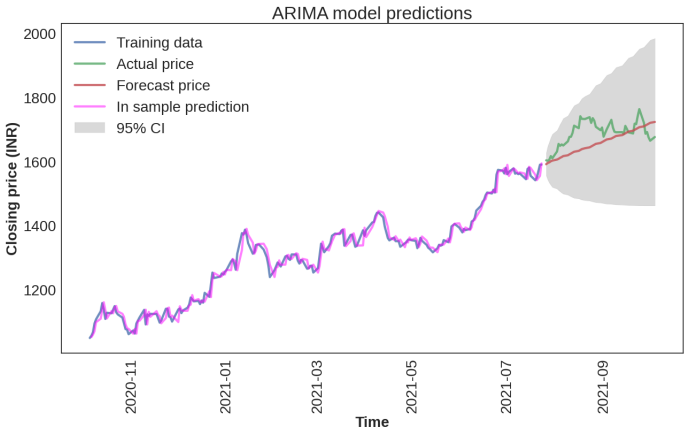


Fig. 21. Prediction of ARIMA model

We see although the ARIMA model has successfully captured the upward trend of the stock price, it couldn't capture the detailed variation of the data. However, the error is not very high; the true value is inside the 95% Confidence interval band.

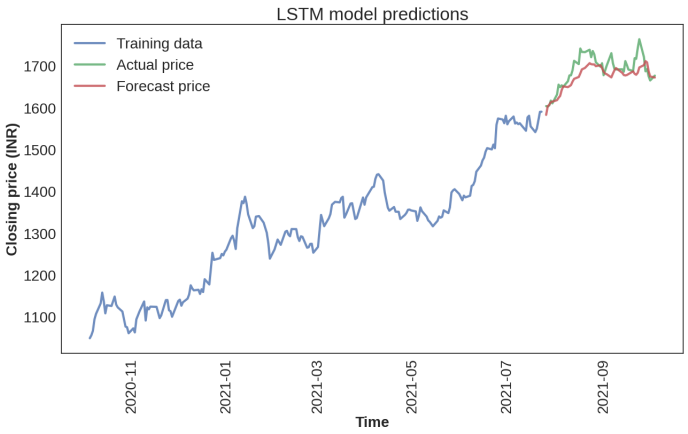


Fig. 22. Predctiction of LSTM

LSTM has clearly captured the minute details of the data. So, it is evidently much better than the conventional ARIMA model.

IX. Conclusions

So, in this paper, we have successfully demonstrated the techniques of working with stock price datasets. We have seen that most of the important information regarding an organisation's stock can be obtained by simple visualization and elementary statistical techniques. For this, we don't even have to perform modelling. Modelling is appropriate for the study of forecast and future trends. For this elementary study, we have performed only visualization of the closing price of the stocks, the daily return rate and risk association with each investment. There are many other such elementary techniques, which we didn't discuss in this paper as this paper aims to provide techniques for working with temporal data. Each different time

series has its unique visualization and elementary working techniques. By our elementary study, Infosys has come out to be the best company for investment in the long run. Classical predictive models such as ARIMA work well with time-series data. Their in-sample predictions are really good. However, when it comes to the prediction of unseen data, these models greatly simplify the output by ignoring complex patterns. LSTM in this place comes as a more powerful tool by capturing these complex patterns induced by seasonality and trends. One avenue for further research can be to experiment with the LSTM model on a bigger window frame with proper cross-validation by hyperparameter tuning.

REFERENCES

[1] Time series: https://builtin.com/data-science/time-series-model
[2] Stocks: https://www.analyticsvidhya.com/blog/2021/07/stock-market-forecasting-using-time-series-analysis-with-arima-model/
[3] ARIMA: https://people.duke.edu/~rnau/411arim.htm
[4] LSTM: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c
[5] mplfinance: https://github.com/matplotlib/mplfinance
[6] plotly: https://plotly.com/python/graph-objects/
[7] TensorFlow: https://www.tensorflow.org/

Access the original code here. Code file also attached below.

Final exam DAL , Time series analysis

Anik Bhowmick

AE20B102

In [ ]:
```
!pip install pmdarima
!pip install mplfinance
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mplfinance as mpf
import plotly.graph_objects as go
from matplotlib import colormaps
from functools import reduce
import copy
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima_model import ARIMA
import pmdarima as pm
import numpy as np
from sklearn.metrics import mean_squared_error as mse
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import tensorflow as tf
from tensorflow import keras
from keras import models,layers
from keras.models import Sequential
from keras.layers import LSTM,Dense,Dropout
from keras.utils import plot_model
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
4.manylinux_2_28_x86_64.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 19.1 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-pack
ages (from pmdarima) (1.3.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/
python3.10/dist-packages (from pmdarima) (3.0.5)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-pac
kages (from pmdarima) (1.23.5)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-pack
ages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dis
t-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-pack
ages (from pmdarima) (1.11.3)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/di
st-packages (from pmdarima) (0.14.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages
(from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/pytho
n3.10/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10/dist-p
ackages (from pmdarima) (23.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.1
0/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pack
ages (from pandas>=0.19->pmdarima) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn>=0.22->pmdarima) (3.2.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-pack
ages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (fro
m patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
Collecting mplfinance
  Downloading mplfinance-0.12.10b0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.0/75.0 kB 2.3 MB/s eta 0:00:00
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packag
es (from mplfinance) (3.7.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from mplfinance) (1.5.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->mplfinance) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-pack
ages (from matplotlib->mplfinance) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist
-packages (from matplotlib->mplfinance) (4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist
-packages (from matplotlib->mplfinance) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib->mplfinance) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-p
ackages (from matplotlib->mplfinance) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib->mplfinance) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->mplfinance) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib->mplfinance) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pack
ages (from pandas->mplfinance) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages
```

```
(from python-dateutil>=2.7->matplotlib->mplfinance) (1.16.0)
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.10b0
```

In [ ]:
```
Data_cognizant=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock price
Data_HCL=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock price predic
Data_HDFC=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock price predi
Data_ICICI=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock price pred
Data_Infosys=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock price pr
Data_SBI=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock price predic
Data_Curr_exchange=pd.read_csv(r'/content/drive/MyDrive/DAL dataset/Endsem stock pr
```

In [ ]:
```
#Utility functions
def set_index_as_column(Data):
  Data=Data.set_index('Date')
  Data.index=pd.to_datetime(Data.index)
  return Data
def get_info(Data):
  Data.info()
def get_statdescr(Data):
  Data.describe()
```

In [ ]:
```
get_info(Data_cognizant)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 694 entries, 0 to 693
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    694 non-null    object 
 1   Open    694 non-null    float64
 2   High    694 non-null    float64
 3   Low     694 non-null    float64
 4   Close   694 non-null    float64
 5   Volume  694 non-null    int64  
dtypes: float64(4), int64(1), object(1)
memory usage: 32.7+ KB
```

In [ ]:
```
Data_cognizant=set_index_as_column(Data_cognizant)
```

In [ ]:
```
get_info(Data_HCL)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249 entries, 0 to 248
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    249 non-null    object 
 1   Open    248 non-null    float64
 2   High    248 non-null    float64
 3   Low     248 non-null    float64
 4   Close   248 non-null    float64
 5   Volume  248 non-null    float64
dtypes: float64(5), object(1)
memory usage: 11.8+ KB
```

In [ ]:
```
Data_HCL=set_index_as_column(Data_HCL)
Data_HCL[Data_HCL.isnull().any(axis=1)]#This is saturday, stock markets remain clos
```

Out[ ]:

Date	Open	High	Low	Close	Volume
2020-11-14	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
Data_HCL.dropna(inplace=True)
Data_HCL.isnull().any()
```

Out[ ]:
```
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

In [ ]:
```
get_info(Data_HDFC)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 680 entries, 0 to 679
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    680 non-null    object 
 1   Open    678 non-null    float64
 2   High    678 non-null    float64
 3   Low     678 non-null    float64
 4   Close   678 non-null    float64
 5   Volume  678 non-null    float64
dtypes: float64(5), object(1)
memory usage: 32.0+ KB
```

In [ ]:
```
Data_HDFC=set_index_as_column(Data_HDFC)
Data_HDFC[Data_HDFC.isnull().any(axis=1)]# These are wrong entries for saturday and
```

Out[ ]:

Date	Open	High	Low	Close	Volume
2019-10-27	NaN	NaN	NaN	NaN	NaN
2020-11-14	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
Data_HDFC.dropna(inplace=True)
Data_HDFC.isnull().any()
```

Out[ ]:
```
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

In [ ]:
```
get_info(Data_ICICI)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 680 entries, 0 to 679
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    680 non-null    object 
 1   Open    678 non-null    float64
 2   High    678 non-null    float64
 3   Low     678 non-null    float64
 4   Close   678 non-null    float64
 5   Volume  678 non-null    float64
dtypes: float64(5), object(1)
memory usage: 32.0+ KB
```

In [ ]:
```
Data_ICICI=set_index_as_column(Data_ICICI)
Data_ICICI[Data_ICICI.isnull().any(axis=1)]# These are wrong entries for saturday c
```

Out[ ]:

Date	Open	High	Low	Close	Volume
2019-10-27	NaN	NaN	NaN	NaN	NaN
2020-11-14	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
Data_ICICI.dropna(inplace=True)
Data_ICICI.isnull().any()
```

Out[ ]:
```
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

In [ ]:
```
get_info(Data_Infosys)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249 entries, 0 to 248
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    249 non-null    object 
 1   Open    248 non-null    float64
 2   High    248 non-null    float64
 3   Low     248 non-null    float64
 4   Close   248 non-null    float64
 5   Volume  248 non-null    float64
dtypes: float64(5), object(1)
memory usage: 11.8+ KB
```

In [ ]:
```
Data_Infosys=set_index_as_column(Data_Infosys)
Data_Infosys[Data_Infosys.isnull().any(axis=1)]
```

Out[ ]:

Date	Open	High	Low	Close	Volume
2020-11-14	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
Data_Infosys.dropna(inplace=True)
Data_Infosys.isnull().any()
```

Out[ ]:
```
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

In [ ]:
```
get_info(Data_SBI)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 680 entries, 0 to 679
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    680 non-null    object
 1   Open    678 non-null    float64
 2   High    678 non-null    float64
 3   Low     678 non-null    float64
 4   Close   678 non-null    float64
 5   Volume  678 non-null    float64
dtypes: float64(5), object(1)
memory usage: 32.0+ KB
```

In [ ]:
```
Data_SBI=set_index_as_column(Data_SBI)
Data_SBI[Data_SBI.isnull().any(axis=1)]
```

Out[ ]:

Date	Open	High	Low	Close	Volume
2019-10-27	NaN	NaN	NaN	NaN	NaN
2020-11-14	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
Data_SBI.dropna(inplace=True)
Data_SBI.isnull().any()
```

Out[ ]:
```
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

In [ ]:
```
get_info(Data_Curr_exchange)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 720 entries, 0 to 719
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       720 non-null    object
 1   Open       699 non-null    float64
 2   High       699 non-null    float64
 3   Low        699 non-null    float64
 4   Close      699 non-null    float64
 5   Adj Close  699 non-null    float64
dtypes: float64(5), object(1)
memory usage: 33.9+ KB
```

In [ ]:
```
Data_Curr_exchange=set_index_as_column(Data_Curr_exchange)
Data_Curr_exchange[Data_Curr_exchange.isnull().any(axis=1)]#Useless to keep, drop d
```

Out[ ]:

Date	Open	High	Low	Close	Adj Close
2019-05-22	NaN	NaN	NaN	NaN	NaN
2019-09-11	NaN	NaN	NaN	NaN	NaN
2019-09-12	NaN	NaN	NaN	NaN	NaN
2019-09-13	NaN	NaN	NaN	NaN	NaN
2019-09-16	NaN	NaN	NaN	NaN	NaN
2019-09-17	NaN	NaN	NaN	NaN	NaN
2019-09-18	NaN	NaN	NaN	NaN	NaN
2019-09-19	NaN	NaN	NaN	NaN	NaN
2019-09-20	NaN	NaN	NaN	NaN	NaN
2019-09-23	NaN	NaN	NaN	NaN	NaN
2019-09-24	NaN	NaN	NaN	NaN	NaN
2019-09-25	NaN	NaN	NaN	NaN	NaN
2019-09-26	NaN	NaN	NaN	NaN	NaN
2019-09-27	NaN	NaN	NaN	NaN	NaN
2019-09-30	NaN	NaN	NaN	NaN	NaN
2019-10-01	NaN	NaN	NaN	NaN	NaN
2019-10-02	NaN	NaN	NaN	NaN	NaN
2019-10-03	NaN	NaN	NaN	NaN	NaN
2019-10-04	NaN	NaN	NaN	NaN	NaN
2019-10-07	NaN	NaN	NaN	NaN	NaN
2019-10-09	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
Data_Curr_exchange.dropna(inplace=True)
```

Exploratory data analysis

In [ ]:
```
#Get the candle stick plot
def plot_candle_stick(Data,text):
  fig = go.Figure(data=[go.Candlestick(x=Data.index,
                open=Data['Open'],
                high=Data['High'],
                low=Data['Low'],
                close=Data['Close'])])
  fig.update_layout(
    title=text,
    title_font=dict(size=30),
    title_x=0.5,
    xaxis_title='Date',
    xaxis_title_font=dict(size=20),
    yaxis_title=r'Stock Price (INR)',
    yaxis_title_font=dict(size=20),xaxis_rangeslider_visible=False,paper_bgcolor='#
  fig.show()
```

```
def get_moving_avg(Data,text):
  mystyle = mpf.make_mpf_style(base_mpf_style='charles',rc={'font.size':30})
  fig,ax=mpf.plot(Data,type='line',mav=(30,50),mavcolors=('orange', 'green'),dateti
  ax[0].legend(['Stock Price','SMA(30)','SMA(50)'],fontsize=30)
  ax[0].set_title(text, fontsize=35)
  #ax[0].tick_params(axis='y', which='both', labelsize=25)
  plt.savefig(f'{text}.png',bbox_inches='tight')
  plt.show()
```

In [ ]:
```
#plot_candle_stick(Data_cognizant,'Cognizant Stock price')
```

In [ ]:
```
#plot_candle_stick(Data_HCL,'HCL Stock price')
```

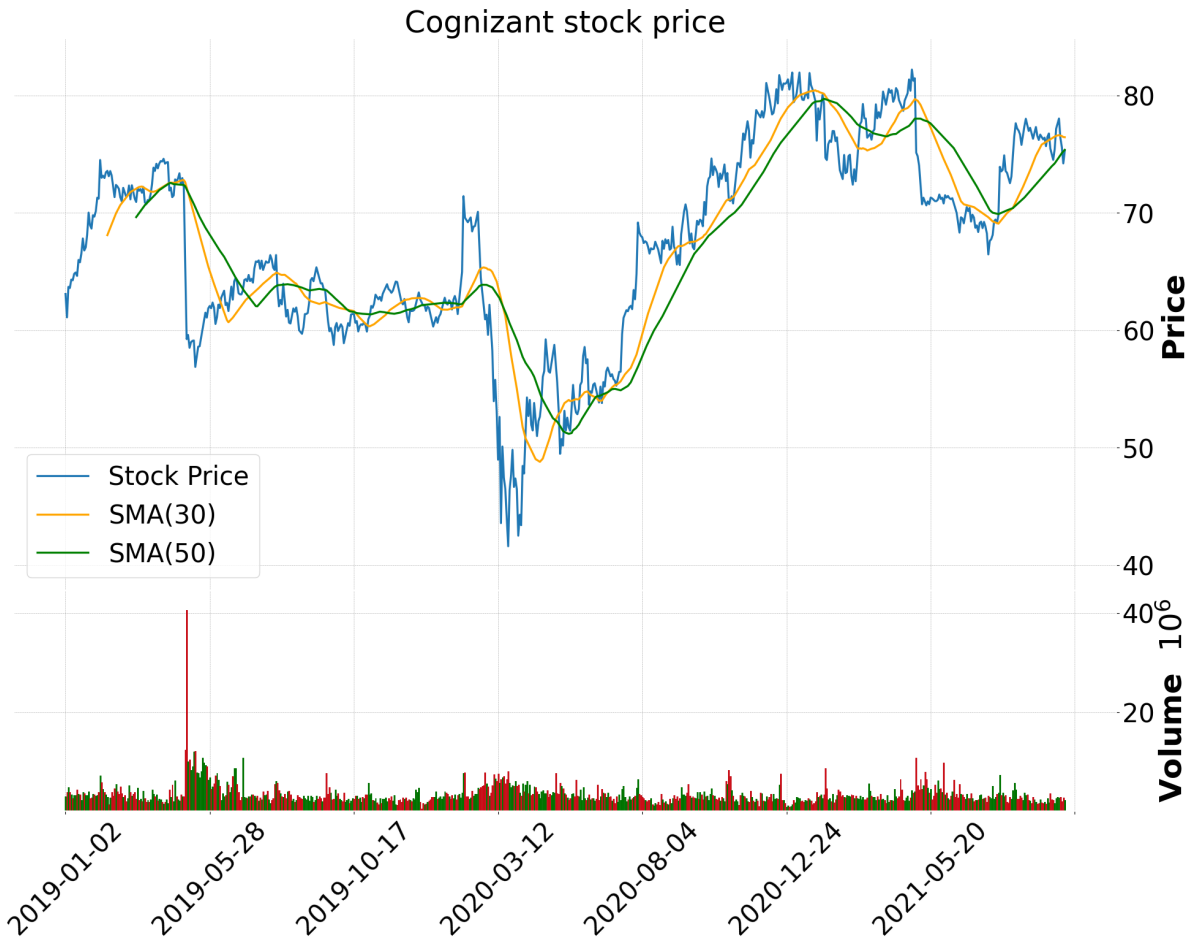
In [ ]:
```
#plot_candle_stick(Data_HDFC,'HDFC Stock price')
```

In [ ]:
```
#plot_candle_stick(Data_ICICI,'ICICI Stock price')
```

In [ ]:
```
#plot_candle_stick(Data_Infosys,'Infosys Stock price')
```

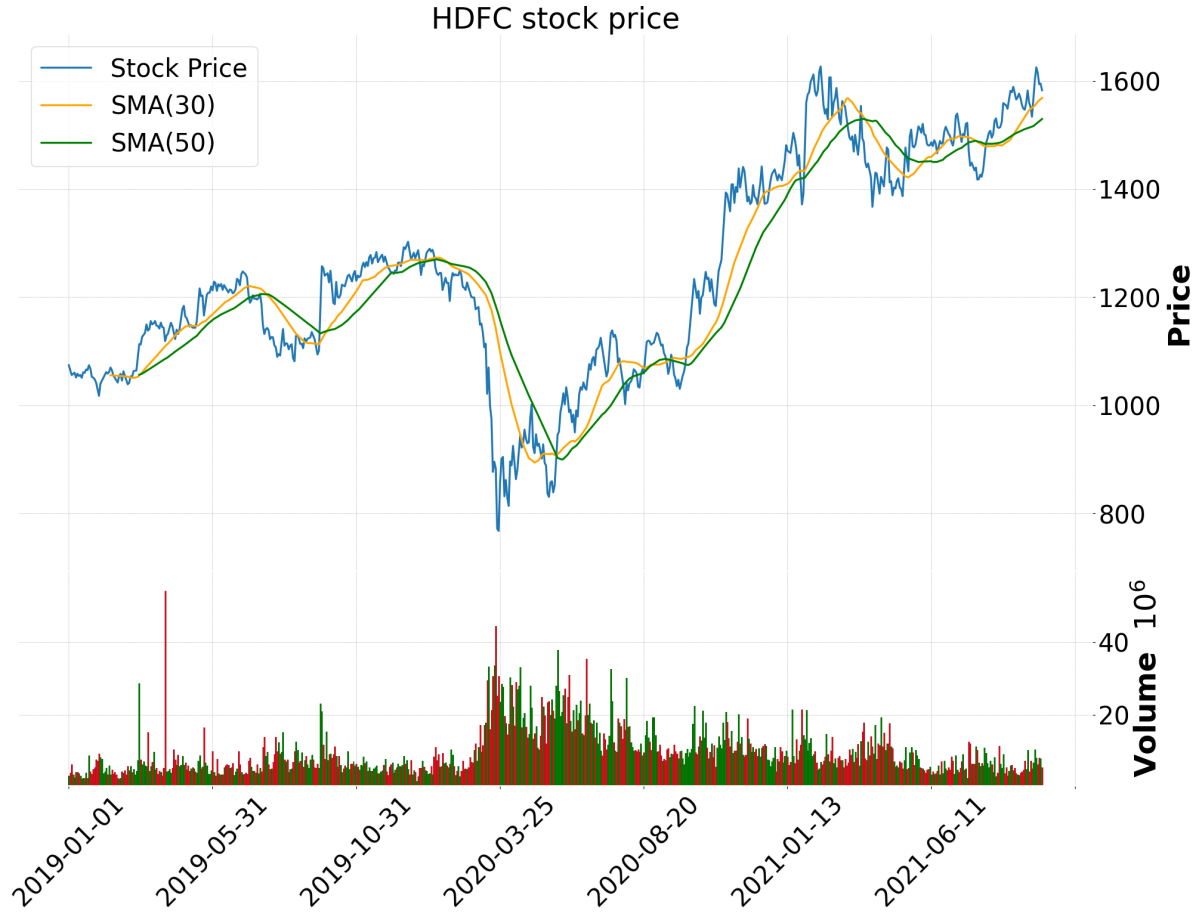
In [ ]:
```
#plot_candle_stick(Data_SBI,'SBI Stock price')
```

In [ ]:
```
get_moving_avg(Data_cognizant,'Cognizant stock price')
```

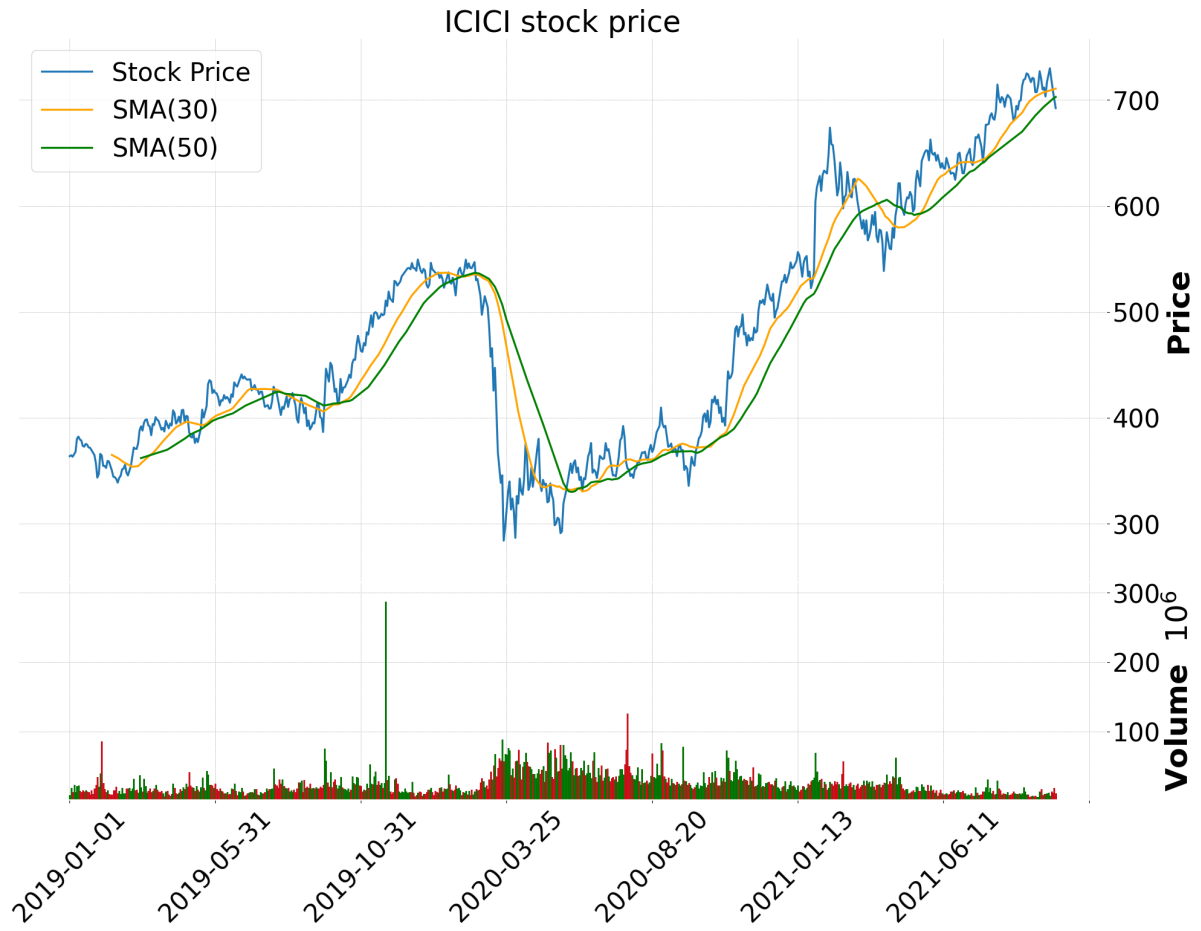


In [ ]:
```
get_moving_avg(Data_HCL,'HCL stock price')
```

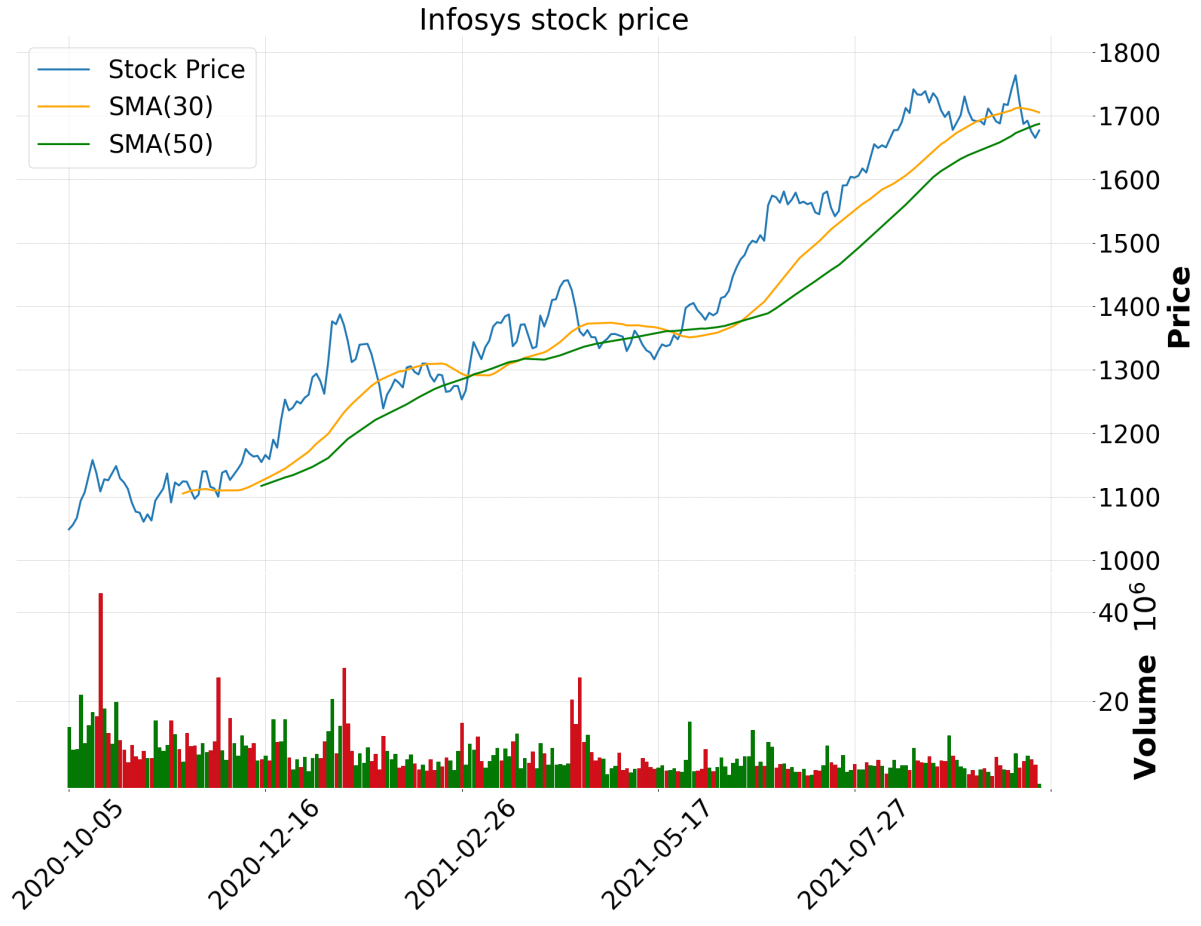
In [ ]:
```
get_moving_avg(Data_HDFC,'HDFC stock price')
```



In [ ]:
```
get_moving_avg(Data_ICICI,'ICICI stock price')
```

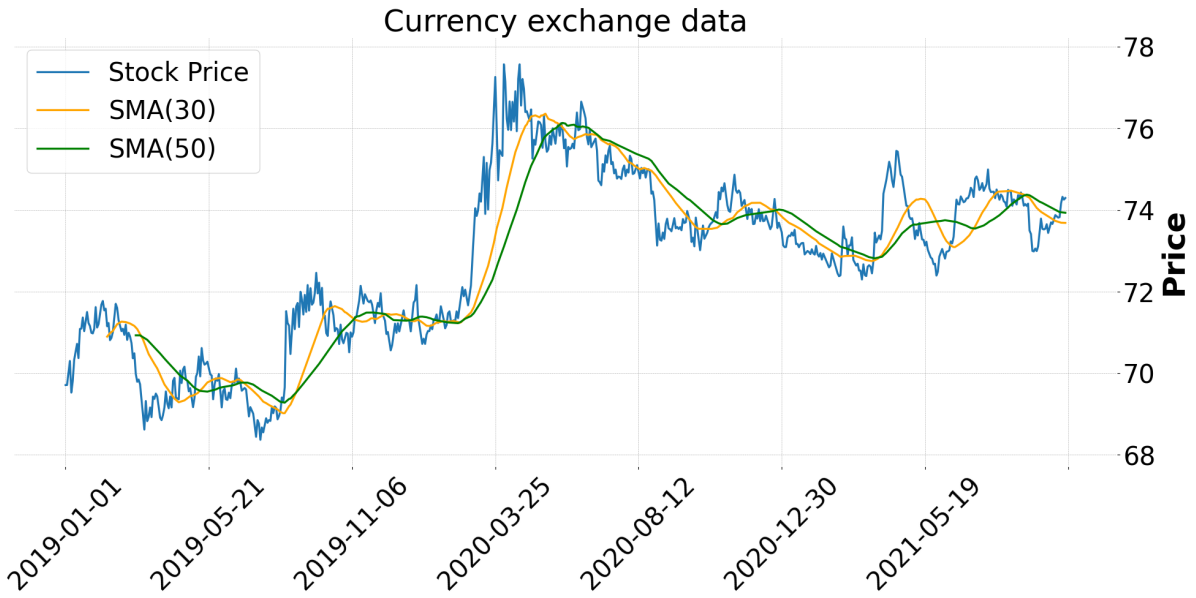


In [ ]:
```
get_moving_avg(Data_Infosys,'Infosys stock price')
```



In [ ]:
```
get_moving_avg(Data_SBI,'SBI stock price')
```

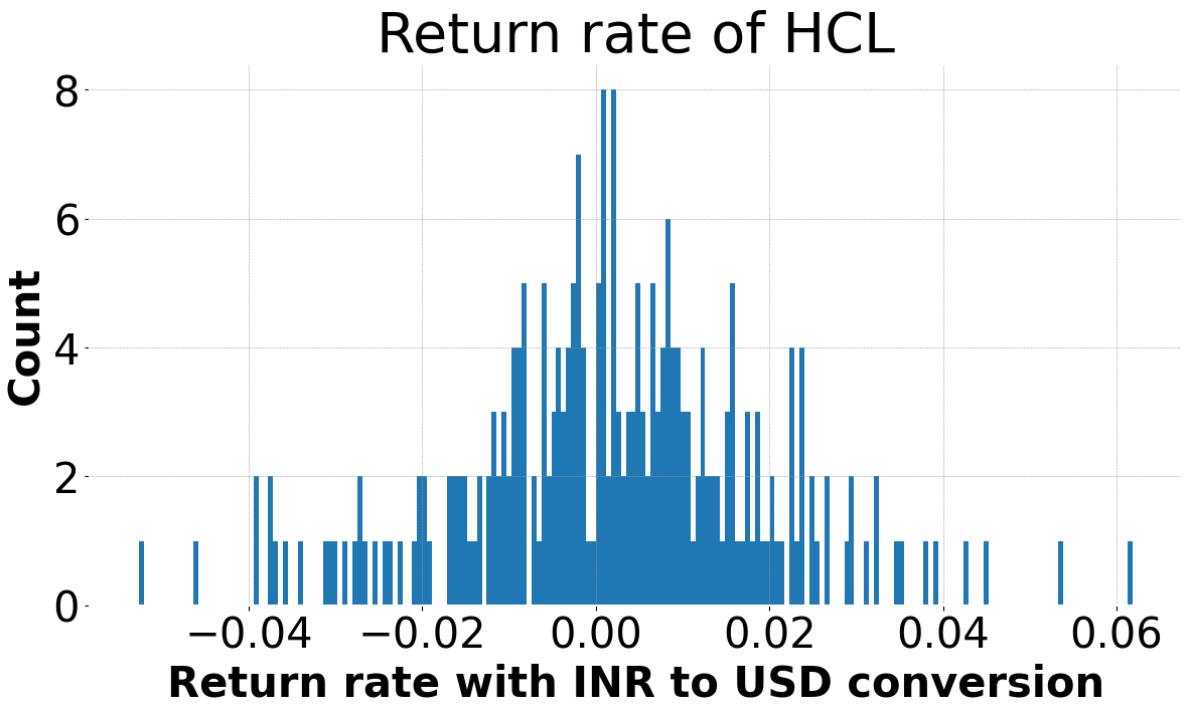
In [ ]:
```
mystyle = mpf.make_mpf_style(base_mpf_style='charles',rc={'font.size':30})
fig,ax=mpf.plot(Data_Curr_exchange,type='line',mav=(30,50),mavcolors=('orange', 'gr
ax[0].legend(['Stock Price','SMA(30)','SMA(50)'],fontsize=30)
ax[0].set_title('Currency exchange data', fontsize=35)
  #ax[0].tick_params(axis='y', which='both', labelsize=25)
plt.savefig('Currency exchange data.png',bbox_inches='tight')
plt.show()
```



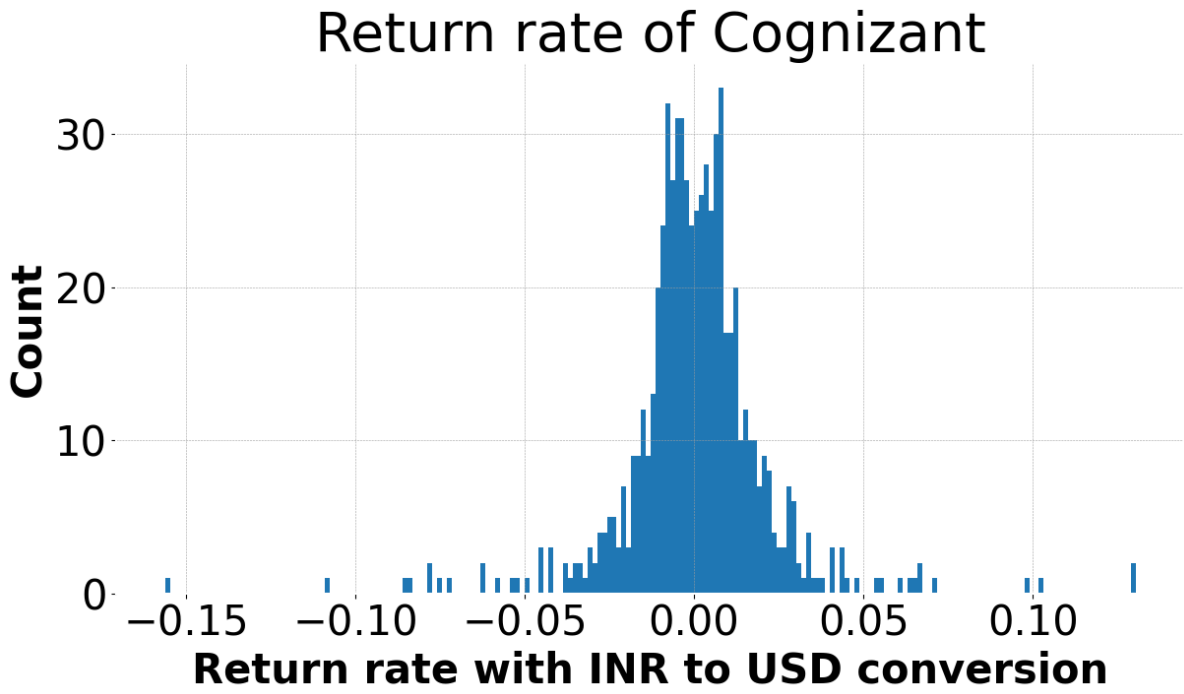
In [ ]:
```
#calculate the daily return rate without currency exchange value
def Plot_DOR(Data,text):
  plt.figure(figsize = (14, 7))
  Data['Daily return'].hist(bins = 200)
  plt.title(f'Return rate of {text}',fontsize=40)
  plt.xticks(fontsize=30)
  plt.yticks(fontsize=30)
```

```
  plt.xlabel('Return rate with INR to USD conversion',fontsize=30)
  plt.ylabel('Count',fontsize=30)
  plt.savefig(f'{text} return.png',bbox_inches='tight')
def Daily_return(Data,text,currency):
  DOR=pd.DataFrame((Data['Close']/currency['Close'])/(Data['Close'].shift(1)/currer
  DOR.rename(columns = {'Close':'Daily return'}, inplace = True)
  Plot_DOR(DOR,text)
  return DOR
def plot_risk(RR):
  colors=[]
  c = colormaps.get_cmap('YlGnBu')
  scaled_data = (RR-RR.min())/(RR.max() - RR.min())
  for decimal in scaled_data['Standard Deviation']:
    colors.append(c(decimal))
  fig, ax = plt.subplots(figsize=(14,7))
  bars = ax.bar(RR.index, RR['Standard Deviation'],color=colors)
  ax.set_title('Comparison of risk across companies',fontsize=40)
  ax.set_xlabel('Company',fontsize=20)
  ax.set_ylabel('Risk (Standard deviation)',fontsize=20)
  ax.tick_params(axis='x',labelsize=30,rotation=90)
  ax.tick_params(axis='y',labelsize=30)
  plt.savefig('Risk.png',bbox_inches='tight')
```

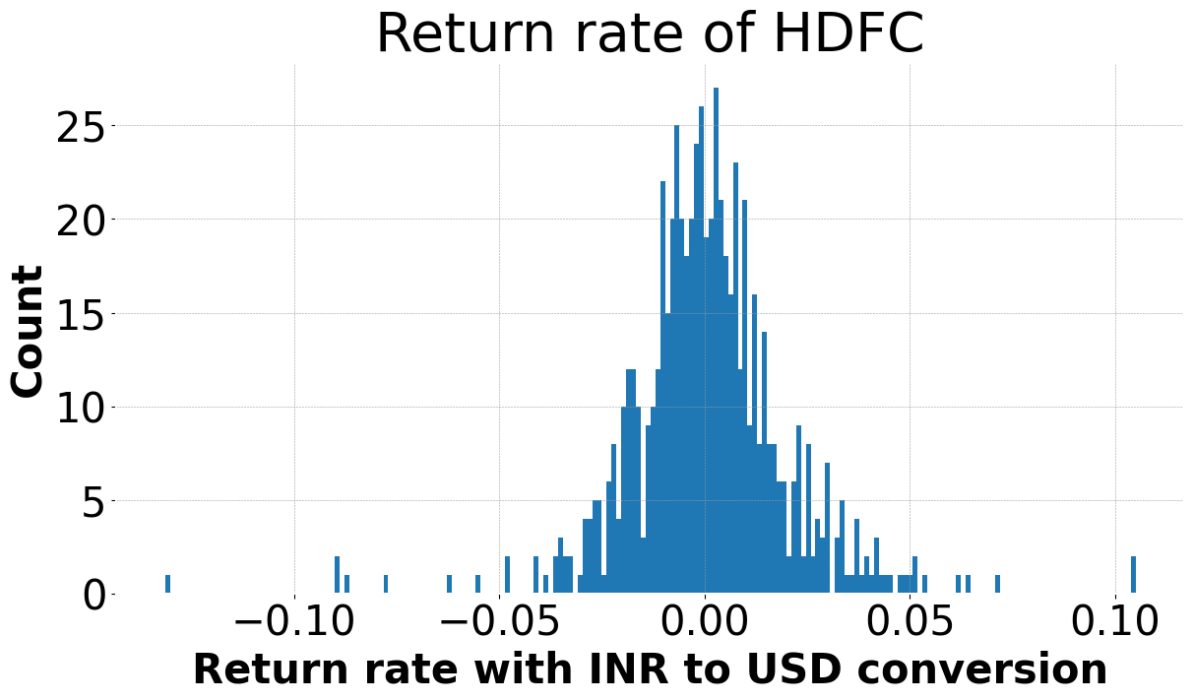
In [ ]:
```
HCL_return_rate=Daily_return(Data_HCL,'HCL',Data_Curr_exchange)
```



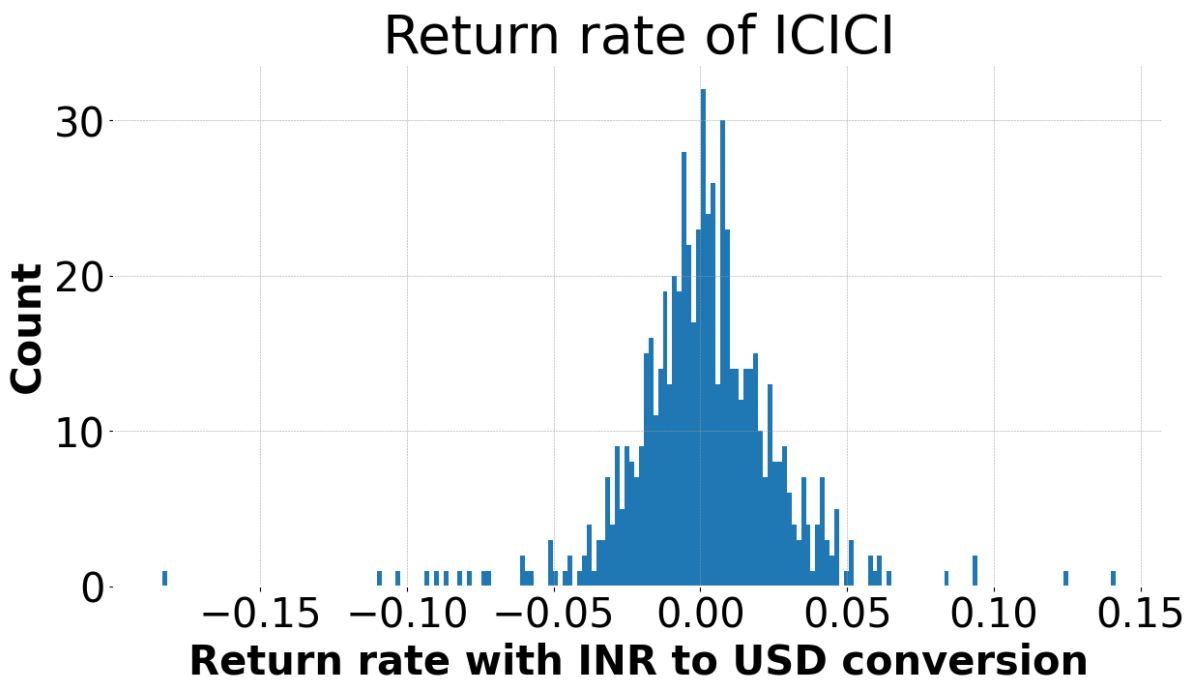
In [ ]:
```
Cognizant_return_rate=Daily_return(Data_cognizant,'Cognizant',Data_Curr_exchange)
```



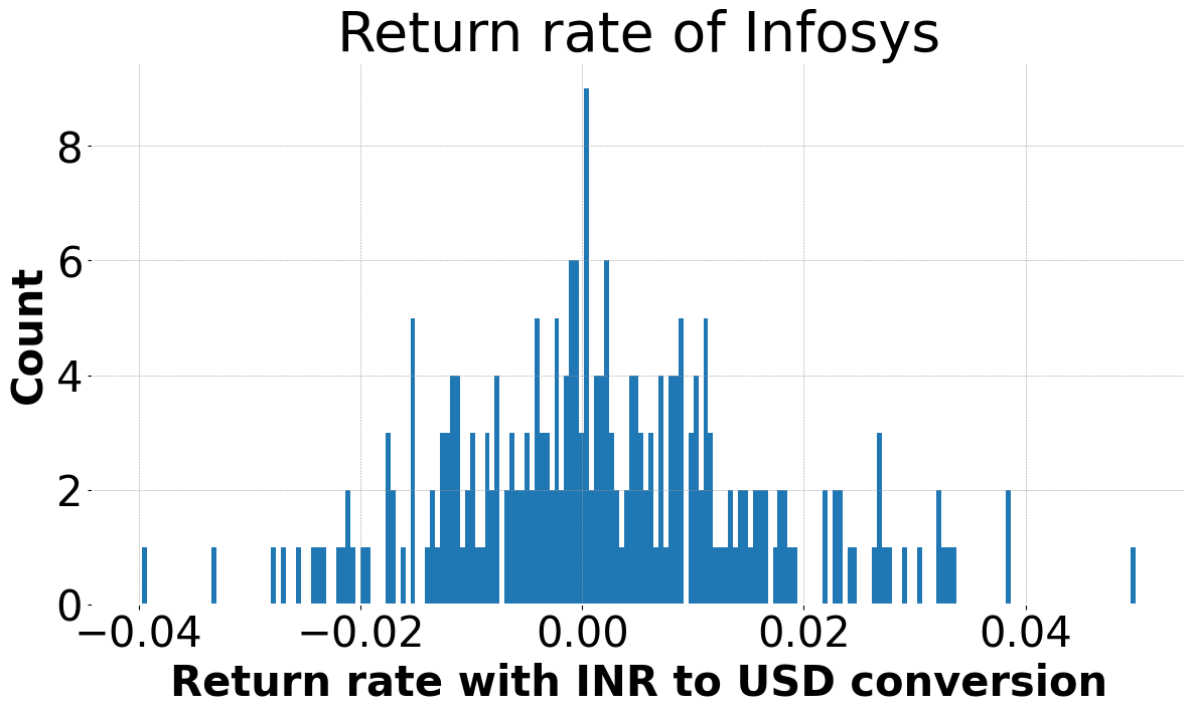
In [ ]:
```
HDFC_return_rate=Daily_return(Data_HDFC,'HDFC',Data_Curr_exchange)
```



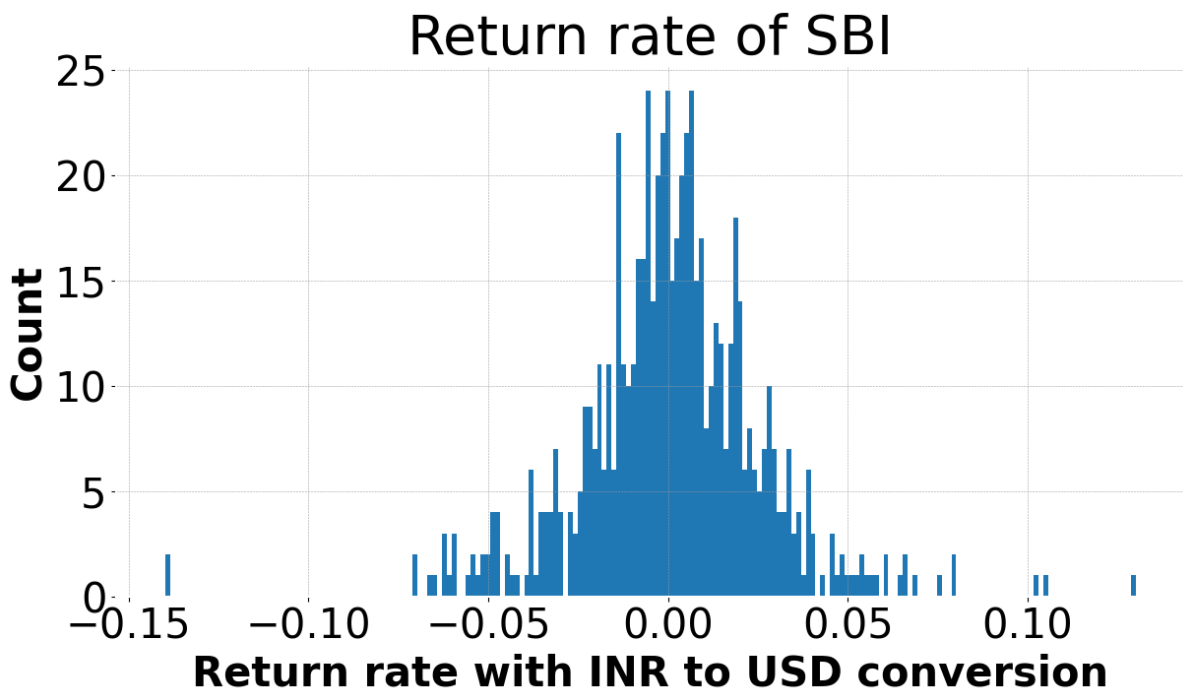
In [ ]:
```
ICICI_return_rate=Daily_return(Data_ICICI,'ICICI',Data_Curr_exchange)
```



In [ ]:
```
Infosys_return_rate=Daily_return(Data_Infosys,'Infosys',Data_Curr_exchange)
```



In [ ]:
```
SBI_return_rate=Daily_return(Data_SBI,'SBI',Data_Curr_exchange)
```



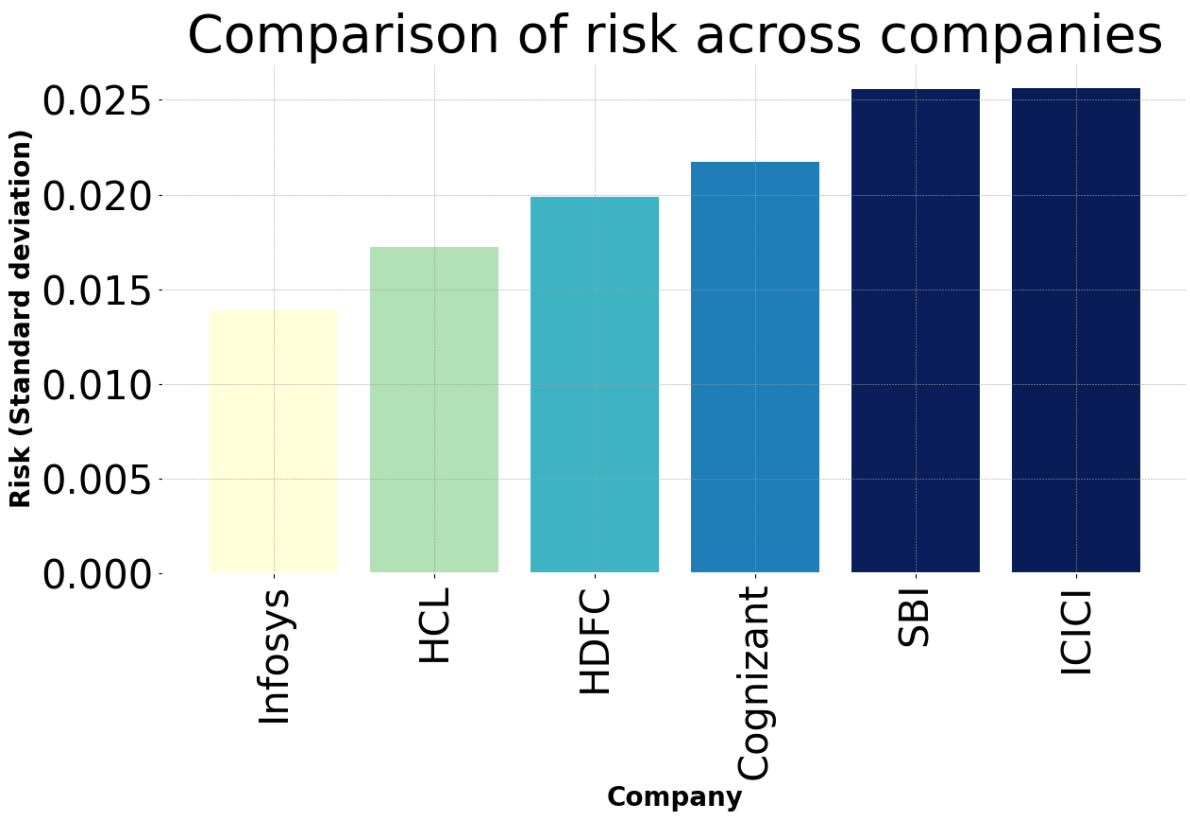
In [ ]:
```
D=[Infosys_return_rate.std().values[0],HCL_return_rate.std().values[0],Cognizant_re
                        SBI_return_rate.std().values[0],ICICI_return_rate.std().val
Risk=pd.DataFrame(data=D,index=['Infosys','HCL','Cognizant','HDFC','SBI','ICICI'],c
Risk=Risk.sort_values('Standard Deviation')
```

In [ ]:
```
Risk
```

Out[ ]:

	Standard Deviation
Infosys	0.013871
HCL	0.017250
HDFC	0.019889
Cognizant	0.021741
SBI	0.025592
ICICI	0.025641

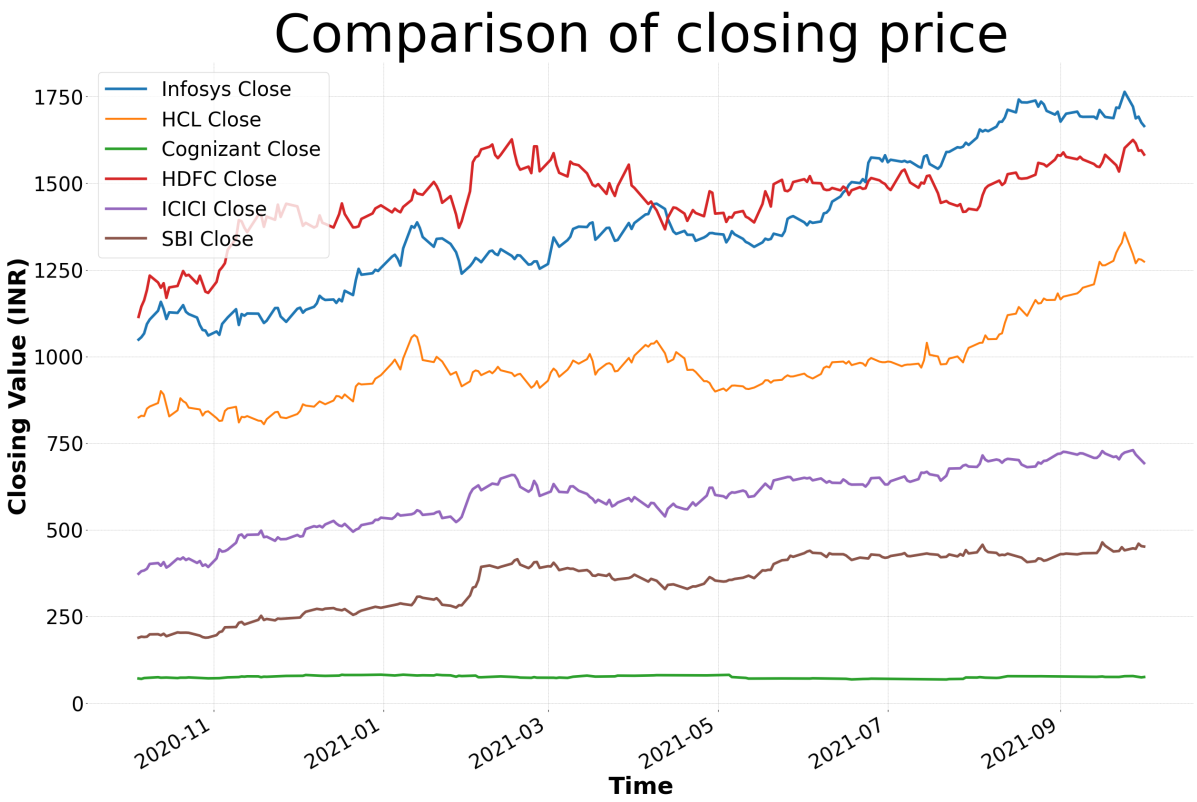
In [ ]:
```
plot_risk(Risk)
```



In [ ]:
```
#Check for stationarity
# From now onwards only focus will be given to Closing price
Infosys_close=copy.deepcopy(Data_Infosys[['Close']])
Infosys_close.rename(columns = {'Close':'Infosys Close'},inplace=True)
HCL_close=copy.deepcopy(Data_HCL[['Close']])
HCL_close.rename(columns = {'Close':'HCL Close'},inplace=True)
Cognizant_close=copy.deepcopy(Data_cognizant[['Close']])
Cognizant_close.rename(columns = {'Close':'Cognizant Close'},inplace=True)
SBI_close=copy.deepcopy(Data_SBI[['Close']])
SBI_close.rename(columns = {'Close':'SBI Close'},inplace=True)
ICICI_close=copy.deepcopy(Data_ICICI[['Close']])
ICICI_close.rename(columns = {'Close':'ICICI Close'},inplace=True)
HDFC_close=copy.deepcopy(Data_HDFC[['Close']])
HDFC_close.rename(columns = {'Close':'HDFC Close'},inplace=True)
```

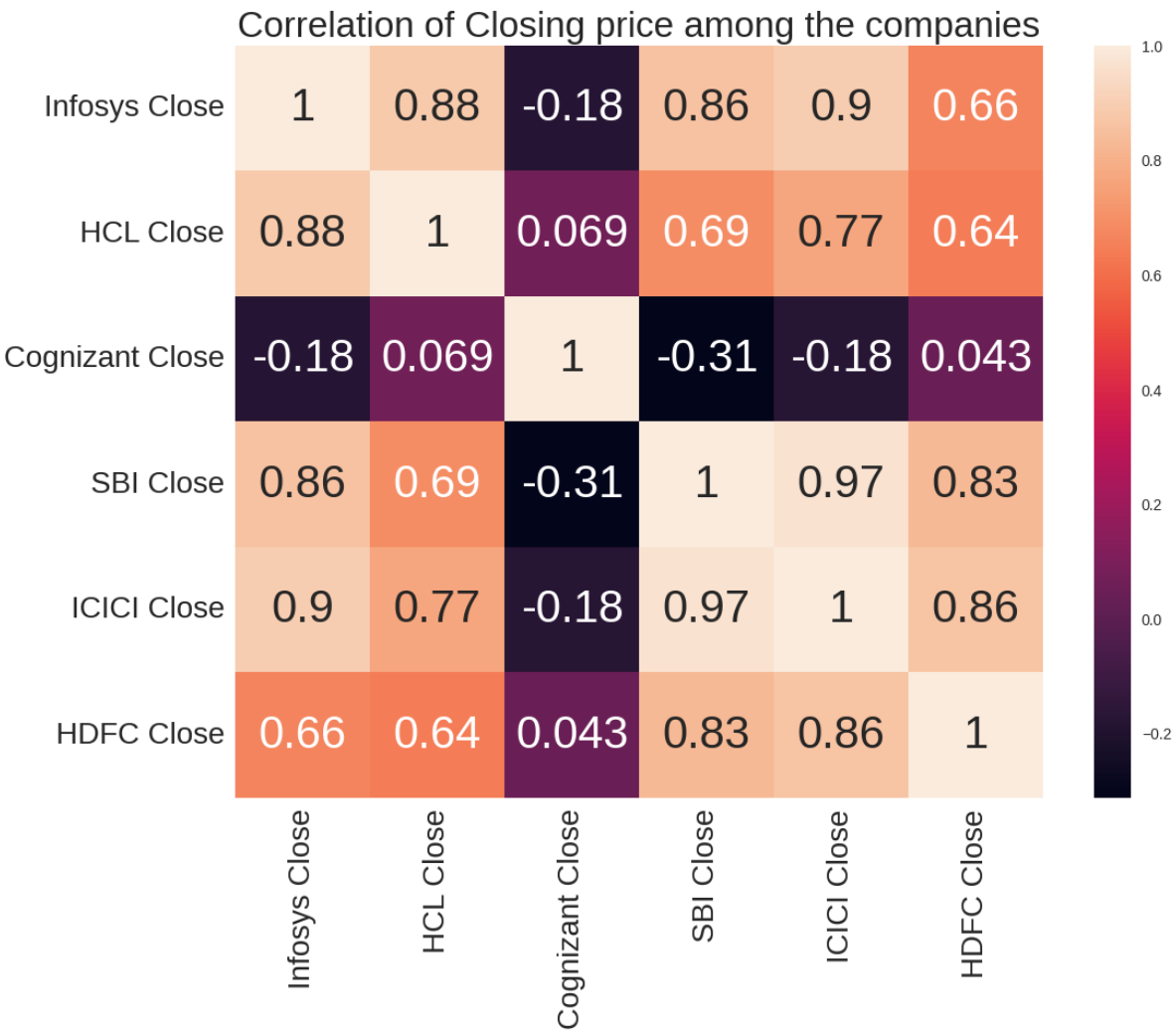
In [ ]:
```
Combined_data=reduce(lambda  left,right: pd.merge(left,right,how='inner',left_index
```

In [ ]:
```
plt.figure(figsize=(30,20))
plt.title("Comparison of closing price",fontsize=80)
Combined_data['Infosys Close'].plot(legend=True,lw=4)
Combined_data['HCL Close'].plot(legend=True,lw=3)
Combined_data['Cognizant Close'].plot(legend=True,lw=4)
Combined_data['HDFC Close'].plot(legend=True,lw=4)
Combined_data['ICICI Close'].plot(legend=True,lw=4)
Combined_data['SBI Close'].plot(legend=True,lw=4)
plt.ylabel('Closing Value (INR)')
plt.xlabel('Time')
plt.savefig('Combined_close.png',bbox_inches='tight')
plt.show()
```



In [ ]:

```
plt.style.use('seaborn-v0_8')
plt.figure(figsize=(12,9))
plt.title("Correlation of Closing price among the companies",fontsize=24)
sns.heatmap(Combined_data.corr(),annot=True)
plt.xticks(rotation=90,fontsize=20)
plt.yticks(rotation=0,fontsize=20)
plt.show()
```



We see the risk is less and closing price of Infosys is higher than any other companies. So by investors may be more interested in investing in Infosys. So we will consider the modelling part only for Infosys

Modeling

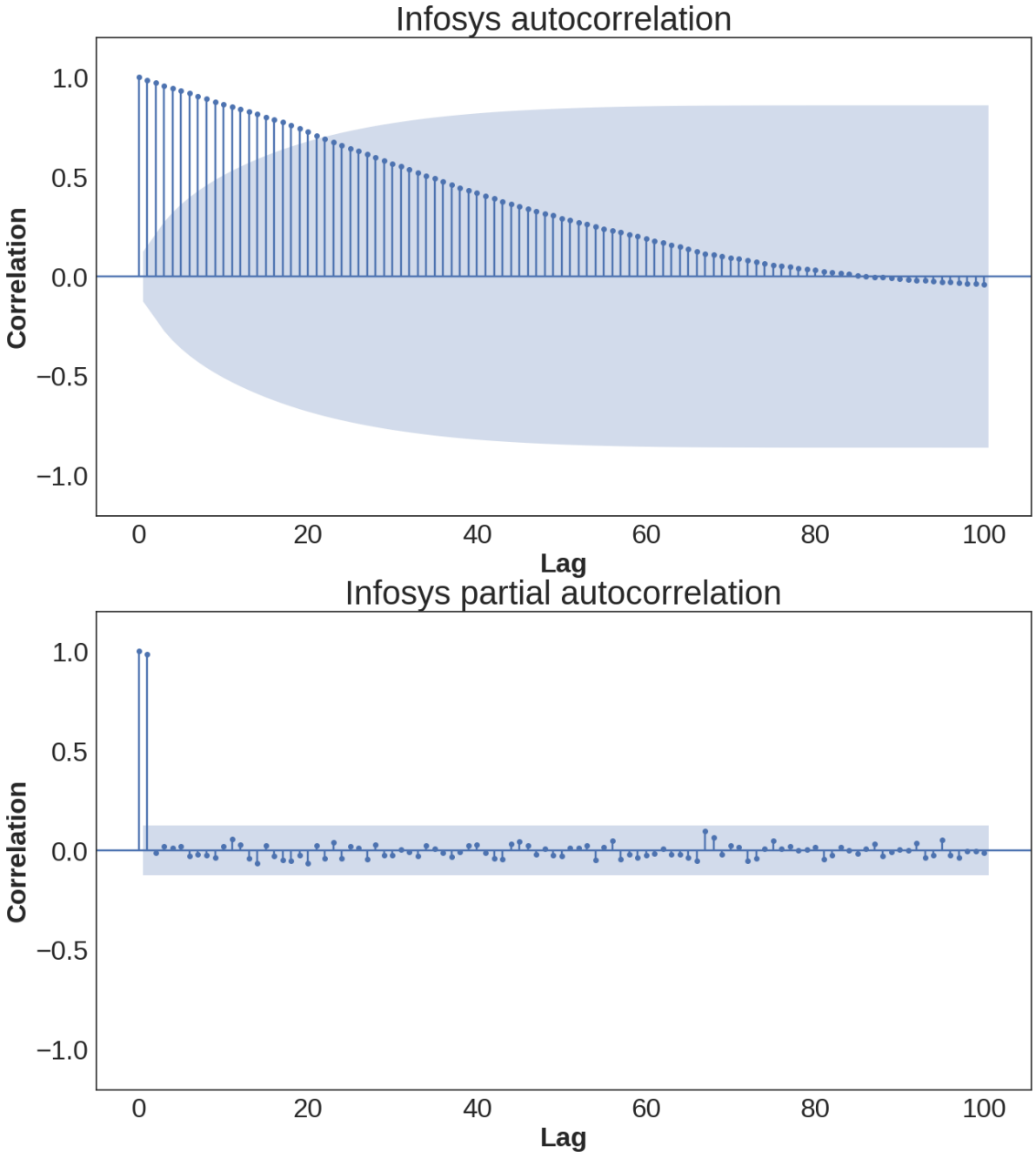
In [ ]:
```
#Test for stationarity of closing price of Infosys
print('p value for Infosys: ',adfuller(Infosys_close.values)[1])
print('p value for HCL: ',adfuller(HCL_close.values)[1])
print('p value for Cognizant: ',adfuller(Cognizant_close.values)[1])
print('p value for ICICI: ',adfuller(SBI_close.values)[1])
print('p value for HDFC: ',adfuller(HDFC_close.values)[1])
```

```
p value for Infosys:  0.772129858462079
p value for HCL:  0.9752910000168649
p value for Cognizant:  0.25317145678908437
p value for ICICI:  0.8149816450211009
p value for HDFC:  0.7261646925917381
```

At 5% significance level none of the dataset is stationary.

In [ ]:
```
plt.style.use('seaborn-v0_8-white')
fig,ax=plt.subplots(nrows=2, ncols=1, figsize=(15, 17))
plot_acf(Infosys_close,lags=100,ax=ax[0])
plot_pacf(Infosys_close,lags=100,ax=ax[1])
ax[0].set_title('Infosys autocorrelation',fontsize=30)
ax[0].set_xlabel('Lag',fontsize=24)
ax[0].set_ylabel('Correlation',fontsize=24)
```

```
ax[0].tick_params(axis='both',labelsize=24)
ax[0].set_ylim(-1.2,1.2)
ax[1].set_title('Infosys partial autocorrelation',fontsize=30)
ax[1].set_ylabel('Correlation',fontsize=24)
ax[1].set_xlabel('Lag',fontsize=24)
ax[1].tick_params(axis='both',labelsize=24)
ax[1].set_ylim(-1.2,1.2)
plt.show()
```



Split the data into train test

In [ ]:
```
training_data_len = int(np.ceil( len(Infosys_close) * .80 ))
train = Infosys_close.iloc[:training_data_len]
test = Infosys_close.iloc[training_data_len:]
```

In [ ]:
```
model = pm.auto_arima(train.values, start_p=1, start_q=1,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=4, max_q=4, # maximum p and q
                                 # frequency of series
```

```
                      d=None,           # let model determine 'd'
                      seasonal=True,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True,
                      with_intercept=True)

print(model.summary())
```

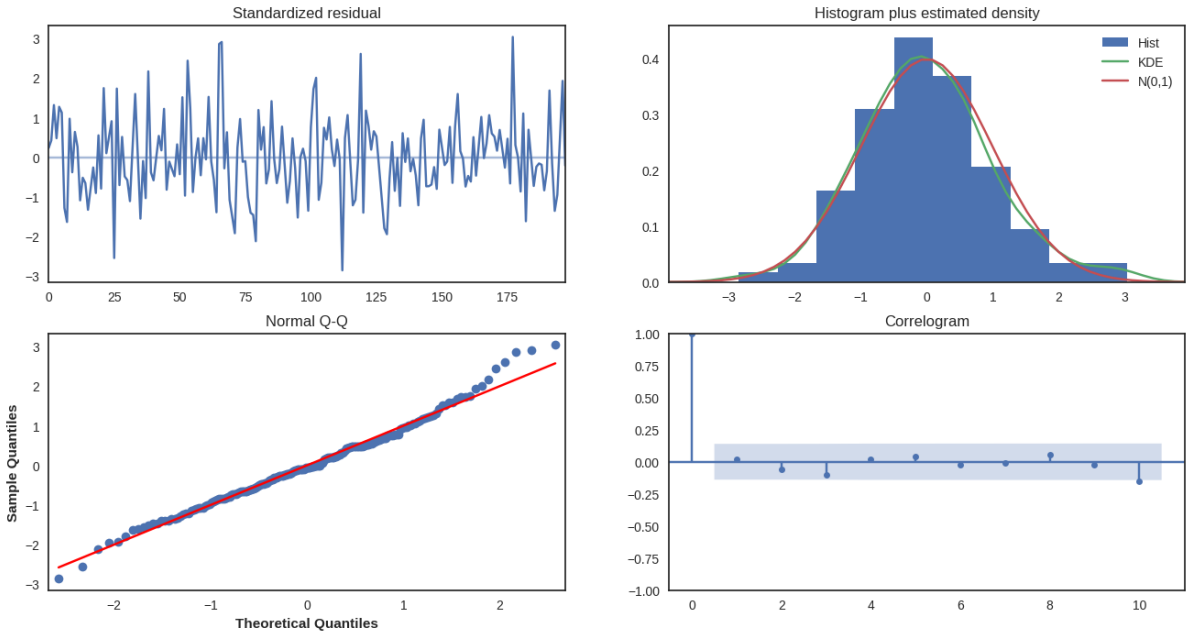
```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1716.715, Time=1.00 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1718.621, Time=0.09 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1718.885, Time=0.26 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1718.325, Time=0.43 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1720.964, Time=0.06 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1719.041, Time=1.03 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1719.109, Time=1.22 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=1717.379, Time=0.63 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1717.313, Time=0.77 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1720.990, Time=1.51 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=1718.746, Time=0.29 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0] intercept
Total fit time: 7.396 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  199
Model:               SARIMAX(1, 1, 1)   Log Likelihood                -854.357
Date:                Sat, 25 Nov 2023   AIC                           1716.715
Time:                        23:10:42   BIC                           1729.868
Sample:                             0   HQIC                          1722.039
                                  - 199
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      5.0098      2.539      1.973      0.049       0.032       9.987
ar.L1         -0.8295      0.096     -8.670      0.000      -1.017      -0.642
ma.L1          0.9273      0.072     12.966      0.000       0.787       1.067
sigma2       327.3509     30.250     10.821      0.000     268.061     386.640
===================================================================================
=
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):                 5.8
5
Prob(Q):                              0.75   Prob(JB):                         0.0
5
Heteroskedasticity (H):               0.60   Skew:                             0.3
2
Prob(H) (two-sided):                  0.04   Kurtosis:                         3.5
4
===================================================================================
=

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

So p=1,q=1,d=1

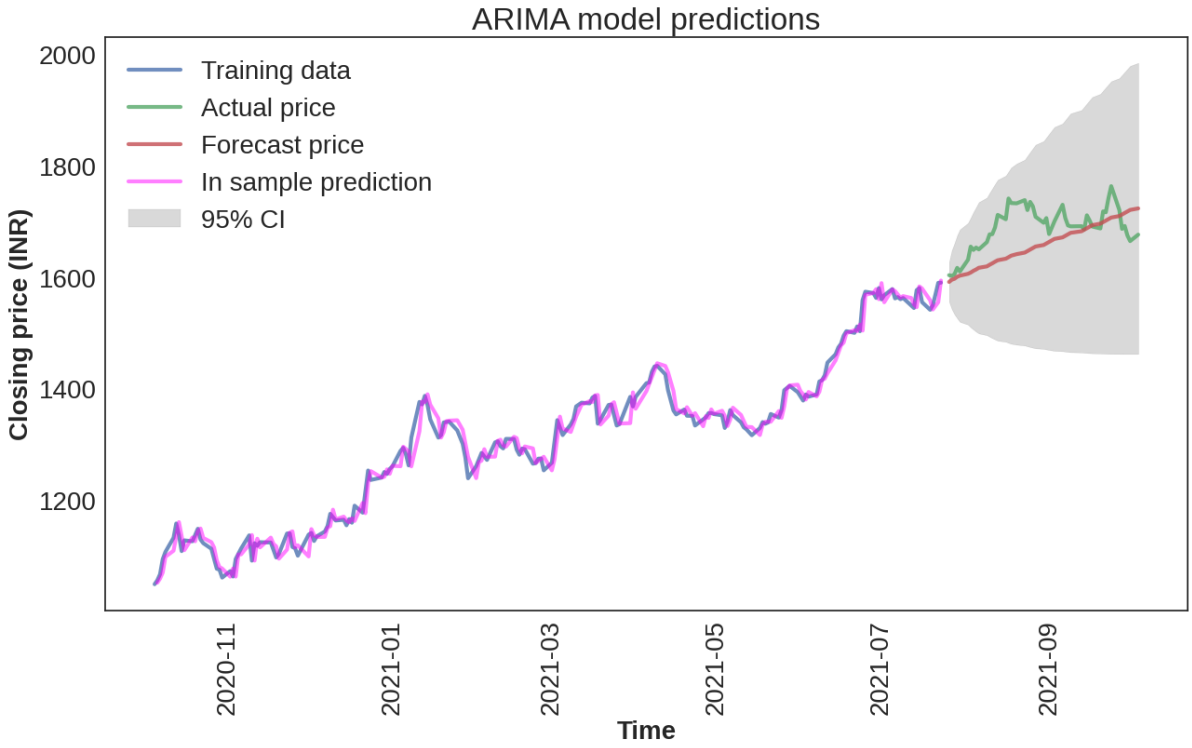
In [ ]:

```
plt.style.use('seaborn-v0_8-white')
model.plot_diagnostics(figsize=(16,8))
```

```
plt.show()
```



In [ ]:
```
#Auto-ARIMA model
plt.style.use('seaborn-v0_8-white')
op, conf_int = model.predict(len(test),return_conf_int=True)
lower_series = pd.Series(conf_int[:, 0], index=test.index)
upper_series = pd.Series(conf_int[:, 1], index=test.index)
fc_series = pd.Series(op, index=test.index)
predictions = model.predict_in_sample()
insam_series=pd.Series(predictions, index=train.index)
plt.figure(figsize=(15,8))
plt.plot(train, label='Training data',linewidth=3,alpha=0.8)
plt.plot(test, label='Actual price',linewidth=3,alpha=0.8)
plt.plot(fc_series, label='Forecast price',linewidth=3,alpha=0.8)
plt.plot(insam_series.loc['2020-10-06':],label='In sample prediction',color='magent
plt.title('ARIMA model predictions',fontsize=24)
plt.ylabel('Closing price (INR)',fontsize=20)
plt.xlabel('Time',fontsize=20)
plt.xticks(rotation=90,fontsize=20)
plt.yticks(fontsize=20)
plt.fill_between(lower_series.index, lower_series, upper_series, color='k', alpha=.
plt.legend(loc='upper left',fontsize=20)
plt.savefig('ARIMA prediction.png',bbox_inches='tight')
plt.show()
```



In [ ]:
```
mse(test,op)**0.5# get the mean squared error
```

Out[ ]:
```
48.203094479758676
```

In [ ]:
```
#Scale the data
S=StandardScaler()
scaled_train=S.fit_transform(train)
scaled_test=S.transform(test)
```

In [ ]:
```
#Use a 60 day data window for training in sequence
Test_set_to_prediction=np.concatenate([scaled_train[-3:],scaled_test],axis=0)
```

Generate the sequence data

In [ ]:
```
window_size = 3
def windowed_dataset(series,batch_size):
    dataset = tf.data.Dataset.from_tensor_slices(series.reshape(-1,1))

    # Window the data but only take those with the specified size
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)

    # Flatten the windows by putting its elements in a single batch
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))

    # Create tuples with features and labels
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))

    # Shuffle the windows
    #dataset = dataset.shuffle(shuffle_buffer)

    # Create batches of windows
    dataset = dataset.batch(len(series)).prefetch(1)

    return dataset
```

In [ ]:
```
X,y=np.array(list(windowed_dataset(scaled_train,10))[0][0]),np.array(list(windowed_
X_test,y_test=np.array(list(windowed_dataset(Test_set_to_prediction,10))[0][0]),np.
```

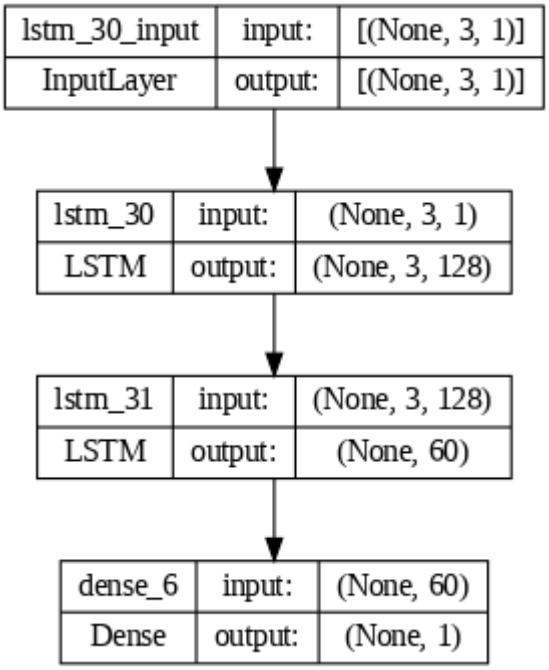
In [ ]:
```
X_test.shape
```

Out[ ]:
```
(49, 3, 1)
```

In [ ]:
```
Model1=Sequential(
[
    LSTM(units=128,input_shape=(X.shape[1],1),return_sequences=True),
    LSTM(units=60),
    Dense(units=1),
])
```

In [ ]:
```
#Visualize the model
plot_model(Model1, show_shapes = True,expand_nested = True,dpi = 80)
```

Out[ ]:



In [ ]:
```
Model1.compile(optimizer = "adam",loss='mean_squared_error')
Model1.summary()
```

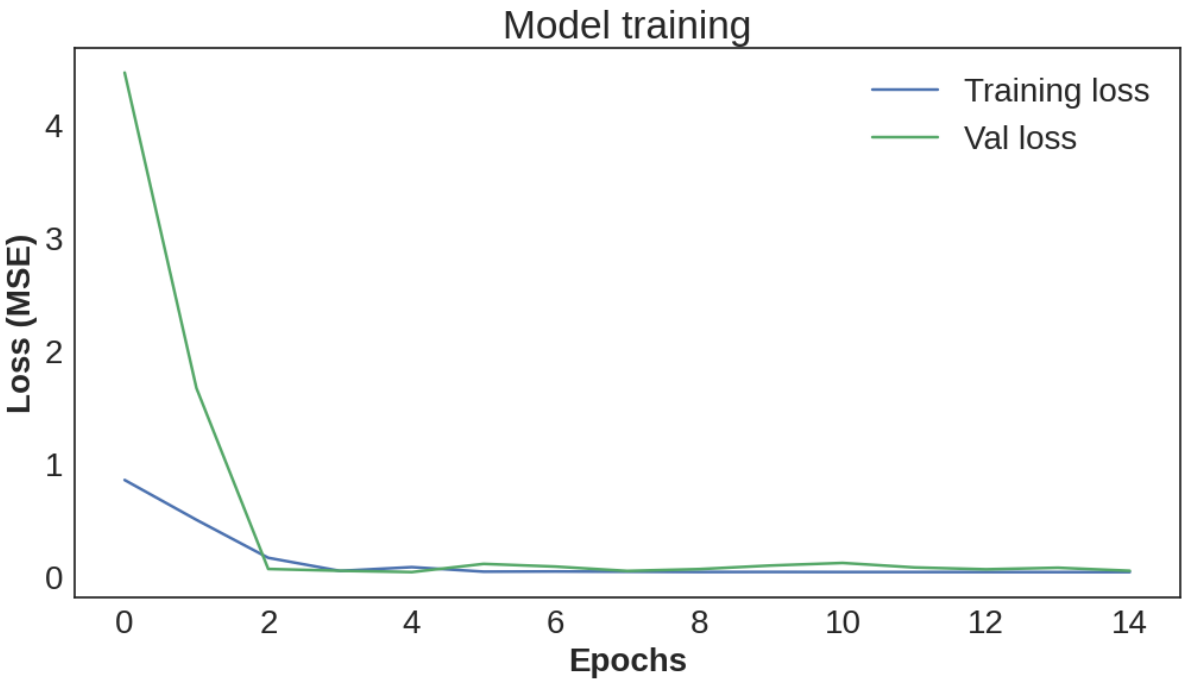
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_30 (LSTM)              (None, 3, 128)            66560

 lstm_31 (LSTM)              (None, 60)                45360

 dense_6 (Dense)             (None, 1)                 61

=================================================================
Total params: 111981 (437.43 KB)
Trainable params: 111981 (437.43 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:
```
callbacks=tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=10,
    restore_best_weights=True)
history=Model1.fit(X,y,epochs=300,validation_data=(X_test,y_test),callbacks=[callba
```

```
Epoch 1/300
7/7 - 8s - loss: 0.8528 - val_loss: 4.4647 - 8s/epoch - 1s/step
Epoch 2/300
7/7 - 0s - loss: 0.4997 - val_loss: 1.6676 - 86ms/epoch - 12ms/step
Epoch 3/300
7/7 - 0s - loss: 0.1629 - val_loss: 0.0640 - 99ms/epoch - 14ms/step
Epoch 4/300
7/7 - 0s - loss: 0.0469 - val_loss: 0.0480 - 102ms/epoch - 15ms/step
Epoch 5/300
7/7 - 0s - loss: 0.0802 - val_loss: 0.0364 - 91ms/epoch - 13ms/step
Epoch 6/300
7/7 - 0s - loss: 0.0403 - val_loss: 0.1090 - 82ms/epoch - 12ms/step
Epoch 7/300
7/7 - 0s - loss: 0.0418 - val_loss: 0.0852 - 90ms/epoch - 13ms/step
Epoch 8/300
7/7 - 0s - loss: 0.0397 - val_loss: 0.0466 - 104ms/epoch - 15ms/step
Epoch 9/300
7/7 - 0s - loss: 0.0371 - val_loss: 0.0627 - 93ms/epoch - 13ms/step
Epoch 10/300
7/7 - 0s - loss: 0.0376 - val_loss: 0.0946 - 92ms/epoch - 13ms/step
Epoch 11/300
7/7 - 0s - loss: 0.0369 - val_loss: 0.1174 - 99ms/epoch - 14ms/step
Epoch 12/300
7/7 - 0s - loss: 0.0368 - val_loss: 0.0776 - 100ms/epoch - 14ms/step
Epoch 13/300
7/7 - 0s - loss: 0.0366 - val_loss: 0.0611 - 79ms/epoch - 11ms/step
Epoch 14/300
7/7 - 0s - loss: 0.0361 - val_loss: 0.0748 - 86ms/epoch - 12ms/step
Epoch 15/300
7/7 - 0s - loss: 0.0362 - val_loss: 0.0479 - 95ms/epoch - 14ms/step
```

In [ ]:

```
plt.style.use('seaborn-v0_8-white')
plt.figure(figsize=(12,6))
plt.title('Model training',fontsize=24)
plt.plot(history.history['loss'],label='Training loss')
plt.plot(history.history['val_loss'],label='Val loss')
plt.ylabel('Loss (MSE)',fontsize=20)
plt.xlabel('Epochs',fontsize=20)
plt.legend(fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.savefig('Model training.png',bbox_inches='tight')
plt.show()
```



In [ ]:
```
predictions = Model1.predict(X_test)
predictions = S.inverse_transform(predictions)
train_pred=S.inverse_transform(Model1.predict(X))
```

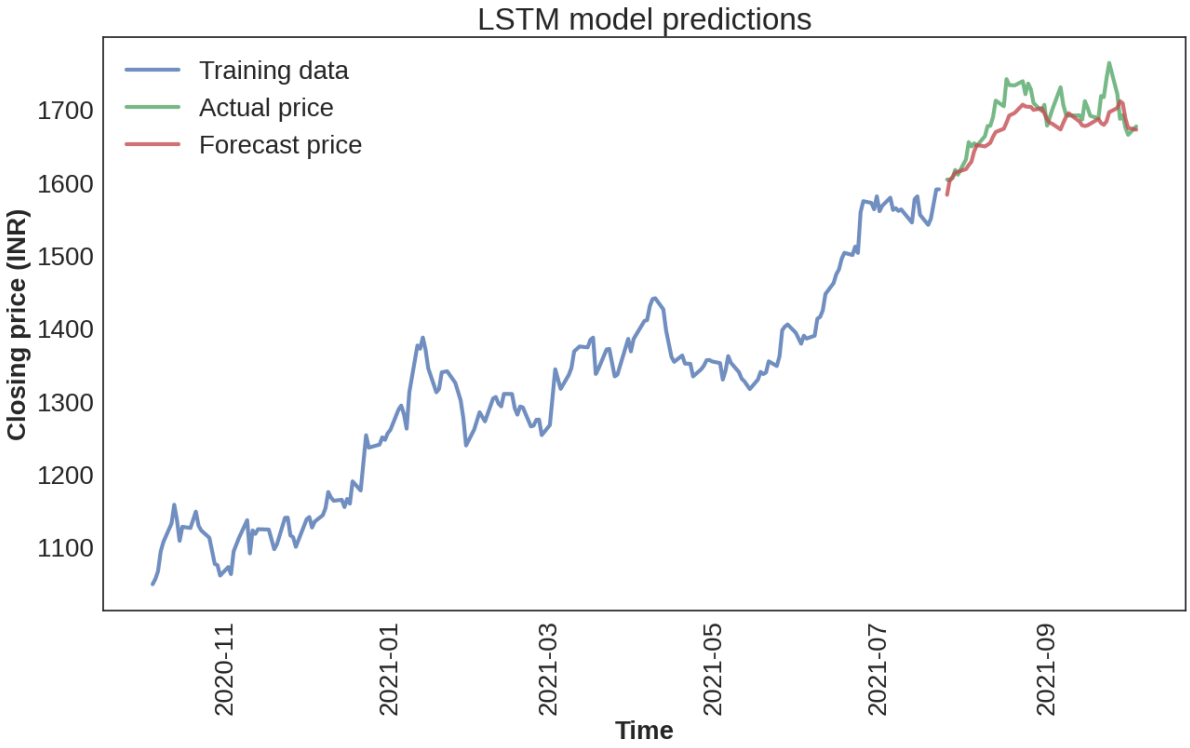
```
2/2 [==============================] - 0s 9ms/step
7/7 [==============================] - 0s 6ms/step
```

In [ ]:
```
mse(test,predictions)**0.5 # RMSE of LSTM prediction
```

Out[ ]:
```
27.091381416981854
```

In [ ]:
```
LSTM_pred_series=pd.Series(predictions.reshape(-1), index=test.index)
```

In [ ]:
```
plt.style.use('seaborn-v0_8-white')
plt.figure(figsize=(15,8))
plt.plot(train, label='Training data',linewidth=3,alpha=0.8)
plt.plot(test, label='Actual price',linewidth=3,alpha=0.8)
plt.plot(LSTM_pred_series, label='Forecast price',linewidth=3,alpha=0.8)
plt.title('LSTM model predictions',fontsize=24)
plt.ylabel('Closing price (INR)',fontsize=20)
plt.xlabel('Time',fontsize=20)
plt.xticks(rotation=90,fontsize=20)
plt.yticks(fontsize=20)
plt.legend(loc='upper left',fontsize=20)
plt.savefig('LSTM prediction.png',bbox_inches='tight')
plt.show()
```


LSTM model predictions
Training data
Actual price
Forecast price
Closing price (INR)
1700
1600
1500
1400
1300
1200
1100
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
Time

End of Code