SRI VENKATESWARA INTERNSHIP PROGRAM

FOR RESEARCH IN ACADEMICS

(SRI-VIPRA: 2412)

Project Report -2021

"STATISTICAL MODELLING: ANALYSIS AND PREDICTION OF TIME-DEPENDENT EVENTS"

Sri Venkateswara College

University of Delhi

Dhaula Kuan

New Delhi -110021

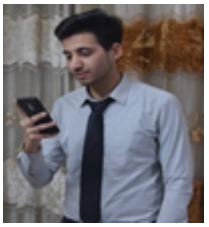
SRI VIPRA PROJECT 2021

Name of Mentor: Mr. Akash Varshney
Name of Department: Statistics
Designation: Assistant Professor

Name of Mentor: Dr. Dipika
Name of Department: Statistics
Designation: Assistant Professor

Title of the Paper— Statistical modelling: Analysis and Prediction of Time-Dependent Events

List of students under the SRI-VIPRA Project

S.No.	Name	Course	Picture
1.	Anuja Bharadwaj	Statistics (H)	
2.	Bhavya Batra	Statistics (H)	
3.	Riya Jindal	Statistics (H)	
4.	Aishwarya Srivastava	Statistics (H)	
5.	Saksham Joshi	Statistics (H)	

6.	Shameek Phukan	Statistics (H)	
7.	Khushi Panjabi	Statistics (H)	
8.	Shruti Vohra	Statistics (H)	
9.	Ananya Sinha	Statistics (H)	
10.	Navya Agarwal	Statistics (H)	
11.	Archisha Singh	Statistics (H)	

12.	Sunny Chaudhary	Statistics (H)	

Coordinator	**Mr. Akash Varshney**	**Dr. Dipika**
SRIVIPRA 2021	**Mentor**	**Mentor**
Sri Venkateswara College	**Department of Statistics**	**Department of Statistics**

CERTIFICATE

This is to certify that the aforementioned students from Sri Venkateswara College have participated in the summer project entitled "Statistical modelling: Analysis and prediction of time-dependent events**"**. The participants have carried out the research project work under our guidance and supervision from July 10, 2021, to August 25, 2020. The work carried out is original and was carried out in an online mode.

Mr. Akash Varshney

Mentor

Department of Statistics

Dr. Dipika

Mentor

Department of Statistics

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to our mentors Mr. Akash Varshney and Dr. Dipika, Department of Statistics, Sri Venkateswara College, University of Delhi, Delhi for providing us with the research problem and providing expert guidance and inspiration throughout our research project.

We are thankful to our Principal, Professor C. Sheela Reddy for creating this research opportunity, and to the Convener of SRI VIPRA -2021 for providing us with a platform to work, learn and progress.

Anuja Bhardwaj

Khushi Panjabi

Bhavya Batra

Shruti Vohra

Riya Jindal

Ananya Sinha

Aishwarya Srivastava

Navya Agarwal

Saksham Joshi

Archisha Singh

Shameek Phukan

Sunny Chaudhary

CONTENTS

SUMMARY

Financial modeling is the task of building an abstract representation (a model) of a real world financial situation. This is a statistical model designed to represent (a simplified version of) the performance of a financial asset or portfolio of a business, project, or any other investment.

In our project, we have considered NIFTY 50 (flagship index) data for the past 20 years and applied statistical modelling and Machine Learning techniques in order to predict the Closing Price as well as the volatility of the Closing Price for the next day. Since NIFTY 50 data is a time series data , we have carried out the decomposition of all the components of a time series into consideration with an aim to find out the hidden trends, seasonality and irregularities in the NIFTY 50 Data.

In order to predict the Closing Price and volatility for the Closing Price we have fitted ARIMA model and GARCH model respectively using the inbuilt libraries in Python. After getting satisfactory results on the test data, we finalized the parameters for both ARIMA and GARCH. Along with the mentioned statistical models we have used the Machine Learning algorithm LSTM as well so as to predict the future closing price. Also, in order to find out the relationship between the closing price of various popular indices, we have calculated the required correlation coefficients thus getting an idea about the extent to which two variables are related, describing simple relationships without making a statement about cause and effect. These relationships are then graphically represented.

Keywords: Stock Market, ARIMA, GARCH, LSTM, PYTHON

INTRODUCTION

The stock market, also known as the stock exchange or the share market, is an organized market for the purchase and sale of industrial and financial securities. It is a convenient place where trading in securities is conducted systematically per a set of rules and regulations. It is one of the most important constituents of capital markets and acts as an investment intermediary while facilitating economic and industrial development for countries.

The Indian Stock Market occupies a predominant place in the world's economy. It includes the National Stock Exchange (NSE) and the Bombay Stock Exchange (BSE). BSE is the oldest stock exchange, with Sensex as a major index. NSE has better technologies and advancements, with NIFTY 50 as a major index.

Singapore Nifty, which is more commonly known as SGX Nifty is a derivative market by Indian Nifty Index to be traded in the Singapore stock exchange. The movement of the SGX Nifty is highly correlated to the Nifty-50 Index. SGX Nifty opens at 6:30 AM (Indian Standard Time) daily and gives the direction of the Indian Stock Market.

The NIFTY Realty Index is designed to reflect the behaviour and performance of Real Estate companies. The Index comprises 10 realty companies listed on the National Stock Exchange of India (NSE).

"The NIFTY Realty Index is computed using a free float, market-capitalization-weighted method with a base date of December 29, 2006, and base value of 1000, wherein the level of the index reflects the total free float market value of all the stocks in the index relative to a particular base period. The method also takes into account constituent changes in the index and important corporate actions such as stock splits, rights, etc. without affecting the index value." (NSE INDIA, n.d.)

The Financial Times Stock Exchange 100 Index, also called the FTSE, is a share index of the 100 companies listed on the London Stock Exchange with (in principle) the highest market capitalization.

U.S. Treasury Bonds are government debt securities issued by the United States federal government that have maturity ranges between 10 and 30 years.

Over the years, different varieties of input vectors have been considered for stock market prediction, and have been used on the same dataset. Researchers have focused on inputs from time-series as well as heterogeneous market data. Preparing models and forecasting for financial time series data has been possible because of data mining, machine learning and deep learning algorithms.

Although these techniques play an important role in the prediction of time series data, many research works illustrate that data mining and machine learning techniques have displayed problems due to tremendous noise, non-stationary characteristics, hidden patterns and complex dimensionality of data. [1]

In New Zealand, the hedonic price model and artificial neural network (ANN) was studied for housing price prediction using multiple factors, some of which were house size, age and type, number of bedrooms and other amenities; the result showed satisfactory results for ANN to predict housing prices but the "black box" architecture of ANN limits their usefulness to practitioners in the field [2]. Correlation analysis and scatter plots were used to study the relationship between selected macroeconomic variables with Nifty Realty Index using annual data from 2006 to 2016. The results concluded that Inflation, Money Supply and Crude Oil Prices had a negative impact on Nifty Realty Stock Index while Interest Rates and Exchange Rates had a positive effect on Nifty Realty Stock Index. The performance of the realty sector in Sensex depends upon the economy and industry performance up to 50%. In an analysis of three stocks, namely, Unitech, DLF and Sobha Developers it was found that DLF Company is better off compared to the others as it offers good returns and is less volatile. The tools used for finding the volatility of return on stocks were Beta, Standard Deviation, Coefficient of Correlation and Alpha [3].

Artificial neural network models such as multilayer perceptions (MLPs), radial basis function networks (RBF), general regression neural networks (GRNNs), etc. are used in time-series forecasting of relationships between indices. Traditional methods, such as time-series regression, exponential smoothing and AutoRegressive Integrated Moving Average (ARIMA) are based on linear models. All these methods assume linear relationships among the past values of the forecast variable and therefore non-linear patterns cannot be captured by these models. Recently, Artificial Neural Networks (ANN) has been proposed as a promising alternative to time-series forecasting. Tiffany Hui- Kuang Yu, Kun-Huang Huarng noted that some successful applications of neural networks include credit ratings. The proposed model can be used to forecast directly regardless of whether out-of-sample observations appear in the in-sample observations. This study performs out-of-sample forecasting and the results are compared with those of previous studies to demonstrate the performance of the proposed model [4].

Short-term interest rates have the power to forecast short-term stock returns and risk premiums on observation of comments between stock and bond prices. This is reiterated by many empirical studies that have shown that the term "structure of nominal interest rates" contains information potentially useful for the conduct of monetary policy [5,6]. It is also found that the bond and stock prices in India had a bivariate causality in the year 2009 and univariate causality in 2010. The

results are interesting and support the view that excess volatility causes granger between the stock and bond markets. Unit root test, co-integration analysis and granger causality test were used to arrive at these results [7].

OBJECTIVE OF THE STUDY

1. Analyze Price and Value pattern of Nifty over last 20 Years

2. Forecast Price of Nifty using Time Series Model ARIMA

3. Measure and Forecast Volatility of return using GARCH Model

4. Forecast Nifty price using Artificial Intelligence and Machine Learning Models RNN and LSTM

5. Compare and Correlate between

    i. Nifty with SGX Nifty
    ii. Nifty and US Treasury Bonds
    iii. Nifty and FTSE
    iv. CII with NIFTY Realty

RESEARCH METHODOLOGY

For the study, we used open-source secondary data available on the stock market. The final data analysis was done using Python. The following concepts were used for developing our research methodology:

Time Series

A time series is a sequential set of data points, typically measured over successive times. A time series containing records of a single variable is termed univariate. But if records of more than one variable are considered, it is termed multivariate.

Components of a Time Series

A time series, in general, is supposed to be affected by four main components, which can be separated from the observed data. These components are:

1. Trend
2. Seasonal Variations
3. Cyclical Variations
4. Irregular component

Trend

The general tendency of a time series to increase, decrease or stagnate over a long period of time is termed as Secular Trend or simply Trend. Thus, it can be said that trend is a long-term movement in a time series. For example, series relating to population growth, number of houses in a city etc. show an upward trend, whereas a downward trend can be observed in series relating to mortality rates, epidemics, etc.

Seasonal Variations

Seasonal variations in a time series are fluctuations within a year, during a particular season. Factors like climate and weather conditions, customs, traditional habits, etc cause seasonal variations. For example, sales of ice cream increase in summers whereas sales of woollen clothes increase in winters. Seasonal variation is an important factor for businessmen, shopkeepers and producers for making proper future plans.

Cyclical Variations

The cyclical variation in a time series describes the medium-term changes in the series, caused by circumstances that repeat cyclically. The duration of a cycle extends over a longer period, usually two or more years. Most economic and financial time series show some kind of cyclical variation.

Correlation

Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It is a common tool for describing simple relationships without making statements about cause and effect.

Correlation Coefficient

The correlation coefficient, r, is a summary measure that describes the extent of the statistical relationship between two interval or ratio-level variables. The correlation coefficient is scaled so that it always lies between -1 and +1. When the value of r is close to 0, there is little relationship between the variables and the farther away from 0 the value is, in either the positive or negative direction, the greater is the relationship between the two variables.

Scatter Diagram

A scatter diagram is a diagram that shows the values of two variables X and Y, along with how these two variables relate to each other. The values of variable X are given along the horizontal axis, with the corresponding values of variable Y on the vertical axis.

Types of Correlation

When a regression model is used, one of the variables is defined as an independent variable and the other is defined as the dependent variable. In regression, the independent variable X is considered to have some effect or influence on the dependent variable Y. Correlation methods are symmetric with respect to the two variables, with no indication of causation or direction of influence being part of the statistical consideration.
The scatter plot explains the correlation between the two attributes or variables. It represents how closely the two variables are connected.

There can be three such situations to see the relation between the two variables —

i. *Positive Correlation*

When the values of the two variables move in the same direction, i.e., an increase/decrease in the value of one variable is followed by an increase/decrease in the value of the other variable.

ii. *Negative Correlation*

When the values of the two variables move in the opposite direction, i.e., an increase/decrease in the value of one variable is followed by a decrease/increase in the value of the other variable.

iii. *No Correlation*

When there is no linear dependence or no relation between the two variables.

Pearson Correlation Coefficient

The most common formula is the Pearson Correlation coefficient, used for linear dependency between the data sets. The value of the coefficient lies from -1 to +1. When the coefficient comes down to zero, then the data is considered not related. On the other hand, if we get a value greater than 0, then the data are positively correlated and a value less than 0 implies a negative correlation.

$$r = \frac{n\left(\sum xy\right) - \left(\sum x\right)\left(\sum y\right)}{\sqrt{\left[n\sum x^2 - \left(\sum x\right)^2\right]\left[n\sum y^2 - \left(\sum y\right)^2\right]}}$$

Where,

n = Quantity of Information

$\sum x$ *is Total of the First Variable Value*

$\sum y$ *is Total of the Second Variable Value*

$\sum xy$ *is Sum of the Product of first & Second Value*

$\sum x^2$ *is Sum of the Squares of the First Value*

$\sum y^2$ *is Sum of the Squares of the Second Value*

Spearman's Correlation Coefficient

Spearman rank correlation is a non-parametric test that is used to measure the degree of association between two variables. The Spearman rank correlation test does not carry any assumptions about the distribution of the data and is the appropriate correlation analysis when the variables are measured on a scale that is at least ordinal.

The following formula is used to calculate the Spearman rank correlation:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)}$$

$\rho = Spearman's\ rank\ correlation\ coefficient$

$d_i = difference\ between\ the\ two\ ranks\ of\ each\ observation$

$n = number\ of\ observations$

Autoregressive Integrated Moving Average Model (ARIMA)

ARIMA (p, d, q) forecasting equation: ARIMA models are, in theory, the most general class of models for forecasting a time series which can be made to be "stationary" by differencing (if necessary), perhaps in conjunction with nonlinear transformations such as logging or deflating (if necessary). A random variable that is a time series is stationary if its statistical properties are all constant over time. A stationary series has no trend, its variations around its mean have constant amplitude, and it wiggles in a consistent fashion, i.e., its short-term random time patterns always look the same in a statistical sense. The latter condition means that its autocorrelations (correlations with its own prior deviations from the mean) remain constant over time, or equivalently, that its power spectrum remains constant over time. A random variable of this form can be viewed (as usual) as a combination of signal and noise, and the signal (if one is apparent) could be a pattern of fast or slow mean reversion, or sinusoidal oscillation, or rapid alternation in sign, and it could also have a seasonal component. An ARIMA model can be viewed as a "filter" that tries to separate the signal from the noise, and the signal is then extrapolated into the future to obtain forecasts.

An ARIMA model is a class of statistical models for analyzing and forecasting time series data.

It explicitly caters to a suite of standard structures in time-series data, and as such provides a simple yet powerful method for making skilful time-series forecasts.

ARIMA is an acronym that stands for Autoregressive Integrated Moving Average. It is a generalization of the simpler Autoregressive Moving Average and adds the notion of integration.

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

i. *Autoregression (AR)*

A model that uses the dependent relationship between an observation and some number of lagged observations.

ii. *Integrated (I)*

The use of differencing of raw observations (e.g.: subtracting an observation from an observation at the previous time step) in order to make the time-series stationary.

iii. *Moving Average (MA)*

A model that uses the dependency between an observation and a residual error from a moving average model is applied to lagged observations.

Each of these components is explicitly specified in the model as a parameter. A standard notation is used of ARIMA (p, d, q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

The parameters of the ARIMA model are defined as follows:

p: The number of lag observations included in the model, also called the lag order.

d: The number of times that the raw observations are different, also called the degree of differencing.

q: The size of the moving average window, also called the order of moving average.

Seasonal Autoregressive Integrated Moving Average Model (SARIMA)

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component.

It adds three new hyperparameters to specify the Autoregression (AR), Differencing (I) and Moving average (MA) for the seasonal Component of the series, as well as an additional parameter for the period of the seasonality.

There are four seasonal elements that are not part of ARIMA that must be configured; they are:

P: Seasonal autoregressive order

D: Seasonal Difference order

Q: Seasonal Moving average order

m: The number of the time steps for a step for a single seasonal period

Together, the notation for a SARIMA model is specified as:

SARIMA (p, d, q) (P, D, Q)m

An instance of the SARIMAX class can be created by providing the training data and a host of model configuration parameters.

1. # specify training data

2. data = ...

3. # define model

4. model = SARIMAX (data, ...)

The implementation is called SARIMAX instead of SARIMA because the "X" addition to the method name means that the implementation also supports exogenous variables.

These are parallel time series variates that are not modelled directly via AR, I, or MA processes but are made available as a weighted input to the model.

GARCH

GARCH was developed in 1986 by Dr. Tim Bollerslev.
Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) is a statistical model that can be used to analyze several different types of financial data, for instance, macroeconomic data. Financial institutions typically use this model to estimate the volatility of returns of stocks, bonds, and market indices. They use the resulting information to help determine pricing and judge which assets will potentially provide higher returns, as well as to forecast the returns of current investments to help in their asset allocation, hedging, risk management, and portfolio optimization decisions.

GARCH models are used when the variance of the error term is not constant. That is, the error term is heteroskedastic. Heteroskedasticity describes the irregular pattern of variation of an error term, or variable, in a statistical model.

Essentially, wherever there is heteroskedasticity, observations do not conform to a linear pattern. Instead, they tend to cluster. Therefore, if statistical models that assume constant variance are used on this data, then the conclusions and predictive value one can draw from the model will not be reliable.

The variance of the error term in GARCH models is assumed to vary systematically, conditional on the average size of the error terms in previous periods. In other words, it has conditional heteroskedasticity, and the reason for the heteroskedasticity is that the error term is following an autoregressive moving average pattern. This means that it is a function of an average of its own past values.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of RNN that similarly learns a mapping from input to output over time, but the mapping is no longer fixed as in the standard RNN. In backpropagation, one problem that arises is vanishing and exploding gradients (which result in poor results stemming from divide-by-zero errors). To resolve this, LSTMs use "computational gates" to control information flow over time, allowing them to forget less useful historical information. Hence, LSTMs do better with long-term dependencies.
An LSTM gate takes as input the previous hidden state and current input and returns a new state. These gates are of the form $Gt = g(WG \cdot [ht-1, xt] + bG)$, where activation function g will be either tanh or a sigmoid function σ. The input gate decides which values to update, the forget gate decides which values to forget, and the output gate decides which values to output.

First, the LSTM uses the forget gate ft by passing the previous hidden state and current input through a sigmoid function, which tells the LSTM which information to keep in the cell state Ct. Next, an input gate layer (another sigmoid layer) decides the values to be updated. The new values to be updated in the cell state are found with a tanh layer Ct. Then, we update the cell state so that $Ct \leftarrow - ft * Ct-1 + it * Ct$. Finally, we filter outputs using the output gate ot(a sigmoid layer) and finally return $ht = ot * \tanh(Ct)$. These gates are portrayed in the middle module of the figure below.

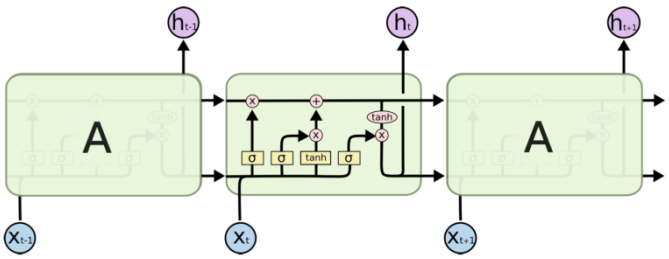


Figure: 1

LSTMs can be interpreted as a general form of another sub-type of RNN, the Gated Recurrent Unit (GRU). The GRU only has an input gate and a reset gate, the latter of which similarly selectively forgets historical information. For sequence-to-sequence modelings, usually, one RNN layer functions as an "encoder" while another functions as a "decoder." The "encoder" processes input and creates an internal state; the internal state is then fed into the "decoder," which predicts the output sequence. In Figure 2, we see that LSTMs can function as encoders and decoders to translate linguistic sequences; character sequences can easily be substituted for numerical values.

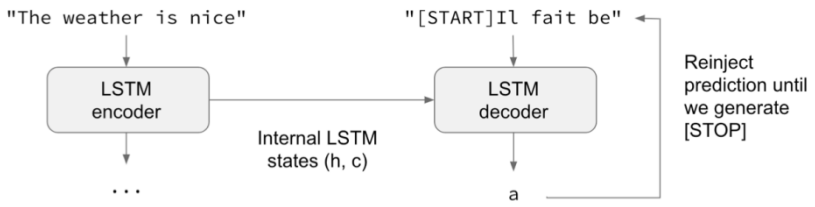


Figure: 2

DATA ANALYSIS

Plotting the time-series data:

```
plt.figure(figsize=(
22,7))plt.plot(df["Close"])pl
t.title("Nifty50",fontsize=25
)plt.xlabel("Vear",fontsize=1
S)
plt.ylabel("Closing
Price",fontsize=lS)pH.show()
```

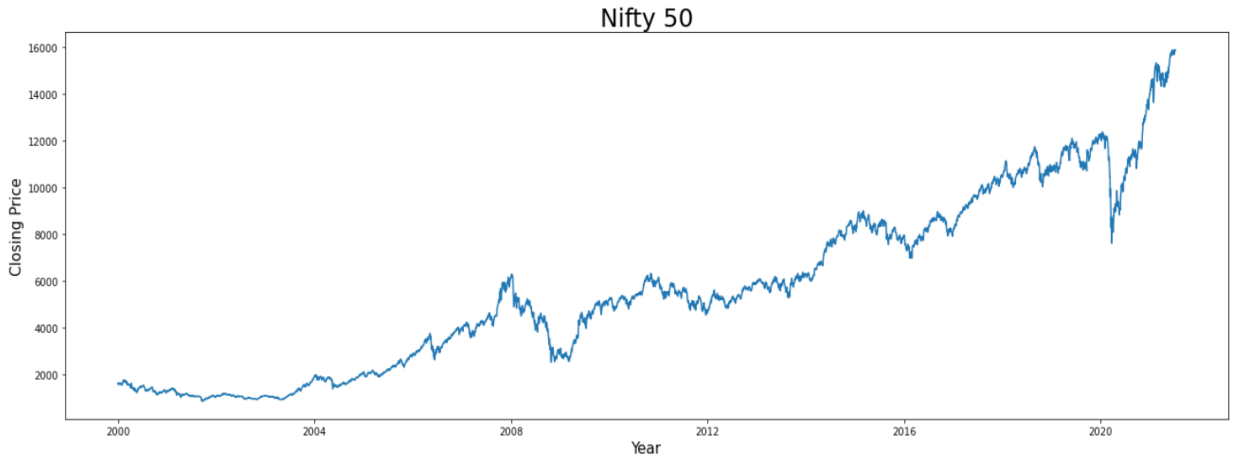


Figure: 3

The graph shows an increasing linear trend, thus decomposing the time series which will help us to get a trend curve.

Decomposition into components

Trend Seasonal Random

Date	Open	High	Low	Close	Volume	Turnover	P/E	P/B	Div Yield
1/3/2000	1482.15	1592.90	1482.15	1592.2	25358322	8.841500e+09	25.91	4.63	0.95
1/4/2000	1594.40	1641.95	1594.40	1638.7	38787872	1.973690e+10	26.67	4.76	0.92
1/5/2000	1634.5	1635.	1555.	1595.	621534	3.084790e	25.9	4.64	0.95

	5	50	05	8	31	+10	7		
1/6/2000	1595.80	1639.00	1595.80	1617.6	51272875	2.531180e+10	26.32	4.70	0.94
1/7/2000	1616.60	1628.25	1597.20	1613.3	54315945	1.914630e+1O	26.25	4.69	0.94

```
fromstatsmodels.tsa.seasonalimportseasonal_decompose

result=seasonal_decompose(df["Close"].resample("M").sum(),model='multiplic
ative')
```

```
from pylab import
rcParamsrcParams['figure.figsize'
]=12,5result.plot();
```

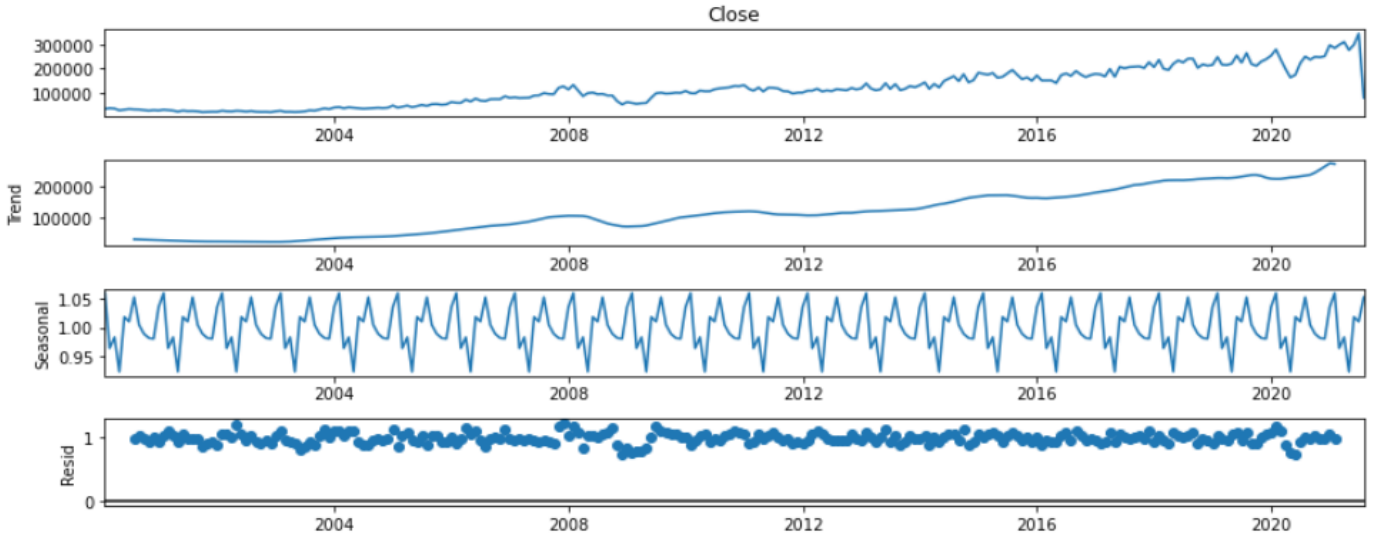


Figure: 4

Thus, we get graphs for each component; let us plot them separately so as to get a better view.

Plotting Trend graph for TS data:

```
plt.figure(figsiz=e(22,7))
plt.xlabel(
"Year",fontsize=15)plt.ylabel("Cl
osingPrice",fontsize=15)
plt.title(
"Trend",fontsize=20)result.trend.
plot()
```

```
<AxesSubplot:title={'center':'Trend'},xlabel='Date',ylabel='ClosingPrice'>
```

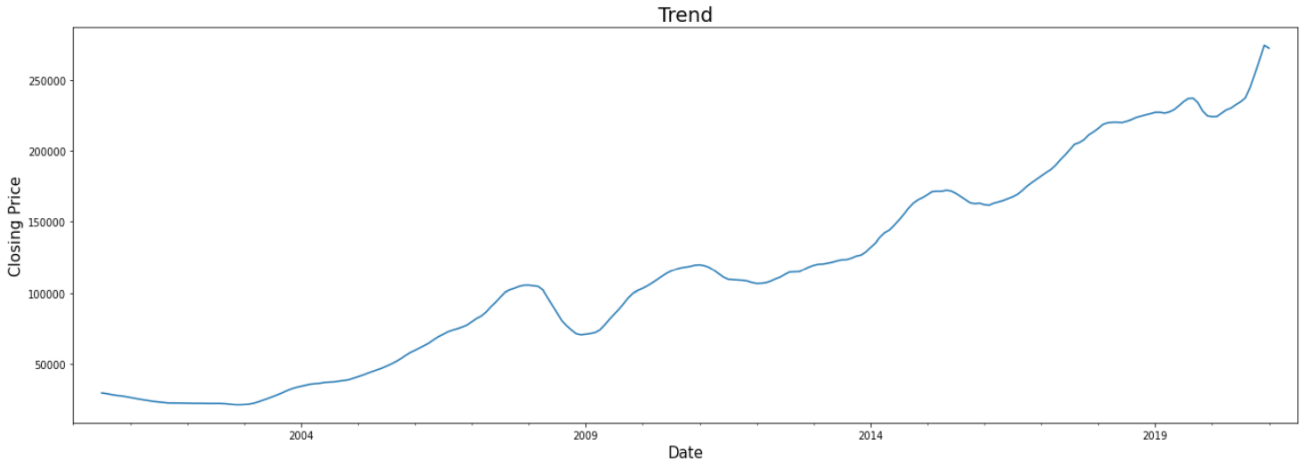


Figure: 5

Thus, there has been an increasing trend in the graph, i.e., closing stock prices have increased over the time period consistently.

Plotting Seasonality Graph:

```
plt.figure(figsize=(22,7))
plt.xlabel(
"Vear",fontsize=15)plt.ylabel("Clos
ingPrice",fontsize=15) plt.title(
"Seasonality",fontsize=20)result.se
asonal.plot()
<AxesSubplot:title={'center':'Seasonality',xlabel='Date',ylabel='ClosingPrice'>
```

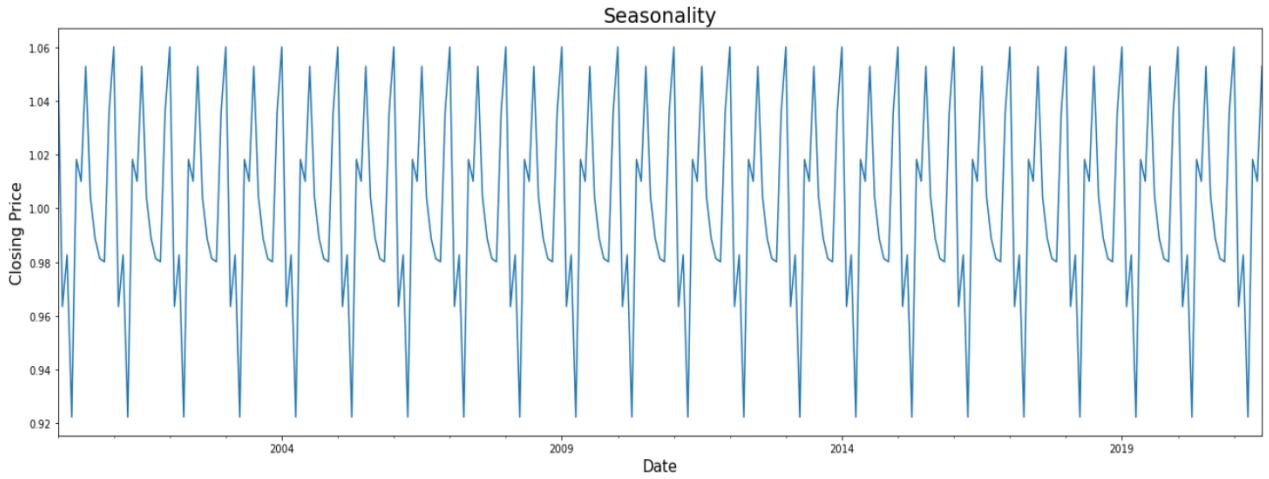


Figure: 6

The graph shows seasonality consistently over the time period by having a look at the graph. Seasonality seems to be following a yearly pattern.

Plotting random component:

```
plt.figure(figsize=(22,7))
plt.xlabel("Year",fontsize=15 )
plt.ylabel("Closing Price",fontsize=15)
plt.title("Random Component",fontsize=20)result.resid.plot()
```

```
<AxesSubplot:title={'center':'RandomComponent'},xlabel='Date',ylabel='Closi
ngPrice'>
```

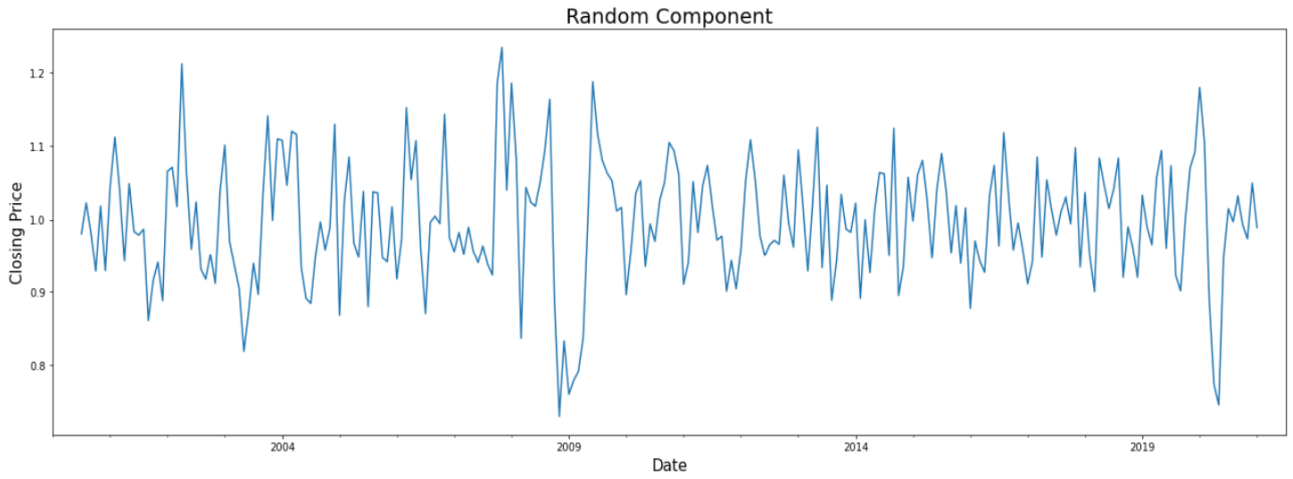


Figure: 7

The graph appears to be stable since the year-2000 and it will remain pretty much the same till 2020.

We observe fluctuations in the graph in the years 2008 and 2020 due to the US Housing Crisis in 2008 and the random components of the Covid-19pandemic in 2020, which made the stock market unstable.

Trend and Seasonal values in a tabular format:

TREND

```
pd.DataFrame(result.trend).head(20)
```

Date	trend
2000-01-31	NaN
2000-02-29	NaN
2000-03-31	NaN
2000-04-30	NaN
2000-05-31	NaN
2000-06-30	NaN
2000-07-31	29401.029167
2000-08-31	28935.904167
2000-09-30	28261.114583
2000-10-31	27700.879167
2000-11-30	27342.487500
2000-12-31	26840.037500
2001-01-31	26201.575000
2001-02-28	25622.135417
2001-03-31	24967.620833
2001-04-30	24399.370833
2001-05-31	23905.131250
2001-06-30	23403.154167
2001-07-31	23013.387500
2001-08-31	22653.291667

SEASONAL VALUES

```
pd.DataFrame(result.seasonal)
```

Date	seasonal
2000-01-31	1.060109
2000-02-29	0.963447
2000-03-31	0.982651
2000-04-30	0.922300
2000-05-31	1.018222
...	...
2021-03-31	0.982651
2021-04-30	0.922300
2021-05-31	1.018222
2021-06-30	1.010077
2021-07-31	1.052866

Checking whether the decomposition is correct or not.

Since we used a multiplicative model, thus

Observed Value=TrendVal x SeasonalVal x RandomComp x Cyclic(Not in our case)

Thus, storing trend val in a, seasonal in b, random in c, and d is the observed value.

```
a=result.trend[
20]b=result.seasonal[20]c=res
ult.resid[
20]d=result.observed[20]
```

```
a*b*c
```

18988.600000000002

```
a=result.trend[
20]b=result.seasonal[20]c=res
ult.resid[
20]d=result.observed[20]
```

18988.600000000002

Since d= a*b*c thus we verified our decomposition

De-Trending the series:

```
de_trend=pd.DataFrame(result.observed/result.trend)
```

Date	0
2000-01-31	NaN
2000-02-29	NaN
2000-03-31	NaN
2000-04-30	NaN
2000-05-31	NaN
2000-06-30	NaN
2000-07-31	1.032294
2000-08-31	1.027124
2000-09-30	0.970429
2000-10-31	0.910930
2000-11-30	0.998186
2000-12-31	0.962312
2001-01-31	1.105775
2001-02-28	1.070877
2001-03-31	1.021477
2001-04-30	0.869361
2001-05-31	1.067037
2001-06-30	0.993467
2001-07-31	1.030509
2001-08-31	0.990991

```
de_trend.head(20)
```

Date	Values after DeTrend
2000-01-31	NaN
2000-02-29	NaN
2000-03-31	NaN
2000-04-30	NaN
2000-05-31	NaN
2000-06-30	NaN
2000-07-31	1.032294
2000-08-31	1.027124
2000-09-30	0.970429
2000-10-31	0.910930
2000-11-30	0.998186
2000-12-31	0.962312
2001-01-31	1.105775
2001-02-28	1.070877
2001-03-31	1.021477
2001-04-30	0.869361
2001-05-31	1.067037
2001-06-30	0.993467
2001-07-31	1.030509
2001-08-31	0.990991

Now plotting the DeTrend Values and checking if the trend has been removed or not.

```
 plt.figure(figsize=(
22,7))plt.xlabel("Year",fontsi
ze=15
)plt.ylabel("DeTrendValues",fo
ntsize=15)
plt.title("GraphofSeriesafterDeTrending",fo
ntsize=25)plt.plot(de_trend)
```

```
[<matplotlib.lines.Line2Dat0x23ed23d4b20>]
```

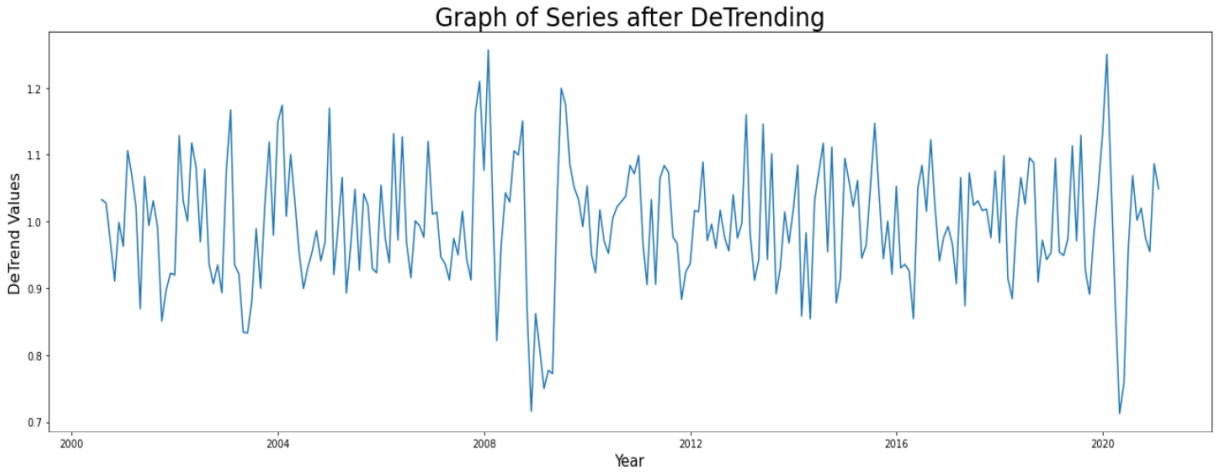


Figure: 8

We observe that the trend from the time series has been removed successfully.

ARIMA:

```
importpandasaspd
                                    importnumpyasnp
importmatplotlib.pyplotasplt
%matplotlib inline
plt.style.use("seaborn-darkgrid")
                                            df =
pd.read_csv(r'C:\Users\lenovo\Desktop\anuja\closediff.csv',index_col=[0],parse_dat
                                  es=[0]) df=df.dropna()
print('Shape of data',df.shape)
df.head()
df
```

```
Shape of data (5353, 1)
```

DATE	CLOSE
2000-03-01	1592.20
2000-04-01	1638.70
2000-05-01	1595.80
2000-06-01	1617.60
2000-07-01	1613.30
...	...
2021-01-07	15680.00
2021-02-07	15722.20
2021-05-07	15834.35
2021-06-07	15818.25
2021-07-07	15879.65

5353 rows × 1 columns

```
#after importing the data, it is date indexed.
```

```
df['CLOSE'].plot(figsize=(12,5))
```

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

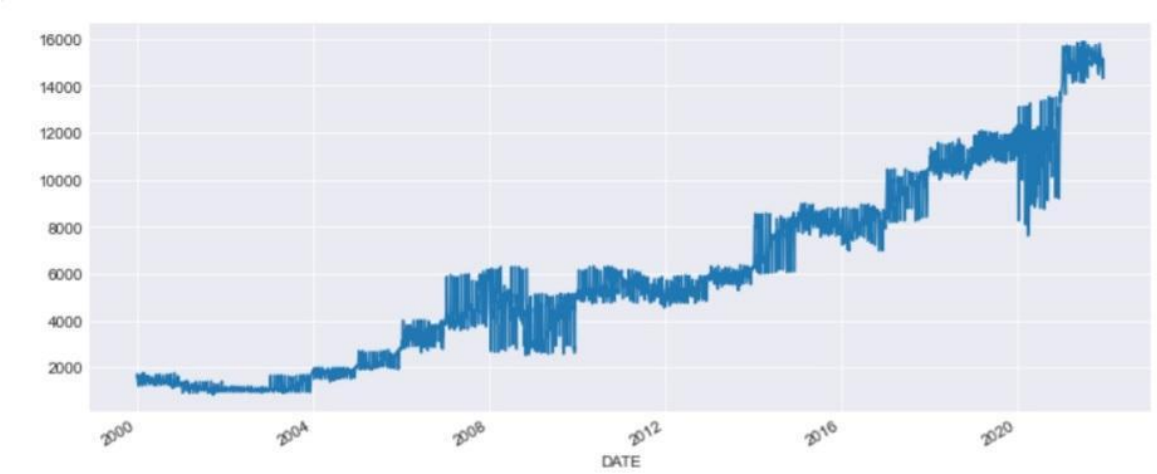


Figure: 9

```
fromstatsmodels.tsa.stattoolsimport adfuller
defad_test(dataset):
 dftest = adfuller(dataset, autolag ='AIC')
print("1. ADF : ",dftest[0])
print("2. P-Value : ", dftest[1])
print("3. Num Of Lags : ", dftest[2])
print("4. Num Of Observations Used For ADF Regression:", dftest[3])
print("5. Critical Values :")
for key, val in dftest[4].items():
print("\t",key, ": ", val)
```

```
:#applying the ADF Test to check the stationarity
ad_test(df['CLOSE'])
```

```
 1. ADF : 0.9959169554366316
 2. P-Value : 0.9942226803290629
 3. Num Of Lags : 18
 4. Num Of Observations Used For ADF Regression: 5334
 5. Critical Values :
1% : -3.4315765560130287
5% : -2.8620820125952537
10% : -2.567058512677479
```

```
#since the p value is greater than the level of significance, alpha, we
accept the null hypothesis which says that the data is
#stationary in nature.
```

```
# Change for (t)th day is Close for (t)th day minus Close for (t-1)th day.
df['Difference'] = df['CLOSE'].diff()
# Plot the Change
plt.figure(figsize=(22, 7))
plt.plot(df['Difference'])
plt.title('First Order Differenced Series', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Difference', fontsize=12)
plt.show()
```

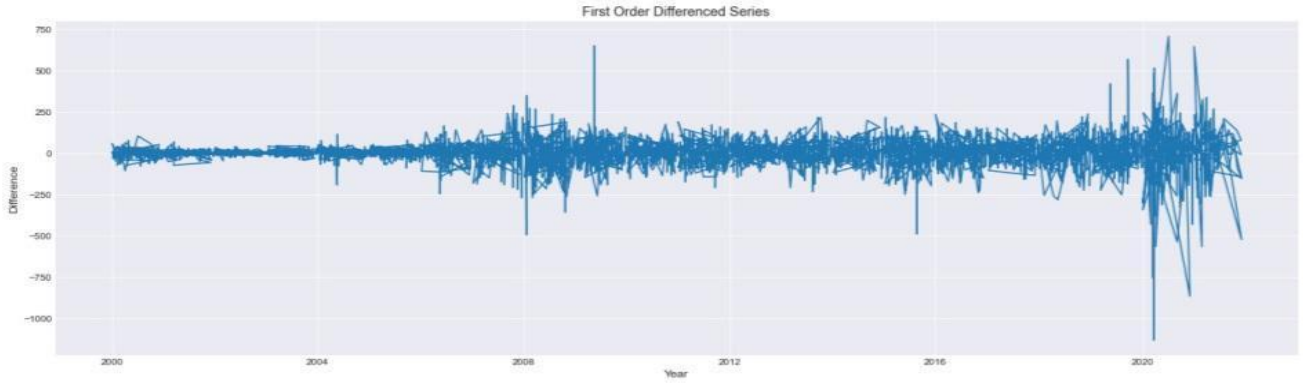


Figure: 10

```
#applying the ADF Test on the differenced series to check the stationarity
after differencing
result = adfuller(df.Difference.dropna())
print(result)
print('ADF Test Statistic: %.2f'% result[0])
print('5%% Critical Value: %.2f'% result[4]['5%'])
print('p-value: %.2f'% result[1])
```

```
(-16.165932722118374, 4.4157929599849096e-29, 17, 5334, {'1%':
-     3.4315765560130287, '5%': -2.8620820125952537, '10%':
-2.567058512677479},   61426.356274841564) ADF Test Statistic:
-16.17
5% Critical Value: -2.86
p-value: 0.00
```

```
#since now the p value is less than the level of significance, we reject
the null hypothesis.
```

ACF and PACF:

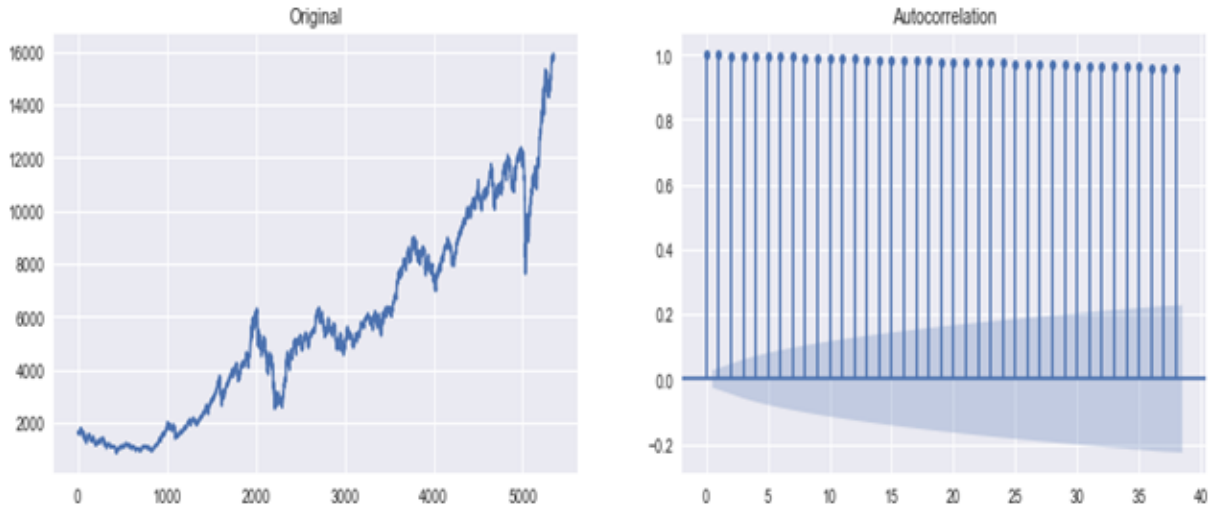


Figure: 11

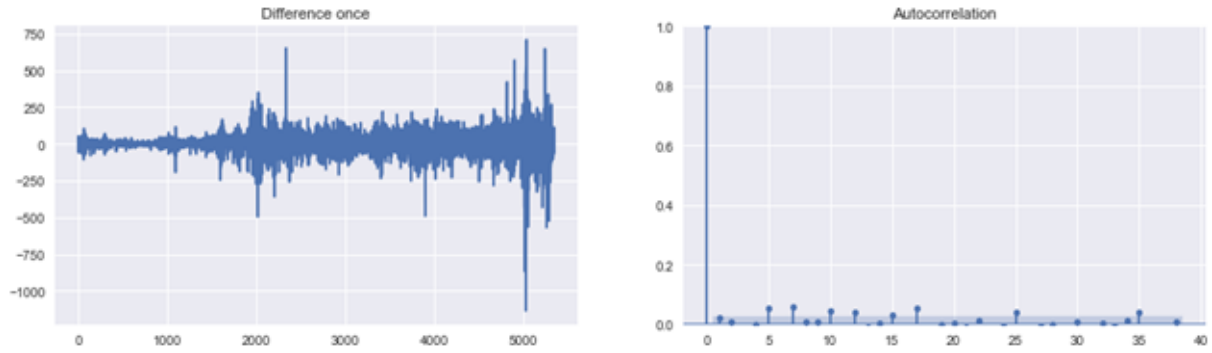


Figure: 12

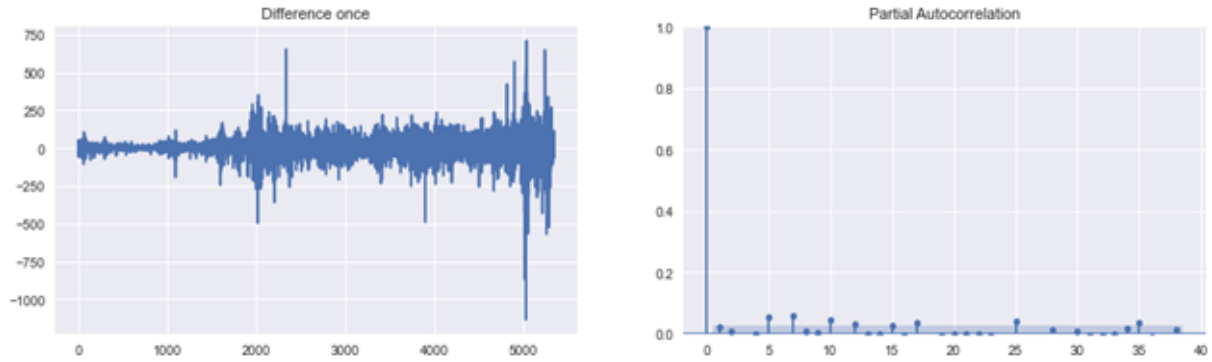


Figure: 13

We can observe that the PACF lag 5 is significant as it is above the significance line.

```
#installed pmdarima from pip
frompmdarimaimport auto_arima
#ignore harmless warnings
importwarnings
warnings.filterwarnings("ignore")
```

```
#applying auto arima function
stepwise_fit = auto_arima(df['CLOSE'], trace=True,
suppress_warnings=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept : AIC=61856.903, Time=12.76
sec
 ARIMA(0,1,0)(0,0,0)[0] intercept : AIC=61883.252, Time=0.22
sec
 ARIMA(1,1,0)(0,0,0)[0] intercept : AIC=61882.253, Time=0.47
sec
 ARIMA(0,1,1)(0,0,0)[0] intercept : AIC=61882.318, Time=0.67
sec
 ARIMA(0,1,0)(0,0,0)[0]           : AIC=61887.452, Time=0.12
sec
 ARIMA(1,1,2)(0,0,0)[0] intercept : AIC=61885.490, Time=2.64
sec
 ARIMA(2,1,1)(0,0,0)[0] intercept : AIC=61885.729, Time=1.38
sec
 ARIMA(3,1,2)(0,0,0)[0] intercept : AIC=61882.592, Time=14.31
sec
 ARIMA(2,1,3)(0,0,0)[0] intercept : AIC=61798.864, Time=14.30
sec
 ARIMA(1,1,3)(0,0,0)[0] intercept : AIC=61877.798, Time=10.25
sec
 ARIMA(3,1,3)(0,0,0)[0] intercept : AIC=61790.678, Time=17.33
sec
 ARIMA(4,1,3)(0,0,0)[0] intercept : AIC=61789.748, Time=16.59
sec
 ARIMA(4,1,2)(0,0,0)[0] intercept : AIC=61878.003, Time=17.25
sec
 ARIMA(5,1,3)(0,0,0)[0] intercept : AIC=61787.077, Time=21.19
sec
 ARIMA(5,1,2)(0,0,0)[0] intercept : AIC=61786.627, Time=18.15
sec
 ARIMA(5,1,1)(0,0,0)[0] intercept : AIC=61838.947, Time=8.88
sec
 ARIMA(4,1,1)(0,0,0)[0] intercept : AIC=61888.533, Time=1.93
sec
```

```
 ARIMA(5,1,2)(0,0,0)[0]             : AIC=61788.119, Time=8.68
sec

Best model: ARIMA(5,1,2)(0,0,0)[0] intercept
Total fit time: 168.207 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	5353
Model:	SARIMAX(5, 1, 2)	**Log Likelihood**	-30884.313
Date:	Fri, 30 Jul 2021	**AIC**	61786.627
Time:	15:22:47	**BIC**	61845.894
Sample:	0	**HQIC**	61807.327
	- 5353		
Covariance Type:	opg		

Covariance Type:	opg				
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	3.0726	3.591	0.856	0.392	-3.966	10.111
ar.L1	-1.3584	0.024	-55.756	0.000	-1.406	-1.311
ar.L2	-0.7293	0.021	-34.003	0.000	-0.771	-0.687
ar.L3	0.0188	0.012	1.564	0.118	-0.005	0.042
ar.L4	-0.0198	0.011	-1.755	0.079	-0.042	0.002
ar.L5	0.0384	0.007	5.163	0.000	0.024	0.053
ma.L1	1.3986	0.023	61.803	0.000	1.354	1.443
ma.L2	0.7902	0.018	43.434	0.000	0.755	0.826
sigma2	5998.3241	44.408	135.074	0.000	5911.287	6085.362

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	72753.41
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	13.53	**Skew:**	-0.78
Prob(H) (two-sided):	0.00	**Kurtosis:**	20.99

```
#Hence, the ebst fit model here would be (5,1,2) since it has the lowest AIC
and BIC compared to all others
```

```
fromstatsmodels.tsa.arima_modelimport ARIMA
```

```
print(df.shape)
train=df.iloc[:-30]
test=df.iloc[-30:]
print(train.shape,test.shape)
#selecting last 30 observations as the test set and the remaining as the train set.
```

(5353, 2)

(5323, 2) (30, 2)

```
model = ARIMA(train['CLOSE'],order = (5,1,2))
model = model.fit()
model.summary()
```

ARIMA Model Results

Dep. Variable:	D.CLOSE	**No. Observations:**	5322
Model:	ARIMA(5, 1, 2)	**Log Likelihood**	-30711.229
Method:	css-mle	**S.D. of innovations**	77.607
Date:	Fri, 30 Jul 2021	**AIC**	61440.458
Time:	15:26:50	**BIC**	61499.675
Sample:	1	**HQIC**	61461.147

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.5804	1.110	2.325	0.020	0.405	4.756
ar.L1.D.CLOSE	-1.3575	0.043	-31.477	0.000	-1.442	-1.273
ar.L2.D.CLOSE	-0.7344	0.044	-16.662	0.000	-0.821	-0.648
ar.L3.D.CLOSE	0.0163	0.025	0.647	0.517	-0.033	0.066
ar.L4.D.CLOSE	-0.0214	0.024	-0.893	0.372	-0.068	0.026
ar.L5.D.CLOSE	0.0383	0.016	2.343	0.019	0.006	0.070
ma.L1.D.CLOSE	1.3971	0.041	34.005	0.000	1.317	1.478
ma.L2.D.CLOSE	0.7945	0.039	20.332	0.000	0.718	0.871

```
5322    15316.936966
5323    15333.273351
5324    15309.763030
5325    15336.245715
5326    15328.950841
5327    15327.155659
5328    15344.404049
5329    15328.614471
5330    15346.414132
5331    15341.781191
5332    15342.194420
5333    15354.217725
5334    15344.424153
5335    15357.568452
5336    15354.820727
5337    15356.388189
5338    15365.055641
5339    15359.329890
5340    15369.217094
5341    15367.895942
5342    15370.101437
5343    15376.586289
5344    15373.603554
5345    15381.225312
5346    15380.970219
5347    15383.508324
5348    15388.579432
5349    15387.442480
5350    15393.492793
5351    15394.025913
```

```
DATE
2021-05-27    15316.936966
2021-05-28    15333.273351
2021-05-31    15309.763030
2021-01-06    15336.245715
2021-02-06    15328.950841
2021-03-06    15327.155659
2021-04-06    15344.404049
2021-07-06    15328.614471
2021-08-06    15346.414132
2021-09-06    15341.781191
2021-10-06    15342.194420
2021-11-06    15354.217725
2021-06-14    15344.424153
2021-06-15    15357.568452
2021-06-16    15354.820727
2021-06-17    15356.388189
2021-06-18    15365.055641
2021-06-21    15359.329890
2021-06-22    15369.217094
2021-06-23    15367.895942
2021-06-24    15370.101437
2021-06-25    15376.586289
2021-06-28    15373.603554
2021-06-29    15381.225312
2021-06-30    15380.970219
2021-01-07    15383.508324
2021-02-07    15388.579432
2021-05-07    15387.442480
2021-06-07    15393.492793
2021-07-07    15394.025913
dtype: float64
```

```
#in order to evaluvate our model efficiency, we use the metric RMSE
fromsklearn.metricsimport mean_squared_error
frommathimport sqrt
test['CLOSE'].mean()
rmse=sqrt(mean_squared_error(pred,test['CLOSE']))
print(rmse)
```

```
test['CLOSE'].mean()
```

15714.388333333334

```
plt.figure(figsize=(22,7))
plt.plot(test_df,label="ActualValues")
plt.plot(pred_df,color="r",label="PredictedValues")
plt.ylim([14000,21000])
plt.legend()
plt.show()
```

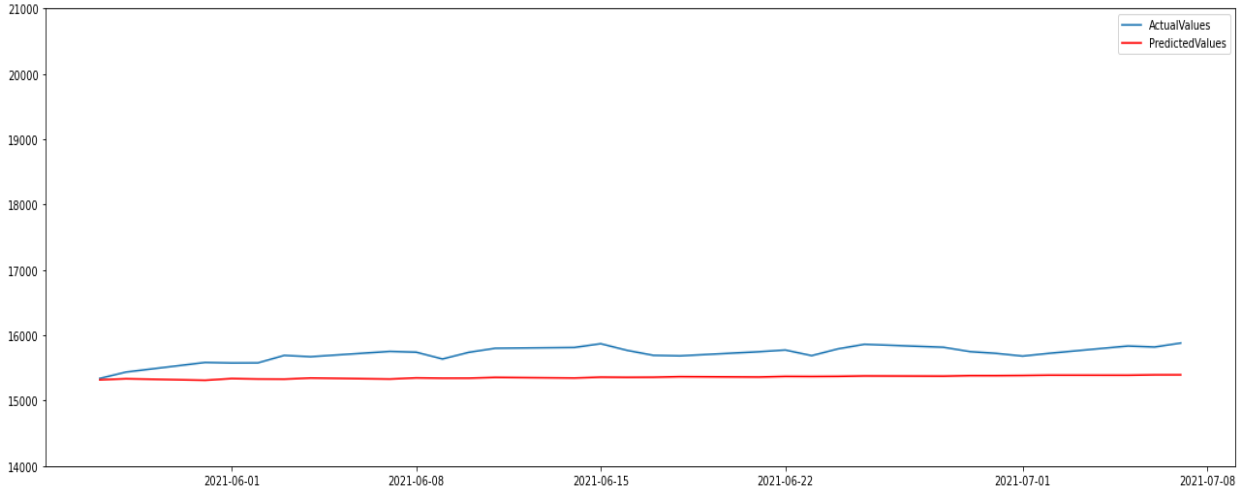


Figure: 14

```
actual_vs_predicted = pd.concat([test['CLOSE'],pred],axis =1)
actual_vs_predicted.columns = ["ACTUAL", "PREDICTED"]
actual_vs_predicted
```

DATE	ACTUAL	PREDICTED
2021-05-27	15337.85	15316.936966
2021-05-28	15435.65	15333.273351
2021-05-31	15582.80	15309.763030
2021-01-06	15574.85	15336.245715
2021-02-06	15576.20	15328.950841
2021-03-06	15690.35	15327.155659
2021-04-06	15670.25	15344.404049
2021-07-06	15751.65	15328.614471
2021-08-06	15740.10	15346.414132
2021-09-06	15635.35	15341.781191
2021-10-06	15737.75	15342.194420
2021-11-06	15799.35	15354.217725
2021-06-14	15811.85	15344.424153
2021-06-15	15869.25	15357.568452
2021-06-16	15767.55	15354.820727
2021-06-17	15691.40	15356.388189
2021-06-18	15683.35	15365.055641
2021-06-21	15746.50	15359.329890
2021-06-22	15772.75	15369.217094
2021-06-23	15686.95	15367.895942
2021-06-24	15790.45	15370.101437
2021-06-25	15860.35	15376.586289
2021-06-28	15814.70	15373.603554
2021-06-29	15748.45	15381.225312
2021-06-30	15721.50	15380.970219
2021-01-07	15680.00	15383.508324
2021-02-07	15722.20	15388.579432
2021-05-07	15834.35	15387.442480
2021-06-07	15818.25	15393.492793
2021-07-07	15879.65	15394.025913

```
> forecast = model_fit.forecast()[0]
forecast
```

```
array([15316.93639429])
```

The forecasted value for the next day is 15316.9364.

```
> from sklearn.metrics import mean_absolute_error as mae
MAE= mae(pred_df,test_df)
print(MAE)
```

```
357.40488999922735
```

The mean square error comes out to be 357.4049

GARCH MODEL:

```
returns=100*df.Close.pct_change().dropna()
plt.figure(figsize=(20,4))
plt.plot(returns)
```

Figure: 15

The graph above shows the Volatility of the NIFTY-50 index over the last 20 years.

Garch(1,1)

```
garch_1_0=arch_model(returns,p=1,q=0)
garch_1_0_fit=garch_1_0.fit()
```

```
Iteration:      1,   Func. Count:      5,   Neg. LLF: 29165.14626118768
Iteration:      2,   Func. Count:     13,   Neg. LLF: 16572.898725581304
Iteration:      3,   Func. Count:     21,   Neg. LLF: 9640.094051505308
Iteration:      4,   Func. Count:     26,   Neg. LLF: 9158.654548912758
Iteration:      5,   Func. Count:     31,   Neg. LLF: 9140.822754818397
Iteration:      6,   Func. Count:     35,   Neg. LLF: 9140.628822666982
Iteration:      7,   Func. Count:     39,   Neg. LLF: 9140.62073006493
Iteration:      8,   Func. Count:     43,   Neg. LLF: 9140.620477354729
Iteration:      9,   Func. Count:     47,   Neg. LLF: 9140.620475280153
Iteration:     10,   Func. Count:     50,   Neg. LLF: 9140.620475280237
Optimization terminated successfully    (Exit mode 0)
            Current function value: 9140.620475280153

Function evaluations: 50 Gradient evaluations: 10
```

```
garch_1_0_fit.summary()
```

Dep. Variable:	Close	**R-squared:**	-0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	-0.000
Vol Model:	ARCH	**Log-Likelihood:**	-9140.62
Distribution:	Normal	**AIC:**	18287.2
Method:	Maximum Likelihood	**BIC:**	18307.0
		No. Observations:	5352
Date:	Tue, Aug 03 2021	**Df Residuals:**	5349
Time:	13:05:42	**Df Model:**	3

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.0691	1.910e-02	3.618	2.969e-04	[3.166e-02, 0.107]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.2660	6.199e-02	20.425	1.013e-92	[1.145, 1.388]
alpha[1]	0.4274	6.168e-02	6.929	4.229e-12	[0.307, 0.548]

Covariance Estimator: robust

All coefficients are significant

```
rolling_predictions=[]
test_size=365

for i in range(test_size):
    train=returns[:-(test_size-i)]
    model=arch_model(train,p=1,q=0)
    model_fit=model.fit(disp="off")
    pred=model_fit.forecast(horizon=1)
    rolling_predictions.append(np.sqrt(pred.variance.values[-1,:][0]))
```

```
rolling_predictions=pd.Series(rolling_predictions,index=returns.index[-365:])
```

```
plt.figure(figsize=(22,4))
true,=plt.plot(returns[-365:])
preds,=plt.plot(rolling_predictions)
plt.title("GARCH(1,0)",fontsize=15)
plt.legend(["True Volatility","Predicted Volatility"],fontsize=15)
plt.ylim([-13,13])
plt.show()
```

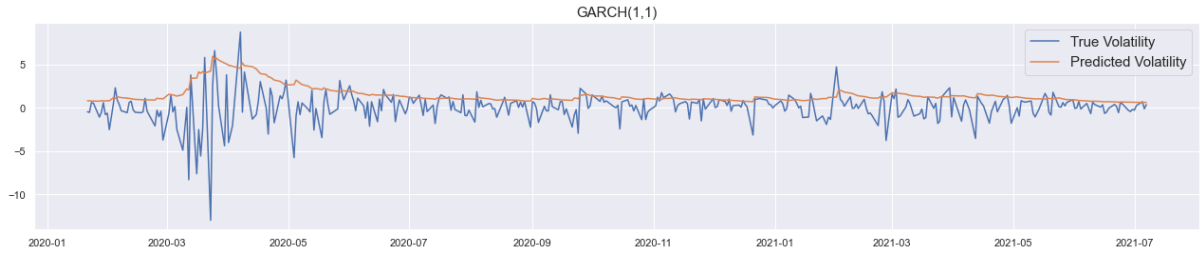


Figure: 16

```
df1=pd.DataFrame(returns[-365:])
```

```
df2=pd.DataFrame(rolling_predictions)
```

```
actual_vs_predicted=pd.concat([df1,df2],1)
```

```
actual_vs_predicted.columns=["Actual Volatility","Predicted Volatility"]
```

```
actual_vs_predicted["Predicted Volatility"]=actual_vs_predicted["Predicted Volatility"]
actual_vs_predicted["Actual Volatility"]=actual_vs_predicted["Actual Volatility"]
```

```
actual_vs_predicted.head(10)
```

Date	Actual Volatility	Predicted Volatility
2020-01-21	-0.447460	1.323465
2020-01-22	-0.517262	1.162688
2020-01-23	0.606679	1.176319
2020-01-24	0.557455	1.167763
2020-01-27	-1.055253	1.158390
2020-01-28	-0.521495	1.330232
2020-01-29	0.611324	1.176828
2020-01-30	-0.772497	1.168372
2020-01-31	-0.612340	1.239017
2020-02-01	-2.510011	1.196883

```
train=returns
model=arch_model(train,p=1,q=1)
model_fit=model.fit(disp="off")
from datetime import timedelta
```

```
pred=model_fit.forecast(horizon=1)
future_dates=[returns.index[-1]+timedelta(days=i) for i in range(1,2)]
pred=pd.Series(np.sqrt(pred.variance.values[-1,:]),index=future_dates)
```

Predicted value for next day, i.e. 8th July 2021

```
pred=pred
print(pred)
```

```
2021-07-08    0.62031

dtype: float64
```

Both GARCH(1,0) and GARCH(1,1) made the same prediction, i.e. 0.62031 for the next day, 8th July 2020

Predicted Volatility

Date		
2020-01-21	-0.447460	0.819918
2020-01-22	-0.517262	0.821933
2020-01-23	0.606679	0.797426
2020-01-24	0.557455	0.798564
2020-01-27	-1.055253	0.769366
2020-01-28	-0.521495	0.833573
2020-01-29	0.611324	0.826572
2020-01-30	-0.772497	0.808479
2020-01-31	-0.612340	0.824666
2020-02-01	-2.510011	0.825723

Garch (2,2)

```
garch_2_2=arch_model(returns,p=2,q=2)
garch_2_2_fit=garch_2_2.fit()
```

```
Iteration:      1,   Func. Count:      8,   Neg. LLF: 49399.93518080427
Iteration:      2,   Func. Count:     20,   Neg. LLF: 31909.950809555947
Iteration:      3,   Func. Count:     31,   Neg. LLF: 2009648434.60998
Iteration:      4,   Func. Count:     39,   Neg. LLF: 8733.498357431148
Iteration:      5,   Func. Count:     47,   Neg. LLF: 8551.99244970796
Iteration:      6,   Func. Count:     55,   Neg. LLF: 8570.576567205439
Iteration:      7,   Func. Count:     64,   Neg. LLF: 8789.61697456646
Iteration:      8,   Func. Count:     72,   Neg. LLF: 8538.408386750312
Iteration:      9,   Func. Count:     79,   Neg. LLF: 8538.364107817972
Iteration:     10,   Func. Count:     86,   Neg. LLF: 8538.373544356498
Iteration:     11,   Func. Count:     94,   Neg. LLF: 8538.355163145878
Iteration:     12,   Func. Count:    101,   Neg. LLF: 8538.354039221424
Iteration:     13,   Func. Count:    107,   Neg. LLF: 8538.35403922279
Optimization terminated successfully    (Exit mode 0)
            Current function value: 8538.354039221424

Iterations: 13

Function evaluations: 107

Gradient evaluations: 13
```

```
garch_2_2_fit.summary()
```

Constant Mean - GARCH Model Results

Dep. Variable:	Close	**R-squared:**	-0.001
Mean Model:	Constant Mean	**Adj. R-squared:**	-0.001
Vol Model:	GARCH	**Log-Likelihood:**	-8538.35
Distribution:	Normal	**AIC:**	17088.7
Method:	Maximum Likelihood	**BIC:**	17128.2
		No. Observations:	5352
Date:	Tue, Aug 03 2021	**Df Residuals:**	5346
Time:	13:04:20	**Df Model:**	6

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.0899	1.516e-02	5.933	2.975e-09	[6.022e-02, 0.120]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0543	1.455e-02	3.733	1.890e-04	[2.580e-02,8.283e-02]
alpha[1]	0.1015	1.633e-02	6.220	4.968e-10	[6.955e-02, 0.134]
alpha[2]	0.1156	1.916e-02	6.034	1.602e-09	[7.804e-02, 0.153]
beta[1]	0.0000	6.239e-02	0.000	1.000	[-0.122, 0.122]
beta[2]	0.7625	5.874e-02	12.979	1.606e-38	[0.647, 0.878]

Covariance estimator: robust

All the coefficients except beta 1 are significant

```
rolling_predictions=[]
test_size=365

for i in range(test_size):
    train=returns[:-(test_size-i)]
    model=arch_model(train,p=2,q=2)
    model_fit=model.fit(disp="off")
    pred=model_fit.forecast(horizon=1)
    rolling_predictions.append(np.sqrt(pred.variance.values[-1,:][0]))

rolling_predictions=pd.Series(rolling_predictions,index=returns.index[-365:])

plt.figure(figsize=(22,4))
true,=plt.plot(returns[-365:])
preds,=plt.plot(rolling_predictions)
plt.title("GARCH(2,2)",fontsize=15)
plt.legend(["True Volatility","Predicted Volatility"],fontsize=15)
plt.show()
```

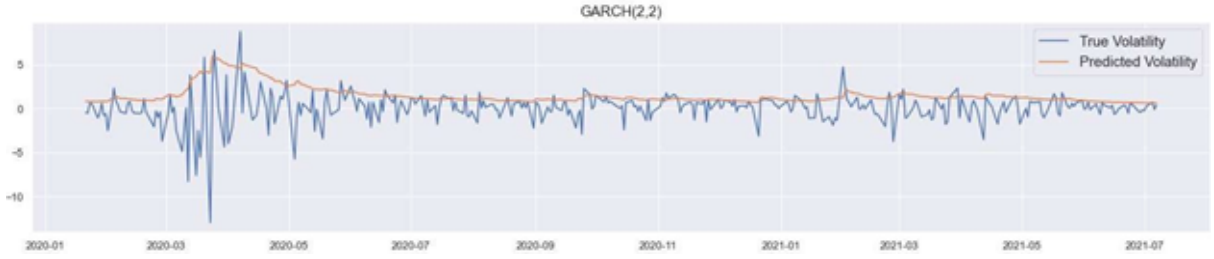


Figure: 17

```
df1=pd.DataFrame(returns[-365:])

df2=pd.DataFrame(rolling_predictions)

actual_vs_predicted=pd.concat([df1,df2],1)

actual_vs_predicted.columns=["Actual Volatility","Predicted Volatility"]

actual_vs_predicted["Predicted Volatility"]=actual_vs_predicted["Predicted Volatility"]
actual_vs_predicted["Actual Volatility"]=actual_vs_predicted["Actual Volatility"]

actual_vs_predicted.head(10)
```

Date	Actual Volatility	Predicted Volatility
2020-01-21	-0.447460	0.819918
2020-01-22	-0.517262	0.821933
2020-01-23	0.606679	0.797426
2020-01-24	0.557455	0.798564
2020-01-27	-1.055253	0.769366
2020-01-28	-0.521495	0.833573
2020-01-29	0.611324	0.826572
2020-01-30	-0.772497	0.808479
2020-01-31	-0.612340	0.824666
2020-02-01	-2.510011	0.825723

```
train=returns
model=arch_model(train,p=2,q=2)
model_fit=model.fit(disp="off")
from datetime import timedelta

pred=model_fit.forecast(horizon=1)
future_dates=[returns.index[-1]+timedelta(days=i) for i in range(1,2)]
pred=pd.Series(np.sqrt(pred.variance.values[-1,:]),index=future_dates)
```

The predicted value for the next day, i.e. 8th July 2021

```
pred=pred
print(pred)
```

```
2021-07-08    0.615942

dtype: float64
```

LSTM:

Python:

Libraries Imported: Pandas, Numpy, Matplotlib, Seaborn, Tensorflow, Keras, sklearn

Dataframe: Nifty50

```
df.head()
```

Out[2]:

Date	Open	High	Low	Close	Volume	Turnover	P/E	P/B	Div Yield
2000-01-03	1482.15	1592.90	1482.15	1592.2	25358322	8.841500e+09	25.91	4.63	0.95
2000-01-04	1594.40	1641.95	1594.40	1638.7	38787872	1.973690e+10	26.67	4.76	0.92
2000-01-05	1634.55	1635.50	1555.05	1595.8	62153431	3.084790e+10	25.97	4.64	0.95
2000-01-06	1595.80	1639.00	1595.80	1617.6	51272875	2.531180e+10	26.32	4.70	0.94
2000-01-07	1616.60	1628.25	1597.20	1613.3	54315945	1.914630e+10	26.25	4.69	0.94

After importing the libraries, the univariate data was created from the 'Close' column of our data. We then split the data into an 80-20 ratio, i.e. 80% of the dataset will be considered as training data and the rest 20% as testing data.

We used the sliding window approach to create our dataset in which the closing prices of the first 7 days were used and the closing price of the 8th day was used as a label. This way we made our task a supervised learning problem.

Plotting of Nifty50 Close Prices:

Figure: 18

Prediction Plots:

1. Model 1: The first model we used contains 1 LSTM layer with 128 cells. The mean absolute error, in this case, was 96.59.

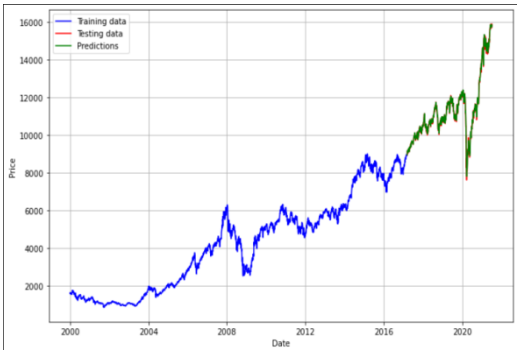


Figure: 19

Figure: 20

2. Model 2: In the second model, we used 2 LSTM layers with 128 cells each and a fully connected/dense layer with 64 neurons with relu activation in each layer. The mean absolute error, in this case, was 93.07.

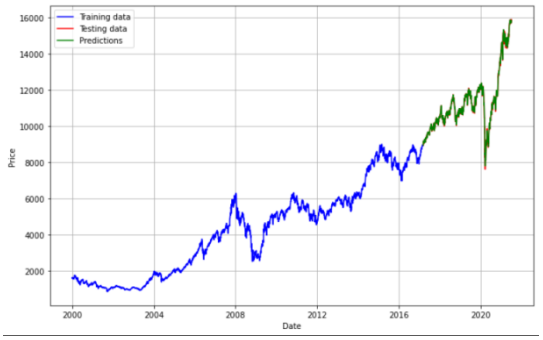


Figure: 21

Figure: 22

3. **Model 3:** For the third model, we used the same model architecture as model 2 but in this case, we used feature scaling to bring our features on the same scale. The mean absolute error, in this case, was 100.16.

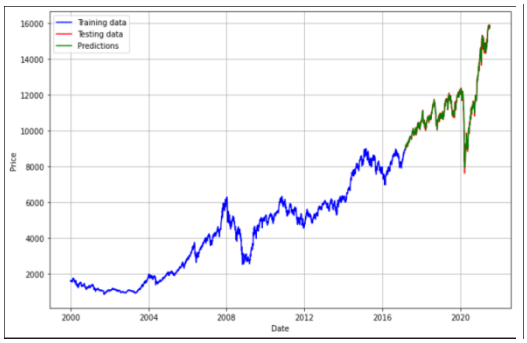


Figure: 23

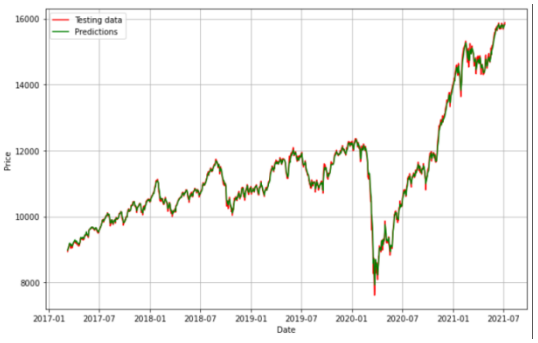


Figure: 24

Code : https://github.com/satorugojo3795/Nifty-50-price-prediction

Autoencoder:

```
model = keras.Sequential()
model.add(keras.layers.LSTM(
    units=64,
    input_shape=(X_train.shape[1], X_train.shape[2])
))
model.add(keras.layers.Dropout(rate=0.2))
model.add(keras.layers.RepeatVector(n=X_train.shape[1]))
model.add(keras.layers.LSTM(units=64, return_sequences=True))
model.add(keras.layers.Dropout(rate=0.2))
model.add(keras.layers.TimeDistributed(keras.layers.Dense(units=X_train.shape[2])))
model.compile(loss='mae', optimizer='adam')
```

```
history = model.fit(
    X_train, y_train,
    epochs=40,
    batch_size=64,
    validation_split=0.1,
    shuffle=False
)
```

In the LSTM autoencoder network architecture, the first couple of neural network layers create the compressed representation of the input data, the encoder, then we use a repeat vector layer to distribute the compressed representational vector across the time steps of the decoder. The final output layer of the decoder provides us with the reconstructed input data.

We then instantiate the model and compile it using Adam as our neural network optimizer and mean absolute error for calculating our loss function.

Loss Distribution:

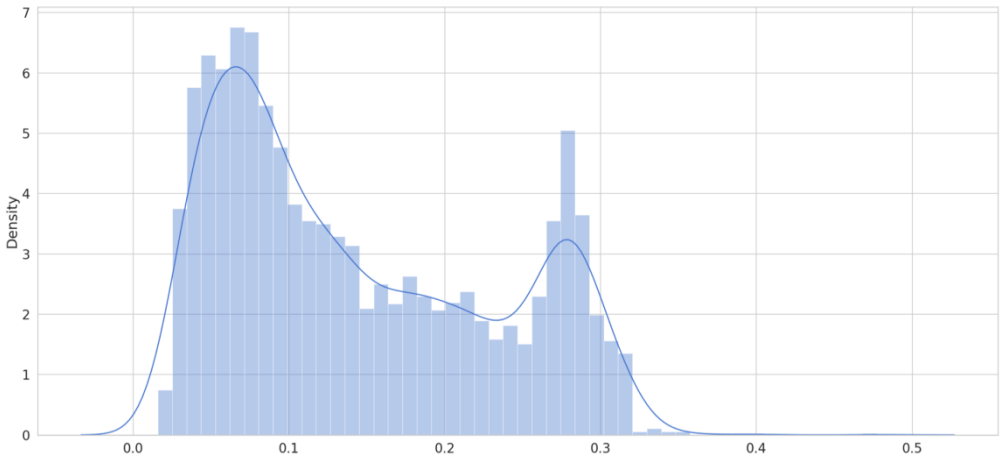


Figure: 25

By plotting the distribution of the calculated loss in the training set, we determine a threshold value of 0.4 for identifying an anomaly. In doing this, the threshold is set above the "noise level" so that false positives are not triggered.

```
X_test_pred = model.predict(X_test)

test_mae_loss = np.mean(np.abs(X_test_pred - X_test), axis=1)
```

```
THRESHOLD = 0.4

test_score_df = pd.DataFrame(index=test[TIME_STEPS:].index)
test_score_df['loss'] = test_mae_loss
test_score_df['threshold'] = THRESHOLD
test_score_df['anomaly'] = test_score_df.loss > test_score_df.threshold
test_score_df['Close'] = test[TIME_STEPS:].Close
```

```
plt.plot(test_score_df.index, test_score_df.loss, label='loss')
plt.plot(test_score_df.index, test_score_df.threshold, label='threshold')
plt.xticks(rotation=25)
plt.legend();
```

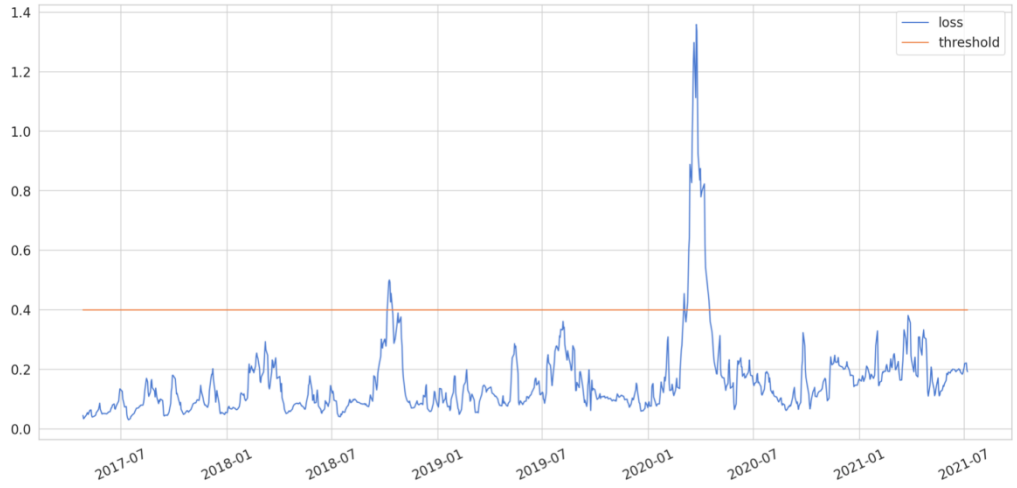


Figure: 26

Anomalies*:*

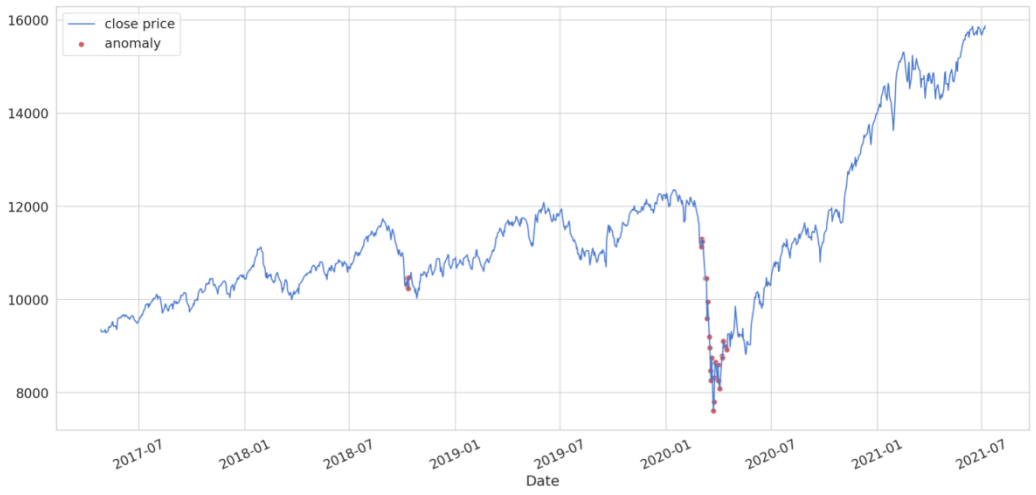


Figure: 27

The model detected the sudden fall in the Nifty 50 index from 3rd March 2020 to 15th April 2020 caused by the Covid-19 pandemic, and the falls which occurred in the second week of October 2018, which were results of heavy sell-off in world markets.

On March 4 and 6, markets fell by around 1000 points. On March 9, the Nifty 50 again broke down by 538 points, this time due to the havoc created by the fear of the COVID-19 outbreak. On March 16, Nifty ended below the 9200-mark at 9197.40 and by March 23, Nifty was at 7610.25 points as the coronavirus-led lockdowns across the world triggered fears of a recession.

Correlation analysis:

Correlation analysis of stock data can be implemented using Python. It is used to provide a piece of statistical evidence determining how the two concerned variables are related to each other — positively, negatively or non-related.

We have calculated correlation coefficients for

i. SGX Nifty with NIFTY
ii. FTSE with NIFTY
iii. US Treasury Bonds with NIFTY
iv. CII and NIFTY Realty

For the calculation of correlation, the everyday 'Close' values have been taken under consideration.

Python:

Libraries imported: Pandas, Numpy, MatPlotlib, Seaborn (to plot heatmaps), Scipy.stats (to calculate Pearson and Spearman correlation coefficient)

i. *NIFTY with SGX Nifty*

Time Duration: 4th January 2010 to 31st December 2020

```
df_1 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/sgx rupees.xlsx",parse_dates=["Date"],sheet_name="SGX")
df_1.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume	Close Rs.
0	2010-01-04	8.32	8.32	8.22	8.26	5.359976	1809000.0	273.62076
1	2010-01-05	8.30	8.33	8.23	8.31	5.392423	3024000.0	274.85325
2	2010-01-06	8.31	8.35	8.30	8.34	5.411890	3437000.0	273.44358
3	2010-01-07	8.35	8.52	8.34	8.46	5.489757	7120000.0	276.34590
4	2010-01-08	8.48	8.48	8.38	8.40	5.450823	2433000.0	273.93240

Dataframe: SGX Nifty

```
df_2 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/sgx rupees.xlsx",sheet_name="NIFTY")
df_2.head()
```

	Date	Open	High	Low	Close
0	2010-01-04	5200.90	5238.45	5167.10	5232.20
1	2010-01-05	5277.15	5288.35	5242.40	5277.90
2	2010-01-06	5278.15	5310.85	5260.05	5281.80
3	2010-01-07	5281.80	5302.55	5244.75	5263.10
4	2010-01-08	5264.25	5276.75	5234.70	5244.75

Dataframe: NIFTY

```
close_prices = pd.DataFrame(columns=["Date","Close sgxNifty","Close nifty50"])
```

```
#converting to csv
close_prices.to_csv("closediff.csv",index=False)
```

```
ser1 = pd.read_csv("closediff.csv")
```

Converting the data to CSV and creating the series.

```
ser1.isnull().sum().sum()
```

```
1
```

```
ser1['Close sgxNifty'].isnull().sum()
```

```
1
```

```
ser1 = ser1[ser1['Close sgxNifty'].notna()]
```

Checking for the presence of any null values and eliminating them to create the final data for the correlation analysis.

```
ser1
```

	Date	Close sgxNifty	Close nifty50
0	2010-01-04	273.62076	5232.20
1	2010-01-05	274.85325	5277.90
2	2010-01-06	273.44358	5281.80
3	2010-01-07	276.34590	5263.10
4	2010-01-08	273.93240	5244.75
...	...	...	...
2643	2020-12-22	507.29598	13466.30
2644	2020-12-23	515.52690	13601.10
2646	2020-12-28	510.24363	13873.20
2647	2020-12-29	511.37700	13932.60
2648	2020-12-30	516.10038	13981.95

2648 rows × 3 columns

Karl Pearson's Coefficient: 0.8201890837612921

Spearman's Coefficient: 0.8709626028466887

Visualisation of correlation:

Heatmap:

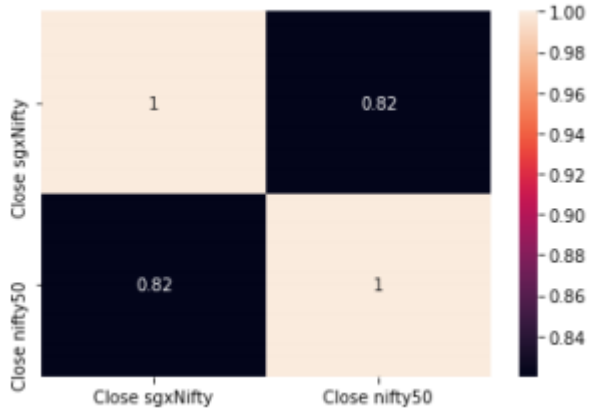


Figure: 28

Relational Plot:

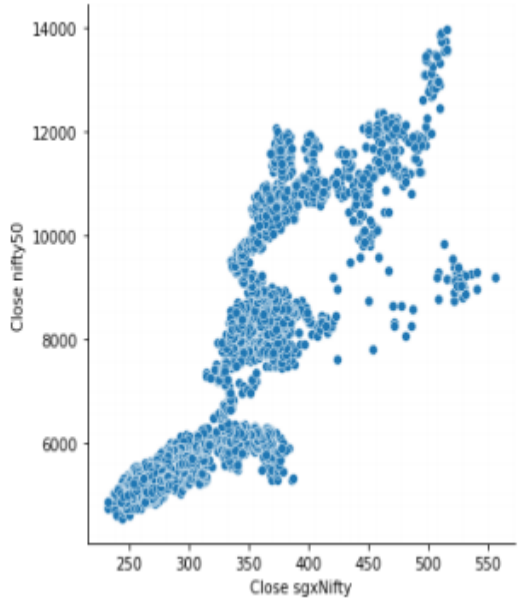


Figure: 29

Area Plot:

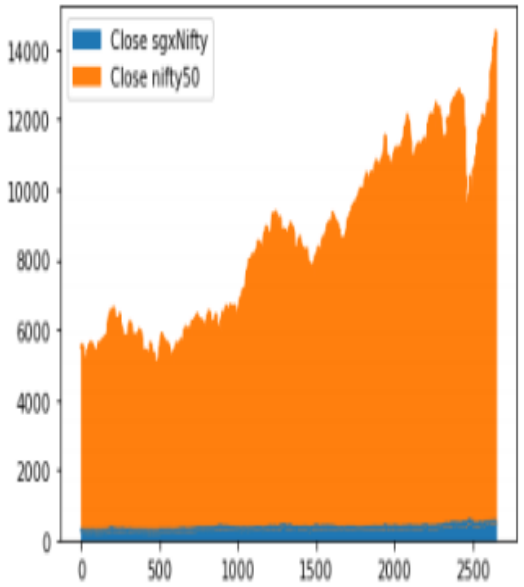


Figure: 30

Plotting of Close Prices:

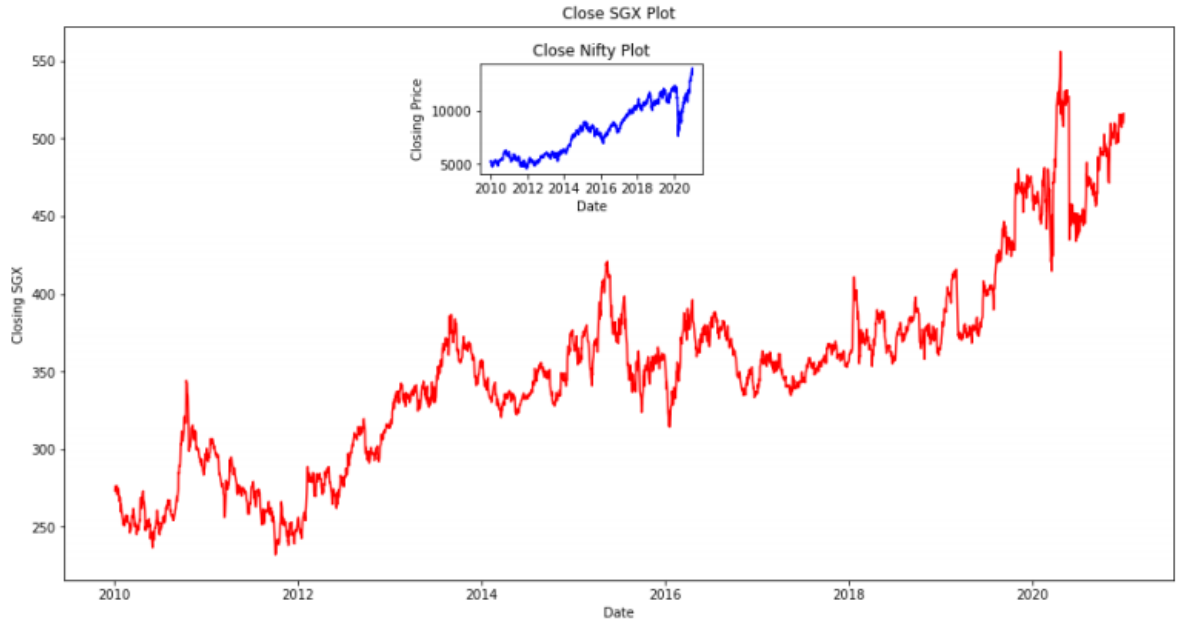


Figure: 31

The value of Karl Pearson's Correlation Coefficient comes out to be 0.82, denoting a strong positive correlation between SGX Nifty and Nifty.

ii. NIFTY with FTSE

Time Duration: 4th January 2010 to 31st December 2020

```
df_1 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/FTSE Data Final.xlsx",parse_dates=["Date"],sheet_name="FTSE Rupees")
df_1.head()
```

	Date	Price	Open	High	Low	Volume	Chg%	Ftse Rs.
0	2010-01-04	74.522	75.159	75.325	74.395	0	-0.0114	409896.33748
1	2010-01-05	73.915	74.444	74.726	73.619	0	-0.0081	408195.58750
2	2010-01-06	73.231	73.789	74.090	73.019	0	-0.0093	404970.35924
3	2010-01-07	72.745	73.125	73.404	72.552	0	-0.0066	402041.24640
4	2010-01-08	72.877	72.718	73.577	72.436	0	0.0018	403318.80848

Dataframe: FTSE

```
df_2 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/FTSE Data 2009-2021.xlsx",sheet_name="NIFTY")
df_2.head()
```

	Date	Open	High	Low	Close
0	2010-01-04	5200.90	5238.45	5167.10	5232.20
1	2010-01-05	5277.15	5288.35	5242.40	5277.90
2	2010-01-06	5278.15	5310.85	5260.05	5281.80
3	2010-01-07	5281.80	5302.55	5244.75	5263.10
4	2010-01-08	5264.25	5276.75	5234.70	5244.75

Dataframe: NIFTY

```
close_prices["Date"] = date
```

```
#converting to csv
close_prices.to_csv("closediff6.csv",index=False)
```

```
ser1 = pd.read_csv("closediff6.csv")
```

Converting the data to CSV and creating the series.

```
ser1.isnull().sum().sum()
```

```
0
```

```
ser1['Close ftseNifty'].isnull().sum()
```

```
0
```

```
ser1 = ser1[ser1['Close ftseNifty'].notna()]
```

Checking for the presence of null values and eliminating them to create the final data for the correlation analysis.

```
ser1
```

	Date	Close ftseNifty	Close nifty50
0	2010-01-04	409896.33748	5232.20
1	2010-01-05	408195.58750	5277.90
2	2010-01-06	404970.35924	5281.80
3	2010-01-07	402041.24640	5263.10
4	2010-01-08	403318.80848	5244.75
...	...	...	...
2641	2020-12-22	636701.03140	13466.30
2642	2020-12-23	646521.99750	13601.10
2643	2020-12-29	654170.75405	13932.60
2644	2020-12-30	653044.89766	13981.95
2645	2020-12-31	645160.44824	13981.75

2646 rows × 3 columns

Karl Pearson's Coefficient: 0.6931471120909336

Spearman's Coefficient: 0.6704761073054877

Visualisation of correlation:

Heatmap:

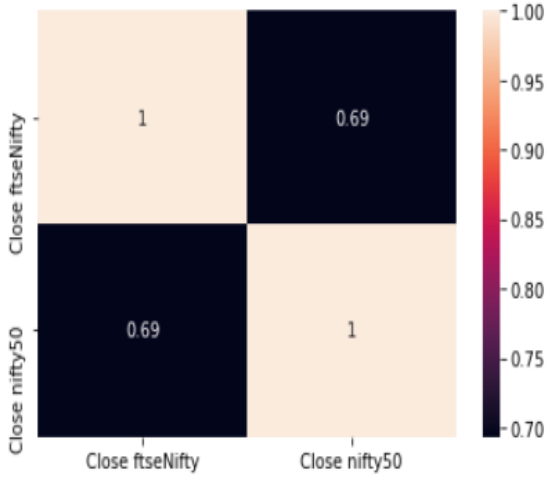


Figure: 32

Relational Plot:

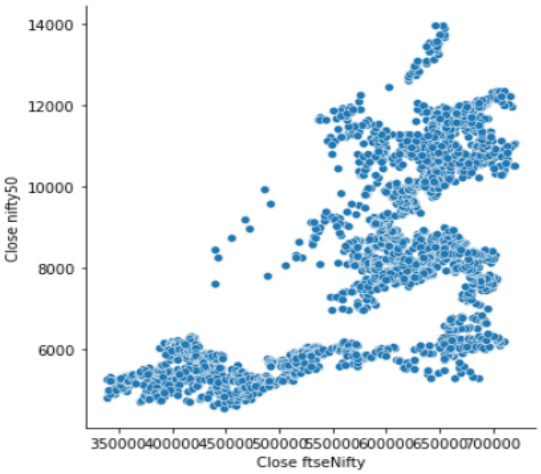


Figure: 33

Area Plot:

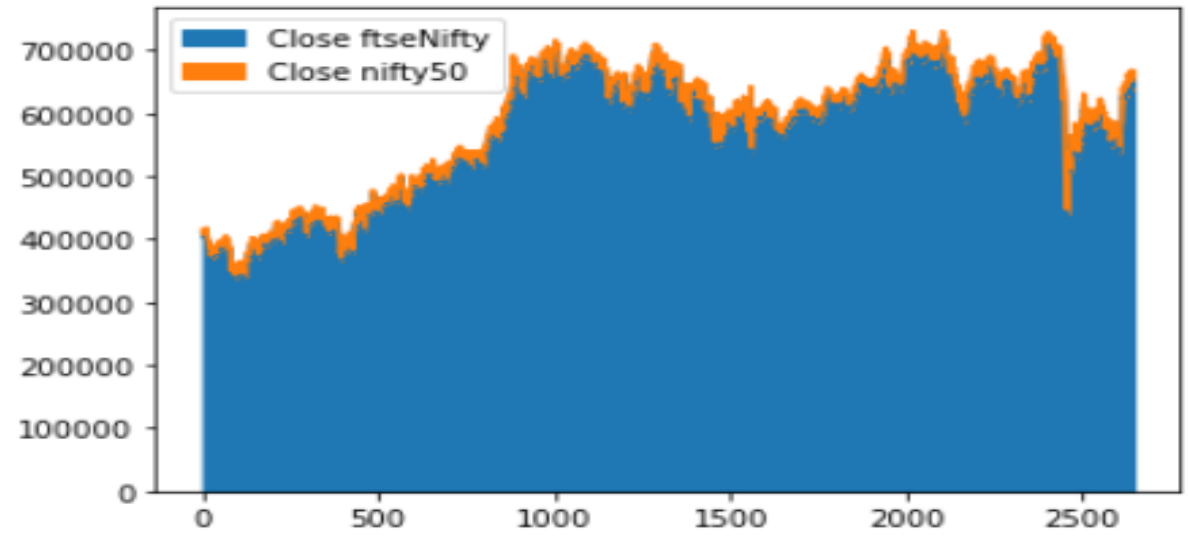


Figure: 34

Plotting of Close Prices

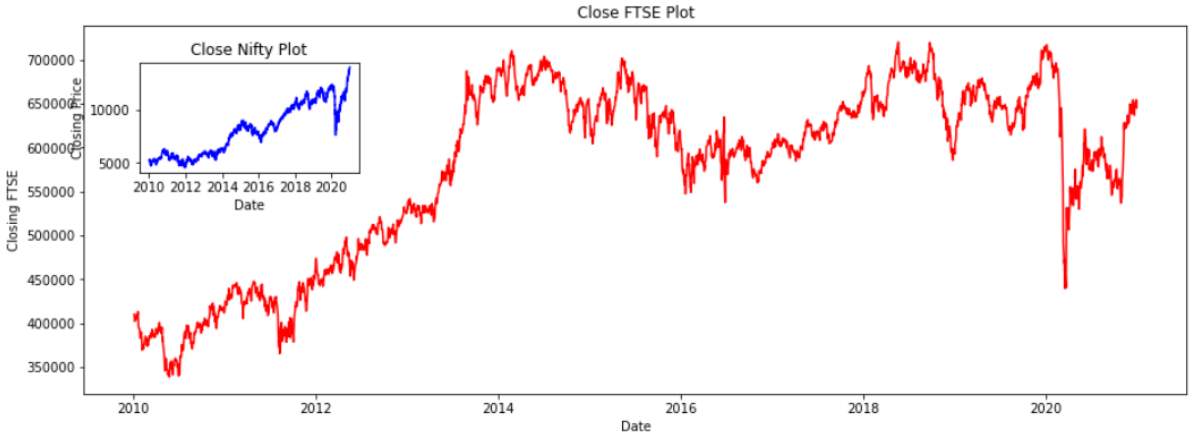


Figure: 35

The value of Karl Pearson's Correlation Coefficient comes out to be 0.69, denoting a strong positive correlation between FTSE and Nifty.

iii. NIFTY with US Treasury Bonds

Time Duration: 3rd January 2011 to 31st December 2020

```
df_1 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/US Tr & NIFTY Final.xlsx",parse_dates=["Date"],sheet_name="US Tr")
df_1.head()
```

	Date	5 Yr	10 Yr	20 Yr	30 Yr
0	2011-01-03	2.02	3.36	4.18	4.39
1	2011-01-04	2.01	3.36	4.21	4.44
2	2011-01-05	2.14	3.50	4.34	4.55
3	2011-01-06	2.09	3.44	4.31	4.53
4	2011-01-07	1.96	3.34	4.25	4.48

Dataframe: US Treasury Bond

```
df_2 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/US Tr & NIFTY Final.xlsx",sheet_name="NIFTY")
df_2.head()
```

	Date	Open	High	Low	Close
0	2011-01-03	6177.45	6178.55	6147.20	6157.60
1	2011-01-04	6172.75	6181.05	6124.40	6146.35
2	2011-01-05	6141.35	6141.35	6062.35	6079.80
3	2011-01-06	6107.00	6116.15	6022.30	6048.25
4	2011-01-07	6030.90	6051.20	5883.60	5904.60

Dataframe: NIFTY

```
close_prices = pd.DataFrame(columns=["Date","Close ustNifty","Close nifty50"])
```

```
#converting to csv
close_prices.to_csv("closediff3.csv",index=False)
```

```
ser1 = pd.read_csv("closediff3.csv")
```

Converting the data to CSV and creating the series

```
ser1.isnull().sum().sum()
```

```
0
```

```
ser1['Close ustNifty'].isnull().sum()
```

```
0
```

```
ser1 = ser1[ser1['Close ustNifty'].notna()]
```

Checking for the presence of null values and eliminating them to create the final data for the correlation analysis

```
ser1
```

	Date	Close ustNifty	Close nifty50
0	2011-01-03	3.36	6157.60
1	2011-01-04	3.36	6146.35
2	2011-01-05	3.50	6079.80
3	2011-01-06	3.44	6048.25
4	2011-01-07	3.34	5904.60
...	...	...	...
2373	2020-12-24	0.94	13749.25
2374	2020-12-28	0.94	13873.20
2375	2020-12-29	0.94	13932.60
2376	2020-12-30	0.93	13981.95
2377	2020-12-31	0.93	13981.75

2378 rows × 3 columns

Karl Pearson's Coefficient: -0.20711355016632077

Spearman's Coefficient: -0.08758583408744762

Visualisation of correlation:

Heatmap:

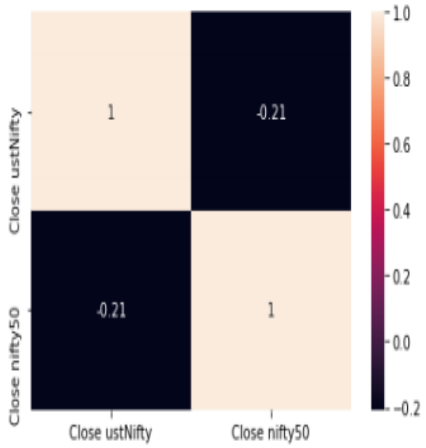


Figure: 36

Relational Plot:

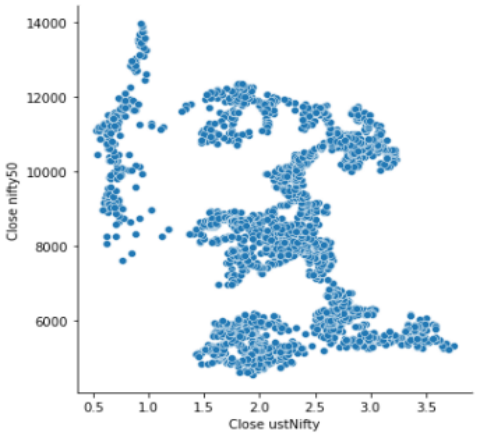


Figure: 37

Area Plot

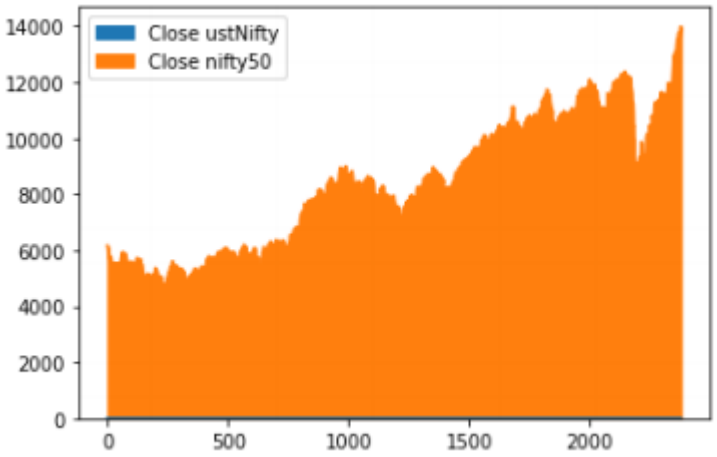


Figure: 38

Plotting of Close Prices

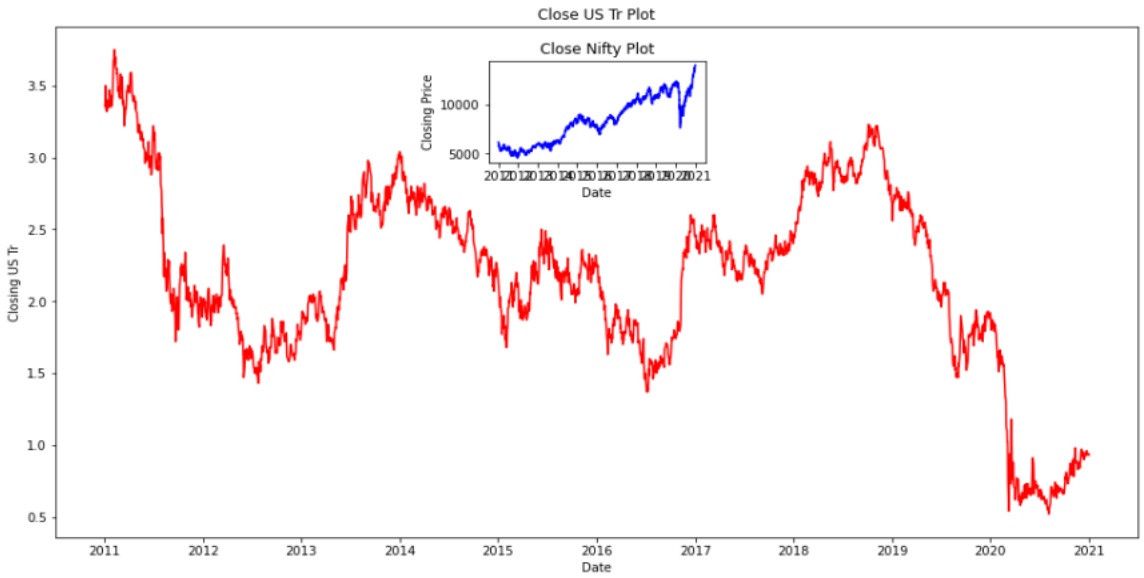


Figure: 39

The value of Karl Pearson's Correlation Coefficient comes out to be -0.21, denoting a weak negative correlation between US Treasury Bonds and Nifty.

iv. CII with NIFTY Realty

Time Duration: 2011-12 to 2020-21

```
In [2]: df_1 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/Nifty Realty Yearly.xlsx",parse_dates=["Financial Year"],sheet_name="CII")
        df_1.head()
```

Out[2]:

	Financial Year	Cost Inflation Index
0	2011-12	184
1	2012-13	200
2	2013-14	220
3	2014-15	240
4	2015-16	254

Dataframe: CII

```
In [7]: df_2 = pd.read_excel("C:/Users/KLIN/OneDrive/Desktop/Nifty Realty Yearly.xlsx",sheet_name="NIFTY R")
        df_2.head()
```

Out[7]:

	Year	Nifty Realty avg
0	2011-12	248.191667
1	2012-13	239.854167
2	2013-14	178.550000
3	2014-15	222.200000
4	2015-16	172.054167

Dataframe: NIFTY Realty

```
close_prices = pd.DataFrame(columns=["CII","Close nifty r"])
```

```
#converting to csv
close_prices.to_csv("closediff5.csv",index=False)
```

```
ser1 = pd.read_csv("closediff5.csv")
```

Converting the data to CSV and creating the series.

```
ser1.isnull().sum().sum()
```

```
0
```

```
ser1['CII'].isnull().sum()
```

```
0
```

```
ser1 = ser1[ser1['CII'].notna()]
```

```
ser1
```

Checking for the presence of null values and eliminating them to create the final data for the correlation analysis.

```
ser1
```

	CII	Close nifty r	Year
0	184	248.191667	2011-12
1	200	239.854167	2012-13
2	220	178.550000	2013-14
3	240	222.200000	2014-15
4	254	172.054167	2015-16
5	264	192.487500	2016-17
6	272	295.366667	2017-18
7	280	255.358333	2018-19
8	289	271.429167	2019-20
9	301	249.879167	2020-21

Karl Pearson's Coefficient: 0.28087648844100666

Spearman's Coefficient: 0.5272727272727272

Visualisation of correlation:

Heatmap:

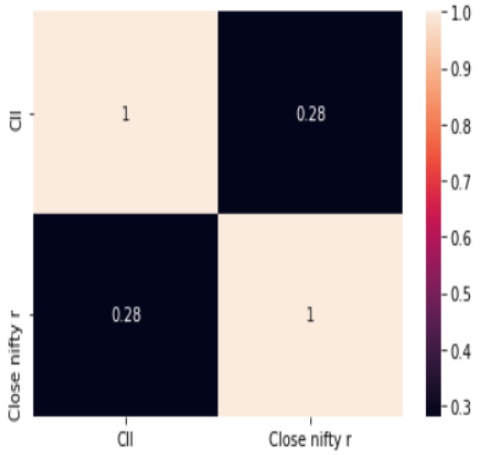


Figure: 40

Relational Plot:

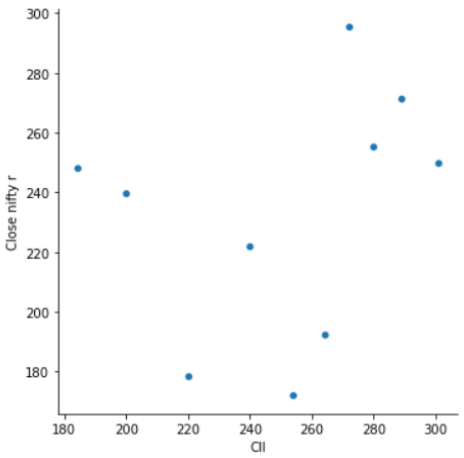


Figure: 41

Area Plot

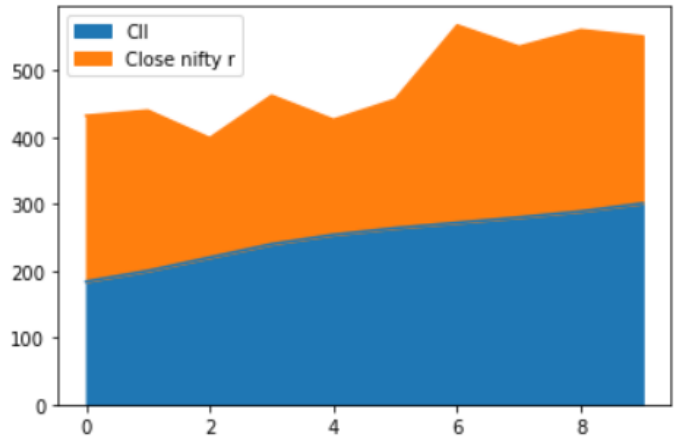


Figure: 42

Plotting of Close Prices

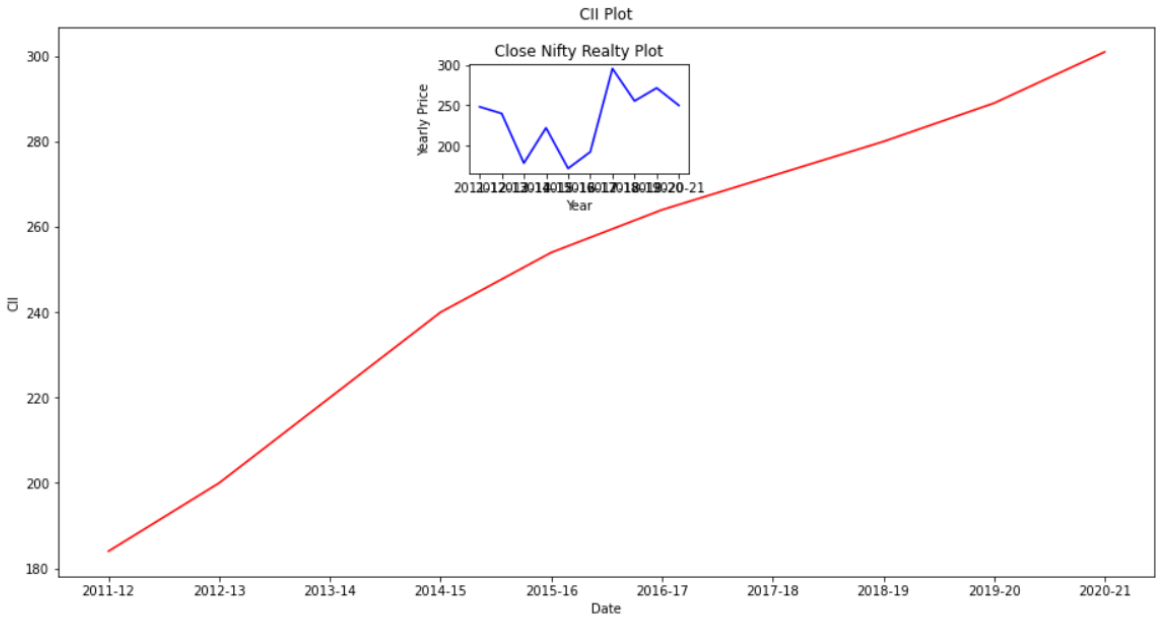


Figure: 43

The value of Karl Pearson's Correlation Coefficient comes out to be 0.28, denoting a weak positive correlation between CII and Nifty Realty.

INTERPRETATION

For analysis, we used a 20-years time-series dataset that has day-level information on major NIFTY-50 indices starting from 1 Jan 2000 to 7 July 2021. We forecast the value and volatility of Nifty -50 indices for the next day, i.e. 8th July 2021.

As the first step, we have shown the four components of time-series, i.e. trend, seasonality, cyclic and random components. The component study helped us in a holistic understanding of the given time series. As we can see in Graph 1.15, there is an overall increasing trend for the last 20 years. There has also been the presence of seasonality in the dataset over the last 20 years. For the random component, there are two major falls in the Nifty-50 index, first in the year 2007-08 i.e. U.S. housing crisis and in the year 2019-20 due to the COVID-19 pandemic.

In the model building, the Nifty-50 data is found to be non-stationary, but by the first order differencing of Nifty 50 data becomes stationary. ARIMA (p, d, q) model is fitted with different values of p, d and q with an objective of minimizing AIC and BIC. Best results are observed with ARIMA (5, 1, 2). Forecasting is done and the result is shown in a table comparing Predicted value and Actual value. The predicted closing value of the Nifty-50 index for the next day comes out to be 15316.93639. Mean Absolute error for the ARIMA (5, 1, 2) is 357.4048. As we can see in the graph, our model finds a value that is less than the actual value, i.e., it underestimates the value.

GARCH MODEL is used to study and predict the volatility of the Nifty-50 indices. For this data, GARCH (1, 1) comes out to be the better model as compared to GARCH (1, 0) and GARCH (2, 2) as both coefficients of GARCH (1,1) are significant. According to GARCH (1,1), the predicted volatility of the Nifty 50 index for the next day is 0.62031.

We used the **LSTM MODEL** to detect the sudden fall in the Nifty 50 index from 3rd March 2020 to 15th April 2020 due to the Covid-19 pandemic, and the falls which occurred in the second week of October 2018, which were results of heavy sell-off in world markets. On March 4 and 6, markets fell by around 1000 points. On March 9, the Nifty 50 again broke down by 538 points, this time due to the havoc created by the fear of the COVID-19 outbreak. On March 16, Nifty ended below the 9200-mark at 9197.40 and by March 23, Nifty was at 7610.25 points as the coronavirus-led lockdowns across the world triggered fears of a recession.

The predicted closing value of Nifty-50 Index for the next day: 15874.05

Mean absolute error of the LSTM model = 93.07332

After comparing Mean Absolute Error (MAE) of ARIMA and LSTM, it is clear that LSTM has less MAE as compared to ARIMA. So, for prediction, the LSTM model will be more preferable as compared to the ARIMA model.

The NIFTY Index is compared with SGX NIFTY, FTSE and US Treasury bonds relating to their value (Price) movements by finding Correlation Coefficients.

Results are as follows:

1. Correlation Coefficient between NIFTY and SGX NIFTY

Karl Pearson Coefficient = 0.8202

Spearman's Correlation Coefficient = 0.8710

This shows that NIFTY and SGX NIFTY have a strong positive correlation in their value movement.

2. Correlation Coefficient between NIFTY and FTSE

Karl Pearson Coefficient = 0.6931

Spearman's Correlation Coefficient = 0.6705

This shows that NIFTY and FTSE have a fairly positive correlation in their value movement.

3. Correlation Coefficient between NIFTY and US Treasury Bond

Karl Pearson Coefficient = -0.2071

Spearman's Correlation Coefficient = -0.0876

This shows that NIFTY and US Treasury Bonds have a weak negative correlation in their value Movement. We can say that both move in opposite directions, but the correlation between them is not that strong.

4. Correlation Coefficient between CII (Cost Inflation Index) and NIFTY Realty

Karl Pearson Coefficient = 0.2809

Spearman's Correlation Coefficient = 0.5273

This shows that CII (House Price Index) and NIFTY Realty are positively correlated in their value movement, but this correlation is not strong.

REFERENCES

[1] Akshay Sachdeva et al (2016): An Effective Time Series Analysis for Equity Market Prediction Using Deep Learning Model

[2] Limsombunchai, Gan and Lee (2004): House Price Prediction: Hedonic Price Model vs. Artificial Neural Network, American Journal of Applied Sciences, 1(3), 193-201.

[3] Swathy Ajayakumar (2017): A Study on the Relationship between NIFTY Realty Stock Index and Macroeconomic Variables, MA thesis, Christ University, Bangalore.

[4] Tiffany Hui-Kuang Yu, Kun-Huang Huarng (2010): A neural network-based fuzzy time series model to improve forecasting, Expert Systems with Applications, 37(4), 3366-3372.

[5] Fama, E. and Schwert, G.W. (1977): Asset returns and inflation, Journal of Financial Economics, 5(2), 115 – 146.

[6] Campbell, J.Y. (1987): Stock returns and the term structure, Journal of Finance Economics, 18(2), 373-399.

[7] M.Venkateshwarlu & T. Ramesh Babu (2011): Stock and Bond Price Dynamics-Evidence From An Emerging Economy, International Business & Economics Research Journal, 10 (9), 93-104.

[8] https://github.com/mrdbourke/tensorflow-deep-learning

[9]https://www.kaggle.com/sudalairajkumar/nifty-indices-dataset?select=NIFTY+NEXT+50.csv

[10] https://www.investing.com/

[11] https://www.incometaxindia.gov.in/charts%20%20tables/cost-inflation-index.htm

[12] https://in.finance.yahoo.com/