A Project Report on

FORECASTING A RUPEE'S PRICE AND STUDY THE FACTORS AFFECTING IT

A Major Project Report

Submitted in partial fulfillment of requirement for the Degree of

M.sc.(statistics) (with specialization in Industrial Statistics)

Mr. SHUBHAM SHRIMANT SHENDE (382799)

Mr. VISHAL MAHADEV NAYAKUDE (382778)

Ms. MAYURI ARUN PATHAK (382779)

DEPARTMENT OF STATISTICS,

SCHOOL OF MATHEMATICAL SCIENCES

JALGAON-425001

2021-2022

CERTIFICATE

This is to certify that **Mr. Shende Shubham Shrimant**, **Mr. Nayakude Vishal Mahadev**, **Ms. Pathak Mayuri Arun** are the students of M.Sc(Statistics) at Kavayitri Bahinabai Chaudhari North Maharahtra University, Jalgaon and have successfully completed their project entitled **FORECASTING A RUPEE'S PRICE AND STUDY FACTORS AFFECTING IT** as a part of M.Sc.(Statistics) program under the guidance and supervision during the academic year 2021-2022.

Place: Jalgaon
Date:

Mr. Manoj. C. Patil
(Project Guide)
Department Of Statistics
Kavayatri Bahinabai Chaudhari
North Maharastra University,Jalgaon

Acknowledgement

We are thankful to **Dr. R. L. Shinde**,(Head of Department of Statistics), Kavayitri Bahinabai Chaudhari North Maharashtra University, Jalgaon, project guide **Mr. Manoj. C. Patil**,(Assistance Prof of Department of Statistics) for their valuable guidance, suggestions, cooperation and constant encouragement which enable us to take every forward step in our project.

We are thankful to our miss **Dr. K. K. Kamalja**, who guided throughtout the project, friends and classmates to give us moral support we are also thankful to all those who helped us directly or indirectly for the project work.

Mr.Shende Shubham Shrimant,

Mr.Nayakude Vishal Mahadev,

Ms.Pathak Mayuri Arun

Contents

Chapter 1

Introduction

1.1 Motivation

Figure 1.1: Indian currency

This is the study regarding how the fluctions in Indian Rupee with respect to US dollar. The fluctuation happened everyday,then how it affects directly or indirectly to our economic situation and what are the main factors affecting on the fluctions and how? So, We are going for in detail study of the price fluctuaion and to check that factors are statistically significant for price fluctuation.

1.2 Objectives

- To study the trend of value of Indian Rupee comparing with US dollar.
- Fitting time series model for forcasting value of Indian Rupee comparing with one US dollar.
- Fitting LSTM (Long Short-Term Memory Model) Machine learning technique.
- To study which factors affecting on value of Indian Rupee.
- Fit regression model for predict value of Indian Rupee comparing with US dollar.

1.3 Introduction About Indian Rupee

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly. There are different types of models available for the analysis of time series data. We apply different types of models depending on the nature of the time series data set.

The history essentially starts from the time when the Britton Woods agreement was passed in 1944. This agreement determined the value of every currency in the world. Everyone was slowly adjusting to it during the time India gained independence. Since Independence in 1947, the value of INR has consistently gone down. As per the modern metric system, the value of INR to USD in 1913 should be 0.09 and if we keep the 1 USD = 1 INR argument then it the value went to 3.31 in 1948 and 3.67 in 1949, by 1970, INR was 7.50 to 1 USD.

In official records, 1 INR was never equal to 1 USD. When India gained independence, it had to accept the international metric system and the value of rupee changed at the same moment.

Since the time of Independence, Indian Rupee has been through a lot of situations that kept bringing down its value. Multiple Economic Crisis, Privatization, Devaluation and loans from The World Bank played a role in determining the value of 1 USD to INR over and over again. 20ptIn the last ten years during which period of the great recession of 2008 has passed the US federal fund rates have been flat at 0.25 percent. This also plays a role in the current value of INR to one USD.

History of Indian Rupees

- **Who named Indian Rupee?**

  The word 'rupee' has been derived from the Sanskrit word *rupyakam*, meaning a silver coin. It owes its origin to rupiya, issued by Sher Shah Suri in 1540-45. Today, the Reserve Bank of India issues currency under the RBI Act 1934.

- **When did Indian rupee started in India?**

  British India Issues commence with the Paper Currency Act of 1861 which gave the Government the monopoly of note issue in India. The first set of British India notes were the 'Victoria Portrait' Series issued in denominations of 10, 20, 50, 100, 1000.

Facts About Indian Rupee

- The Indian Rupee official symbol (Rs) was designed by the Indian academic and designer UdayaKumar. It was adopted only in 2010, after an open competition among Indian residents.
- INR is among the top 10 most counterfeited currencies of the world.
- Currency notes aren't made from regular paper Instead, they are made from pulp that contains cotton, balsam, special dyes and gelatin or polyvinyl alcohol.
- 15 languages features on each currency.
- Currently Indian coin are minted in four locations in our country-Mumbai, Noida ,Hyderabad and Kolkata. Each coin has unique shape right below the year that helps identify which city it was minted in.Mumbai represented by diamond, delhi by dot, Hyderabad by a star and Kolkata mint has no mark at all.

What is foreign exchange rate? Why currency fluctuates against dollar?

Foreign Exchange Market is simply a global market where the trading of currencies takes place. This market is decentralized in nature. In simple words, one currency is traded with another currency at a particular rate. The rate at which two particular currencies are exchanged is called the exchange rate.

The exchange rate is the value of one country's currency in relation to another country's currency. The exchange rate of any country's currency does not remain constant but rather keeps changing. Hence, the changing value of rupee remains a constant part of our everyday news.

For Example, the exchange rate of the Indian rupee in terms of the US dollar is approximately is 1 US dollar = 74.12 Indian Rupee. This means that if you want to buy a dollar from Foreign Exchange Market using Indian Rupee, you will need 74.12 rupees.

How do currencies move?

Most exchange of currencies takes place at banks. Currencies issued by different countries move through banks and it is here that most of the transactions take place. Eg.A person in Delhi having legal US dollar bills can get them converted to the Indian rupee at a particular exchange rate at a bank. This bank represents a small unit in the huge Foreign Exchange Market. The central bank of a country (RBI in India) also maintains a large reserve of foreign currency to deal with any kind of problems for Local currency in

the Foreign Exchange Market. As mentioned earlier authorities of a country do intervene when they sense a bad time for their currency.

They do this by adjusting the supply of a particular currency either directly or changing some other factors. As stated earlier, it is the supply and demand that determine the value of a currency. Since controlling the demand is barely in the hands of authority they influence the value of a currency by adjusting the supply of a currency in the market. US Dollar in India Rupee market

The demand for the US dollar is high since India is importing more products from the US than exporting. In such a scenario, the demand for the US dollar will increase since more dollars will be paid to the US while buying goods from them. And from the Indian side, more dollars will have to be bought from Foreign Exchange Market to pay for these goods.

As such demand for US dollars will increase as compared to the Indian rupee and hence their value. But if the value of the Indian Rupee falls a lot, the government will intervene. Immediately, they will try to reduce the supply of the Indian rupees (to compensate for the low demand). They will buy the Indian rupee from the market using US dollar reserves held by it.

As it buys more Indian currency using the US dollars, the supply of Indian currency decreases while that of the US increases, leading to an increase in the value of the rupee and a decrease in the value of a dollar. They can also influence the supply using other techniques.

In the long run, to sustain one currency at a good value, a country needs to increase the demand for its currency. This is a small example of the process, the actual process works at larger and at many levels. At the end of the day, it is the demand for a particular currency that determines its value in long run. And this demand is influenced by many factors like fiscal and monetary policies in a country, the amount of trade happening in a country, inflation, peoples' confidence in the political and economic conditions of a country.

How the value of a currency is determined?

In most countries of the world, the currency's value is determined by floating exchange rates. In this system, the value of a currency is determined by the basic economic concept of Demand and Supply.

A currency with more demand has a higher value. As the exchange of different currencies takes place in the Exchange market, the demand of each currency in the market determines its value. In this process of value determination, the government or authority of a country exercises no or little control. Even though government or the central bank of the respective country does intervene when the currency destabilises or performs poorly. But overall it is the mechanism of demand for a particular currency that determines its

value.

There is another system of value determination of currencies, even though not as prevalent as the above one. It is called pegged exchange system.Here, the value of a currency of a country is fixed with a particular currency. For Example, a country decides to fix their currency value with relation to the US dollar and determines their currency at 1/5 of a dollar.

Chapter 2

Exploratory analysis of value of Indian Rupee comparing with US dollar

Information about Dataset

Dataset is downloaded from www.investing.com
Data is from 1980-01-02 to 2022-04-29.

- Price : Closing Price of Day.
- Open : It is the price at which the financial security opens in the market when trading begins.
- High : It is the highest price at which a stock traded during a period.
- Low : It is the minimum price of a stock in a period.
- Change : Change of Prcentage compare with Yesterdays value.

[1]:
```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Dataset

[3]:
```
#reading dataset into dataframe as data1
data1 = pd.read_csv("C:/Users/Shubham/OneDrive/Desktop/data.
 ↪csv",index_col="Date")
data1.index=pd.to_datetime(data1.index)
data1[:]
```

[3]:
```
Date          Price   Open     High     Low      Change %
1980-01-02    8.000   8.000    8.000    8.000     0.00%
1980-01-03    7.950   7.950    7.950    7.950    -0.63%
1980-01-04    8.050   8.050    8.050    8.050     1.26%
...             ...     ...      ...      ...       ...
2022-04-27   76.605  76.768   76.806   76.490    -0.07%
2022-04-28   76.660  76.540   76.732   76.419     0.07%
2022-04-29   76.520  76.618   76.694   76.271    -0.18%


[10928 rows x 5 columns]
```

[4]:
```
#checking dimention of data1
data1.shape
```

[4]:
```
(10928, 5)
```

There are 10928 observations with 5 columns.

[5]:
```
#CHECKING FOR NULL VALUES in data1
data1.isnull().sum()
```

[5]:
```
Price       0
Open        0
High        0
Low         0
Change %    0
```

```
dtype: int64
```

There are no null values in whole Dataset.

[6]:
```
#CHECKING COUNT OF UNIQUE VALUES IN COLUMNS
data1.index.value_counts()
```

[6]:
```
1996-01-23    1
2017-07-28    1
1991-04-25    1
              ..
1993-07-26    1
2008-08-13    1
1985-11-14    1
Name: Date, Length: 10928, dtype: int64
```

For each date there is unique value of price.

2.1 Trend analysis

Now we will try to find the trend by using simple line plot graph.

[7]:
```
plt.figure(figsize=(5,5),dpi=200)
plt.plot(data1.index,data1['Price'])
plt.title('Line Plot of Price ')
plt.xlabel('Date')
plt.ylabel(' Price INR (â,¹)')
plt.show()
```

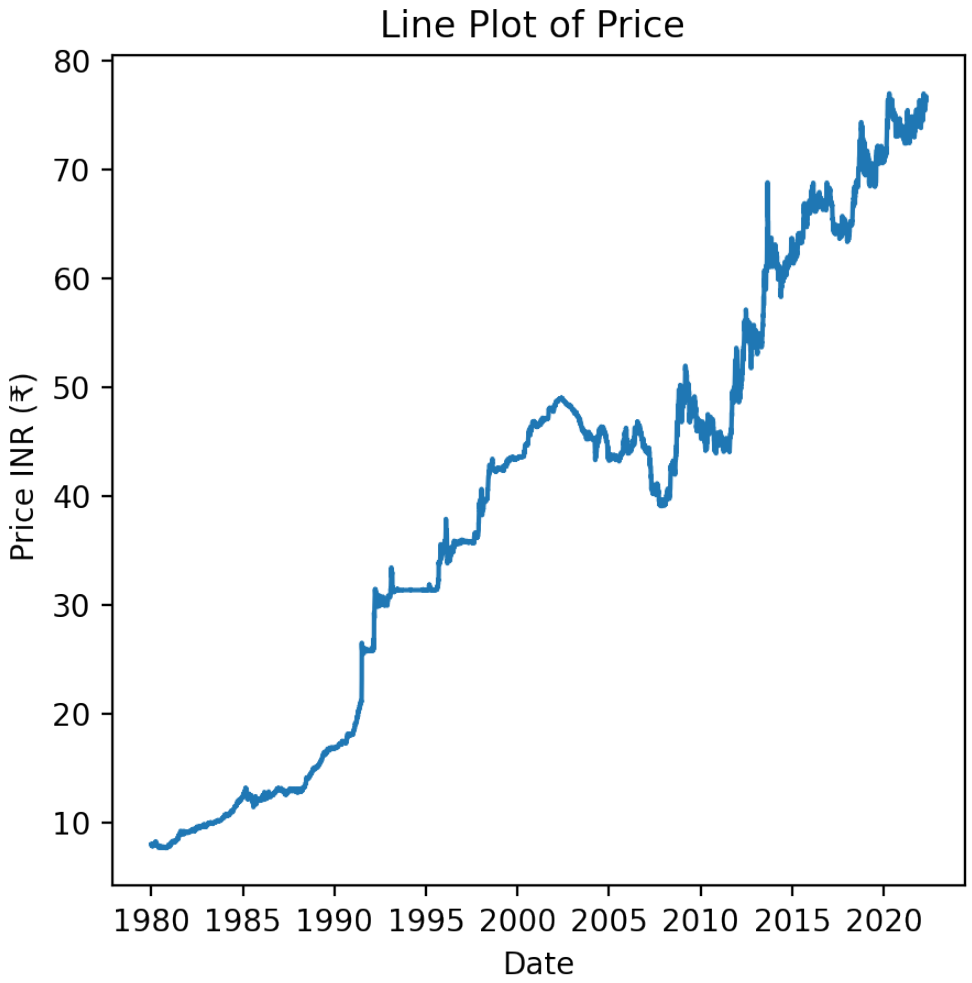

Figure 2.1.1: line plot

From line plot we can see that there is increasing trend in value of Indian rupee.

2.2 Auto-Correlation Function (ACF) plot

ACF (auto-correlation function) is a function which gives us values of auto-correlation of any series with its lagged values. We plot these values along with the confidence band. In simple terms, it describes how well the present value of the series is related with its past values.

In time series analysis, the partial autocorrelation function (PACF) gives the partial correlation of a stationary time series with its own lagged values, regressed the values of the time series at all shorter lags.

[8]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(data1.Price,lags=10927)

plt.xlabel('Lags')

plt.ylabel('autocorrelation')

plt.title('Autocorrelation of Price ')

plt.show()
```

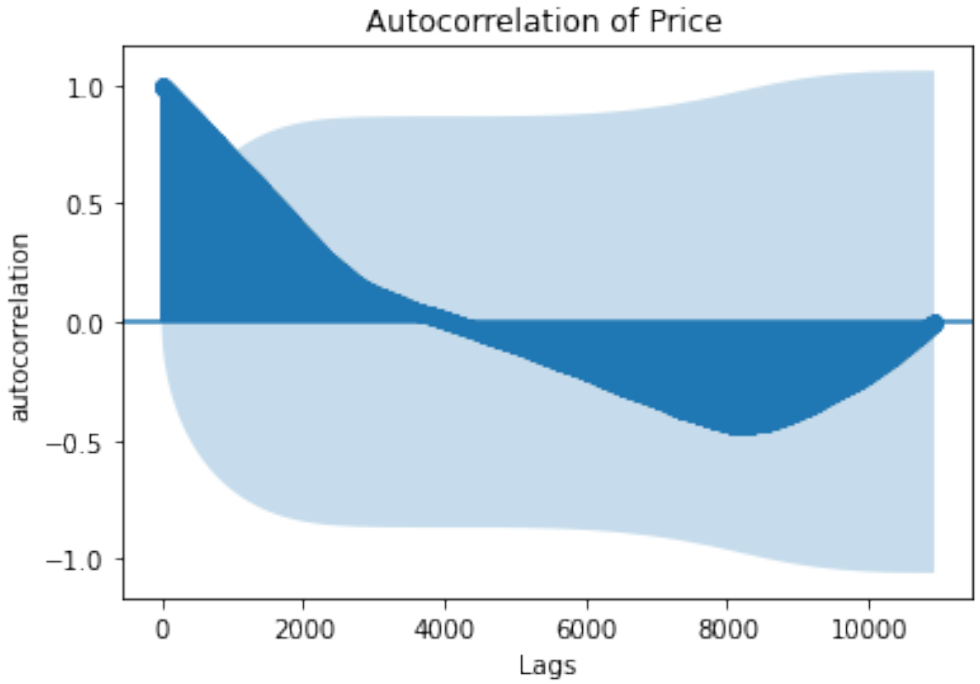

Figure 2.2.1: ACF plot without differencing

We have ploted the ACF plot of our series and it goes to zero in very slowly, suggesting that series is non-stationary.

2.3 Partial Autocorrelation Function (PACF) Plot

In time series analysis, the partial autocorrelation function (PACF) gives the partial correlation of a stationary time series with its own lagged values, regressed the values of the time series at all shorter lags.

[9]:
```
plot_pacf(data1.Price,lags=100)
plt.xlabel('Lags')
plt.ylabel('Partial autocorrelation')
plt.title('Partial Autocorrelation of Price ')
plt.show()
```

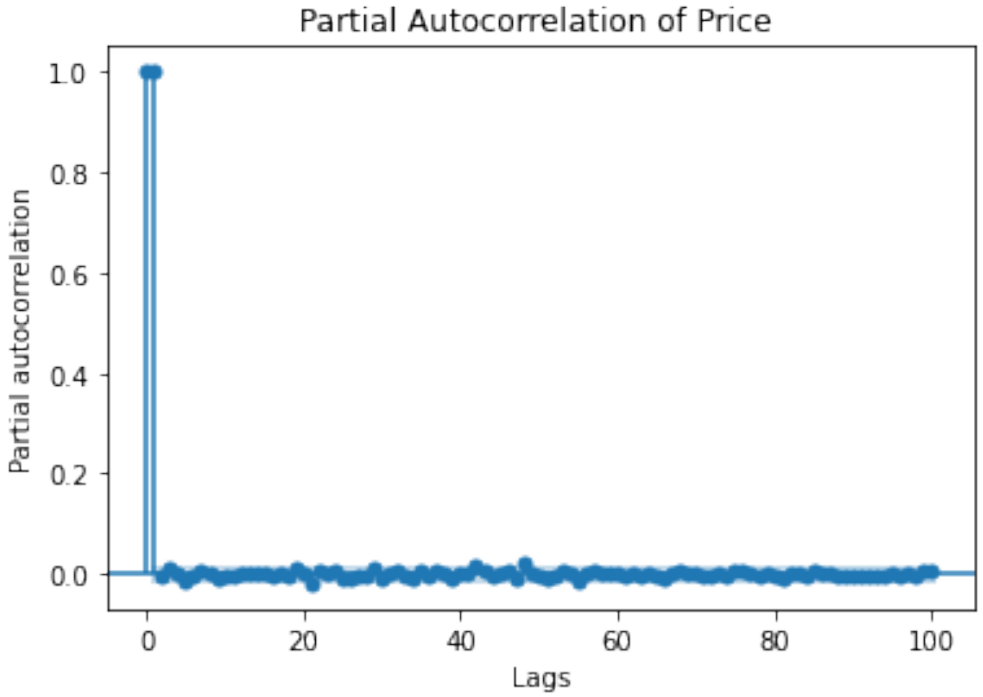

Figure 2.3.1: PACF plot without differencing

We have PACF plot of our series, suggesting that series is non-stationary.

Now, we check for the stationarity of the series by different tests.

Time series is said to be stationary if they do not have trend or seasonal effects. Summary statistics calculated on the time series are consistent over time, like the mean or the variance of the observations. When the time series is not stationary then it is called non-stationary time series.

When a time series is stationary, it can be easier to model. Statistical modelling methods assume or require the time series to be stationary to be effective. We use ADF and

KPSS test for this purpose.

- **ADF test:**

The Augmented Dickey-Fuller test is a type of statistical test called a unit root test.

Null Hypothesis (H0): Series has a unit root, that is series is not stationary.

Alternate Hypothesis (H1): Series does not have a unit root, that is series is stationary.

Decision

If p-value is less than 0.05 level of significance, Then we reject the null hypothesis at 5% Level Of Significance.

[10]:
```
from statsmodels.tsa.stattools import adfuller
result = adfuller(data1['Price'])
print(result)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
(-0.05798326810772723, 0.9534660556560518, 34, 10893, {'1%':
-3.4309504628347613, '5%': -2.8618053712501084, '10%': -2.566911251984947},
-6303.034128557269)
ADF Statistic: -0.057983
p-value: 0.953466
```

We have applied the ADF test on the series with no differencing and the p-value is 0.888266 which is more than 0.05 level of significance, hence we failed to reject the null hypothesis and conclude that the series is non stationary.

- **KPSS test:**

The KPSS test, short for, Kwiatkowski-Phillips-Schmidt-Shin (KPSS), is a type of Unit root test that tests for the stationarity of a given series around a deterministic trend.

Null Hypothesis (H0): Series does not have a unit root, that is series is stationary.

Alternate Hypothesis (H1): Series has a unit root, that is series is not stationary.

If p-value is less than 0.05 level of significance, Then we reject the null hypothesis at 5% Level Of Significance.

[11]:
```
# KPSS test
from statsmodels.tsa.stattools import kpss
result = kpss(data1['Price'])
print(result)
print('KPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#shows series is non stationary
```

```
(25.67483149718737, 0.01, 39, {'10%': 0.347, '5%': 0.463, '2.5%': 0.574,␣
↪'1%':
0.739})
KPSS Statistic: 25.674831
p-value: 0.010000
```

We have applied the KPSS test on the series with no differencing and the p-value is 0.01 which is less than 0.05 level of significance, hence we reject the null hypothesis and conclude that the series is non-stationary.

As we can see that, ADF and KPSS tests give different results, conclude that the series is non-stationary. For making the series stationary, we need differencing of it.

Chapter 3

Forecasting the Price of Rupee using Time Series Model

3.1 BOX-JENKINS Methodology

The approach starts with the assumption that the process that generated the time series can be approximated using an ARMA model if it is stationary or an ARIMA model if it is non-stationary.

1. Identification
2. Estimation
3. Diagnostic Checking

It is an iterative process, so that as new information is gained during diagnostics, we can circle back to step 1 and incorporate that into new model classes.
For this we reffered the book: Brockwell and Davis [2002]

1. IDENTIFICATION: The identification step is further broken down into:

- Assess whether the time series is stationary, and if not, how many differences are required to make it stationary.
- Identify the parameters of an ARMA model for the data.

1.1 DIFFERENCING:

- Unit Root Tests. Use unit root statistical tests on the time series to determine whether or not it is stationary. Repeat after each round of differencing.

- Avoid over differencing. Differencing the time series more than is required can result in the addition of extra serial correlation and additional complexity.

1.2 CONFIGURING AR AND MA: Two diagnostic plots can be used to help choose the p and q parameters of the ARMA or ARIMA. They are:

- Autocorrelation Function (ACF). The plot summarizes the correlation of an observation with lag values. The x-axis shows the lag and the y-axis shows the correlation coefficient between -1 and 1 for negative and positive correlation.
  $\rho_k = E(Y_t\mu)(Y_{t+k}\mu)]/(E[(Y_t\text{-}\mu)^2]E[(Y_{t+k}\text{-}\mu)^2])$
- Partial Autocorrelation Function (PACF). The plot summarizes the correlations for an observation with lag values that is not accounted for by prior lagged observations. Both plots are drawn as bar charts showing the 95% and 99% confidence intervals as horizontal lines. Bars that cross these confidence intervals are therefore more significant and worth noting.
- The model is AR if the ACF trails off after a lag and has a hard cut-off in the PACF after a lag. This lag is taken as the value for p.
- The model is MA if the PACF trails off after a lag and has a hard cut-off in the ACF after the lag. This lag value is taken as the value for q.
- The model is a mix of AR and MA if both the ACF and PACF trail off.

2. ESTIMATION: We use the data to estimate the parameters coefficients of the model.

3. DIAGNOSTIC CHECKING: The idea of diagnostic checking is to look for evidence that the model is not a good fit for the data e.g. residual errors diagnostics.

FIRST ORDER DIFFERENCING OF THE SERIES: In the below table, the column 'daily_vaccinations_with_diff_1' shows the series with order of differencing 1.

$y_t = y_t - y_{t-1}$

[12]:
```
#first order differencing of raw data
data1['Price_with_diff_1'] = data1['Price'] - data1['Price'].shift(1)
data1[:]
```

[12]:

Date	Price	Open	High	Low	Change %	Price_with_diff_1
1980-01-02	8.000	8.000	8.000	8.000	0.00%	NaN
1980-01-03	7.950	7.950	7.950	7.950	-0.63%	-0.050

```
1980-01-04   8.050   8.050   8.050   8.050    1.26%              0.100
...            ...     ...     ...     ...      ...                ...
2022-04-27  76.605  76.768  76.806  76.490   -0.07%             -0.055
2022-04-28  76.660  76.540  76.732  76.419    0.07%              0.055
2022-04-29  76.520  76.618  76.694  76.271   -0.18%             -0.140
[10928 rows x 6 columns]
```

After differencing we got new series.
Line plot of the time series with order of Differencing 1 and it's ACF Plot:

[13]:
```
#ploting the time series after first order differning
plt.figure(figsize=(5,5),dpi=100)
plt.plot(data1.index,data1['Price_with_diff_1'])
plt.title('Line Plot of Price after Differencing ')
plt.xlabel('Date')
plt.ylabel(' Price INR (â‚¹)')
plt.show()
```

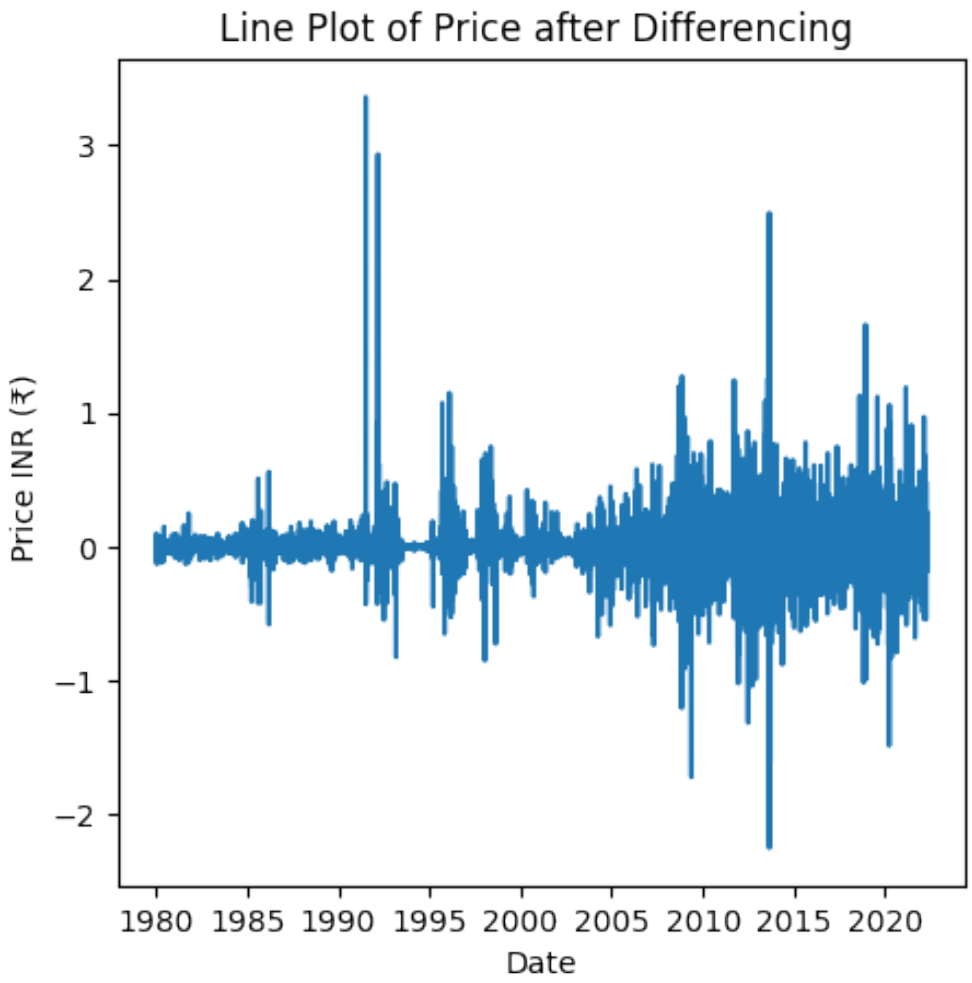

Figure 3.1.0.1: Line Plot of Price after Differencing

Trend and Seasonal componenet is not present in plot now.

[14]:
```
#acf plot of dataframe of diff 1
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(data1['Price_with_diff_1'][1:], lags = 50)
plt.xlabel('Lags')
plt.ylabel('autocorrelation')
plt.title('Autocorrelation of Price ')
plt.show()
```

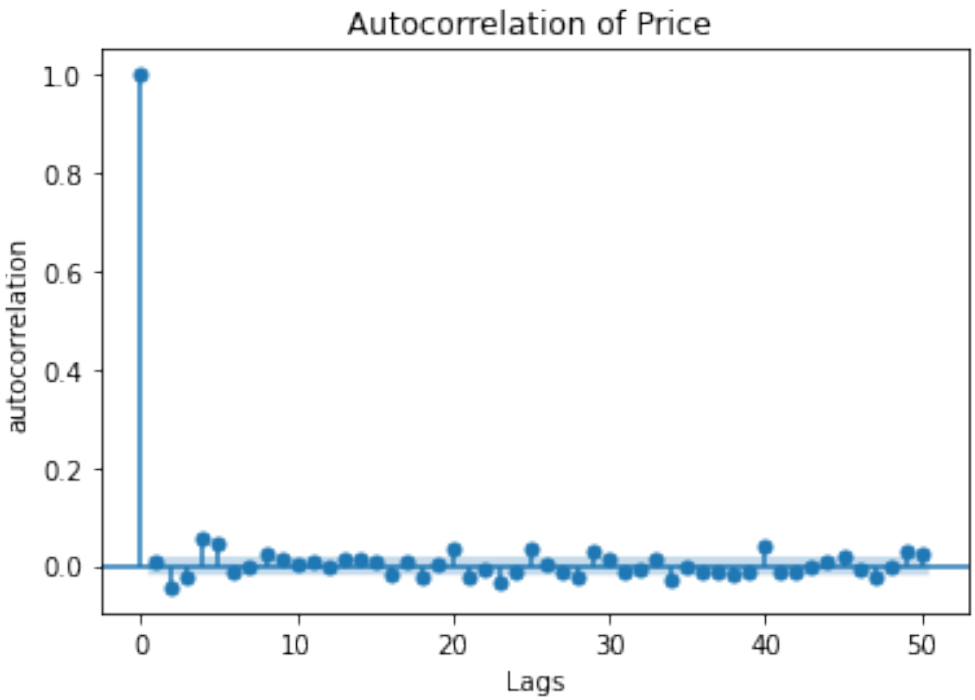

Figure 3.1.0.2: ACF Plot of Price after Differencing

Here, by using ACF plot we can say that after order one diffrencing time series becomes stationary.

[15]:
```
#pacf plot of dataframe of diff 1
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(data1['Price_with_diff_1'][1:], lags = 50)
plt.xlabel('Lags')
plt.ylabel('Partial autocorrelation')
plt.title('Partial Autocorrelation of Price ')
plt.show()
```

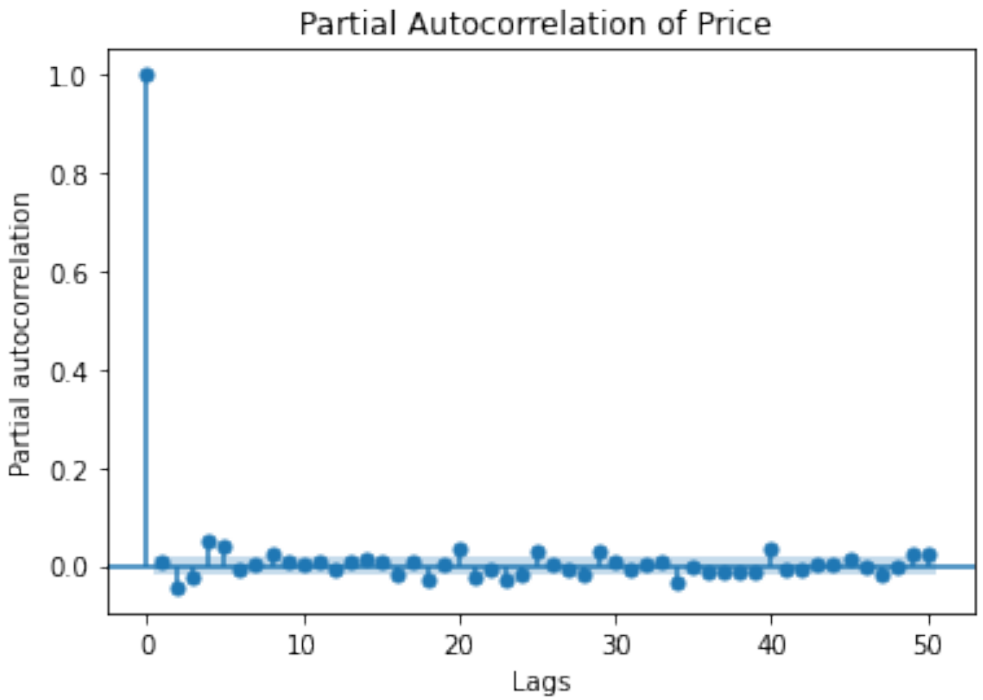

Figure 3.1.0.3: PACF Plot of Price after Differencing

Here, by using PACF plot we can say that after order one diffrencing time series becomes stationary.

By looking at the ACF plot, we can see that it cuts off the region and goes to zero quickly which is kind of sign of stationarity of the series. Now, we will again apply the ADF and KPSS tests to check the stationarity of the series with order of differencing 1.

TESTING THE STATIONARITY OF THE SERIES WITH ORDER OF DIFFERENCING IS 1:

- **ADF test**:

Null Hypothesis (H0): Series has a unit root, that is the series is not stationary.

Alternate Hypothesis (H1): Series does not have a unit root, that is the series is stationary.

[16]:
```
#ADF test on data with diff 1
from statsmodels.tsa.stattools import adfuller
result = adfuller(data1['Price_with_diff_1'][1:])
print(result)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#shows series is stationary
```

```
(-16.84202070210564, 1.1361339886369506e-29, 39, 10887, {'1%':
-3.430950793837518, '5%': -2.861805517520163, '10%': -2.566911329844231},
-6313.066229231874)
ADF Statistic: -16.842021
p-value: 0.000000
```

We have applied the ADF test on the series with first order differencing and the p-value is 0.00000 which is less than 0.05 level of significance, hence we reject the null hypothesis and conclude that the series is stationary.

- **KPSS test**:

Null Hypothesis (H0): Series does not have a unit root, that is the series is stationary.

Alternate Hypothesis (H1): Series has a unit root, that is the series is non-stationary.

[17]:
```
# KPSS test
from statsmodels.tsa.stattools import kpss
result = kpss(data1['Price_with_diff_1'][1:])
print(result)
print('KPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
(0.0698907349682722, 0.1, 39, {'10%': 0.347, '5%': 0.463, '2.5%': 0.574,␣
 ↪'1%':
0.739})
KPSS Statistic: 0.069891
p-value: 0.100000
```

We have applied the KPSS test on the series with first order differencing and the p-value is 0.1 which is greater than 0.05 level of significance, hence we fail to reject the null hypothesis and conclude that the series is stationary.

With the first order of differencing our series has become stationary. We can proceed for the modelling stage.

3.2 Configuring AR & MA order and Estimating AR & MA Parameters:

Further, as we got our series is stationary with first order of differencing, we find out the appropriate order of (p, q). We'll fit the ARIMA model of order (p, d=1, q) and calculate the value of Akaike Information Criteria and consider the corresponding value of p and q for which AIC is minimum. For model fitting we have referred one reasearch paper : Babu and Reddy [2015]

- **ARIMA:**

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

- AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.
- I: Integrated. The use of differencing of raw observations (i.e. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- MA: Moving Average. A model that uses the dependency between an observation and residual errors from a moving average model applied to lagged observations.

  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 differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

ARIMA(p,d,q) model is given by the equation below:

$$y_t = c + \phi_1 y_{t-1} + \ldots .. + \phi_p y_{t-p} + \theta_1 e_{t-1} + \ldots . + \theta_q e_{t-q} + e_t$$

where:

- y_t = the variable that explained in time t
- c = constant or intercept
- ϕ = coefficient of AR polynomial (p parameters)
- θ =coefficient of MA polynomial (q parameters)
- e_t = Residuals or errors in time t

[18]:
```
#Splitting the Data
train = data1['Price'][:8743]
test = data1['Price'][8744:]
```

We have splitted our dataset as 80% used for train the model and 20% used for test the model.

[19]:
```
import pmdarima as pm
model = pm.auto_arima(data1.Price[:
 ↪8743],start_p=1,start_q=1,test='adf',max_p=4,max_q=4,␣
 ↪m=1,max_d=1,seasonal=False,start_P=0, D=0,start_Q=0, max_P=0, max_D=0,␣
 ↪max_Q=0,trace=True,error_action='ignore',␣
 ↪suppress_warnings=True,stepwise=True)
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-6717.868, Time=6.39 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-6716.503, Time=1.20 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-6718.015, Time=1.97 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-6718.243, Time=2.64 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-6706.134, Time=0.57 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-6723.283, Time=2.05 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-6716.981, Time=2.92 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-6728.390, Time=3.67 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-6731.052, Time=11.82 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-6758.993, Time=18.45 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-6775.250, Time=12.78 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-6719.123, Time=7.68 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-6780.958, Time=16.62 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-6732.860, Time=5.77 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=-6768.579, Time=16.37 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-6752.557, Time=20.60 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-6771.977, Time=15.83 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=-6773.910, Time=24.60 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=-6770.842, Time=8.51 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0] intercept
Total fit time: 180.508 seconds
```

Here we can see that the order of our ARIMA model is (3,1,2)(0,0,0)[0].
where,
(3,1,2) are (p,d,q)
(0,0,0) are seasonal (P,D,Q) and [0] is seasonal lag

SARIMAX Results

Dep. Variable:	y	No. Observations:	8743
Model:	SARIMAX(3, 1, 2)	Log Likelihood	3397.479
Date:	Sun, 29 May 2022	AIC	-6780.958
Time:	11:30:07	BIC	-6731.427
Sample:	0 - 8743	HQIC	-6764.079
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0081	0.002	3.276	0.001	0.003	0.013
ar.L1	0.6643	0.007	90.845	0.000	0.650	0.679
ar.L2	-0.9923	0.006	-170.044	0.000	-1.004	-0.981
ar.L3	0.0311	0.004	7.925	0.000	0.023	0.039
ma.L1	-0.6462	0.007	-96.228	0.000	-0.659	-0.633
ma.L2	0.9609	0.007	145.264	0.000	0.948	0.974
sigma2	0.0269	7.8e-05	345.071	0.000	0.027	0.027

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	1310027.34
Prob(Q):	0.99	Prob(JB):	0.00
Heteroskedasticity (H):	7.44	Skew:	2.23
Prob(H) (two-sided):	0.00	Kurtosis:	62.80

The obtained ARIMA (3,1,2) model with AIC (-6780.958) is given by:

$$Y\_t = (00081) + (06643) * Y\_t - 1 - (09923) * Y\_t - 2 + (00311) * Y\_t - 3 + e\_t - (0. \hookrightarrow 6462) * e\_t - 1 + (09609) * e\_t - 2 + e\_t$$

where e_t = Residuals or errors in time t

Function for ARIMA model you can fit as many of arima models by using function and then you eill select proper model having minimum AIC.

[20]:
```
#ARIMA Model
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima_model import ARMAResults
def arima(p,d,q):
    model = ARIMA(data1.Price[:8743], order=(p,d,q))
    model_fit = model.fit(disp=0)
    print(model_fit.summary
```

[21]:
```
arima(3,1,2)
```

ARIMA Model Results

Dep. Variable:	D.Price	No. Observations:	8742
Model:	ARIMA(3, 1, 2)	Log Likelihood	3397.508
Method:	css-mle	S.D. of innovations	0.164
Date:	Sun, 29 May 2022	AIC	-6781.015
Time:	11:30:15	BIC	-6731.484
Sample:	1	HQIC	-6764.137

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0062	0.002	3.493	0.000	0.003	0.010
ar.L1.D.Price	0.6623	0.018	35.901	0.000	0.626	0.698
ar.L2.D.Price	-0.9918	0.015	-67.646	0.000	-1.021	-0.963
ar.L3.D.Price	0.0305	0.011	2.770	0.006	0.009	0.052
ma.L1.D.Price	-0.6442	0.015	-42.447	0.000	-0.674	-0.614
ma.L2.D.Price	0.9598	0.018	54.823	0.000	0.925	0.994

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.3245	-0.9610j	1.0143	-0.1982
AR.2	0.3245	+0.9610j	1.0143	0.1982
AR.3	31.8639	-0.0000j	31.8639	-0.0000
MA.1	0.3356	-0.9640j	1.0207	-0.1967
MA.2	0.3356	+0.9640j	1.0207	0.1967

By using function we got ARIMA(3,1,2) model.

The obtained ARIMA (3,1,2) model with AIC (-6781.015) is given by:

$Y\_t = (00081) + (06623) * Y\_t - 1 - (09918) * Y\_t - 2 + (00305) * Y\_t - 3 + e\_t - (0.$
↪$6442) * e\_t - 1 + (09598) * e\_t - 2 + e\_t$

where e_t = Residuals or errors in time t

Estimate all parameters using auto_arima

This will give all p,d,q parameters as well as if there is seasonality then also it wil give seasonal P,D,Q parameters.

[22]:
```
#fitting arima models
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima_model import ARMAResults
import re
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot
import pylab
import scipy.stats as stats
```

[23]:
```
#reading dataset into dataframe as data1
data1 = pd.read_csv("C:/Users/Shubham/OneDrive/Desktop/data.
 ↪csv",index_col="Date")
data1.index=pd.to_datetime(data1.index)
```

[24]:
```
#Splitting the Data
train = data1['Price'][:8743]
test = data1['Price'][8744:]
```

[25]:
```
from statsmodels.tsa.arima.model import ARIMA
from pmdarima import auto_arima
model=auto_arima(data1.Price[:
 ↪8743],start_p=1,seasonal=False,D=1,trace=True,error_action='ignore',stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-6716.981, Time=3.12 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-6716.503, Time=1.18 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-6718.015, Time=2.04 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-6718.243, Time=2.79 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-6706.134, Time=0.57 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-6717.868, Time=6.50 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-6723.283, Time=2.07 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-6728.390, Time=3.87 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-6731.052, Time=12.01 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-6758.993, Time=18.25 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-6775.250, Time=12.69 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-6719.123, Time=7.49 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-6780.958, Time=16.23 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-6732.860, Time=5.82 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=-6768.579, Time=19.23 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-6752.557, Time=19.21 sec
```

```
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-6771.977, Time=13.71 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=-6773.910, Time=22.10 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=-6770.842, Time=8.03 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0] intercept
Total fit time: 176.957 seconds
```

[27]:
```
from statsmodels.tsa.arima_model import ARIMA
model1 = ARIMA(data1.Price[:8743], order=(3,1,2))
model_fit = model1.fit()
print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:       D.Price           No. Observations:     8742
Model:               ARIMA(3, 1, 2)    Log Likelihood        3397.508
Method:              css-mle           S.D. of innovations   0.164
Date:                Sun, 29 May 2022  AIC                  -6781.015
Time:                11:30:15          BIC                  -6731.484
Sample:              1                 HQIC                 -6764.137
==============================================================================
                 coef    std err          z      P>|z|      [0.025    0.975]
------------------------------------------------------------------------------
const            0.0062      0.002      3.493      0.000       0.003    0.010
ar.L1.D.Price    0.6623      0.018     35.901      0.000       0.626    0.698
ar.L2.D.Price   -0.9918      0.015    -67.646      0.000      -1.021   -0.963
ar.L3.D.Price    0.0305      0.011      2.770      0.006       0.009    0.052
ma.L1.D.Price   -0.6442      0.015    -42.447      0.000      -0.674   -0.614
ma.L2.D.Price    0.9598      0.018     54.823      0.000       0.925    0.994
                                    Roots
==============================================================================
```

```
            Real       Imaginary         Modulus      Frequency
------------------------------------------------------------------------------
AR.1        0.3245      -0.9610j          1.0143       -0.1982
AR.2        0.3245      +0.9610j          1.0143        0.1982
AR.3       31.8639      -0.0000j         31.8639       -0.0000
MA.1        0.3356      -0.9640j          1.0207       -0.1967
MA.2        0.3356      +0.9640j          1.0207        0.1967
------------------------------------------------------------------------------
```

3.3 Forecasting the value of Indian rupee

Now we are ready for forecasting stage. We forecasted the observations and show these with it's actual values.

[28]:
```
y_pred=pd.Series(model_fit.forecast(2184)[0], index=data1.Price[8744:
 ↪10928].index)
y_true=data1.Price[8744:]
print (np.array(y_pred))
print (np.array(y_true))
```

```
[62.22044657 62.30002671 62.34699521 ... 75.80894618 75.81515324
 75.8213603 ]
[62.02  62.1   62.15  ... 76.605 76.66  76.52 ]
```

[29]:
```
df=pd.DataFrame()
df['Actual Value']=y_true
df['Predicted Value']=y_pred
```

Table of Actual value and Predicted value

[30]:
```
df
```

[30]:
```
Date            Actual Value  Predicted Value
2013-12-17        62.020        62.220447
2013-12-18        62.100        62.300027
2013-12-19        62.150        62.346995
...                  ...             ...
2022-04-27        76.605        75.808946
2022-04-28        76.660        75.815153
2022-04-29        76.520        75.821360
--------------------------------------------------------------------------------
[2184 rows x 2 columns]
```

[31]:
```
df.to_csv('ARIMA predicted output.csv')
```

Plot of forecasted value

[32]:
```
train = data1[:8743]
valid = data1[8743:]
valid['Predictions'] = y_pred
plt.figure(figsize=(16,8))
plt.title('Plot of forecasted value ')
plt.xlabel('Date')
plt.ylabel('Price INR (â,¹)')
plt.plot(train['Price'])
plt.plot(valid[['Price','Predictions']])
plt.legend(['Train','Observed','Predictions'], loc='lower right')
plt.show()
```

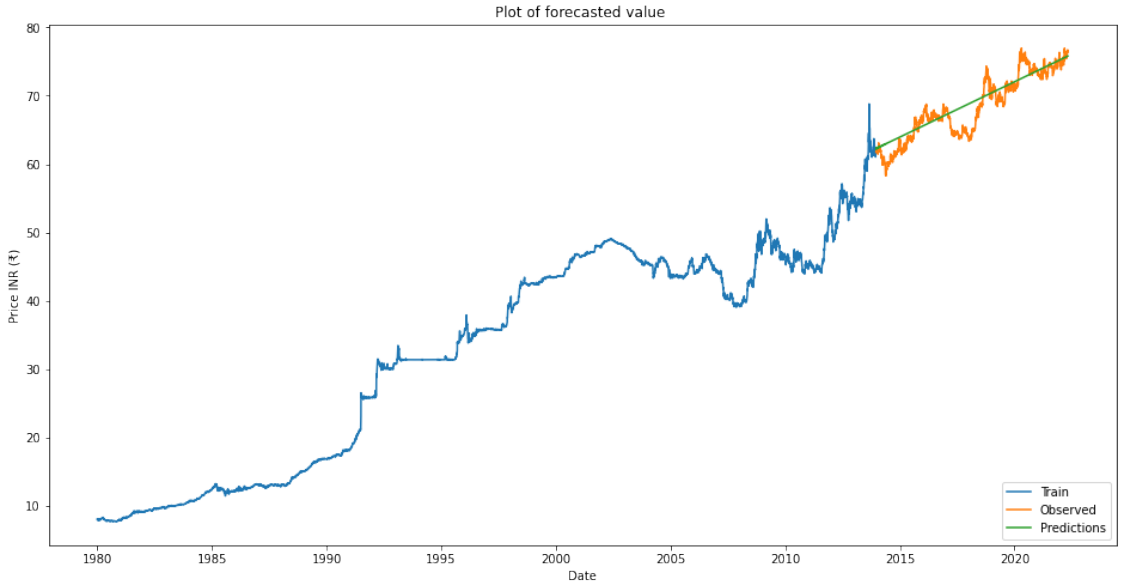

Figure 3.3.0.1: Plot of forecasted values of Price

Fig 5.3.1 is shows the plot predicted price as well as valid or original prices.

[33]:
```
plt.figure(figsize=(16,8))
plt.plot(valid[['Price','Predictions']])
plt.title('Plot of forecasted value ')
plt.legend(['Valid','Predictions'], loc='lower right')
plt.xlabel('Date')
plt.ylabel('Price INR (â,¹)')
plt.show()
```

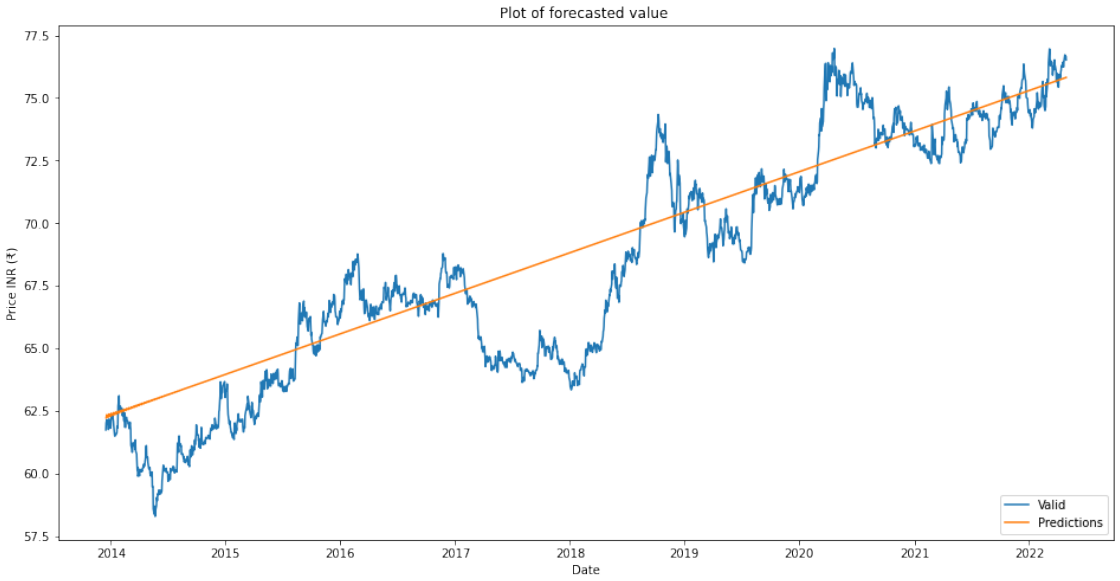

Figure 3.3.0.2: Plot of forecasted values of Price

3.4 Diagnostic Checking

- MAPE (Mean Absolute Percentage Error)

  The Mean Absolute Percentage Error (MAPE), also known as Mean Absolute Percentage Deviation (MAPD), is a measure of prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$\text{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

  where At is the actual value and Ft is the forecast value. Their difference is divided by the actual value At. The absolute value in this ratio is summed for every forecasted point in time and divided by the number of fitted points n.

- MAE (Mean Absolute Error)

  In statistics, Mean Absolute Error (MAE) is a measure of errors between paired observations expressing the same phenomenon. MAE is calculated as the sum of absolute errors divided by the sample size:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

  It is thus an arithmetic average of the absolute errors $|e_i| = |y_i - x_i|$, where y_i is the prediction and x_i the true value. Note that alternative formulations may include relative frequencies as weight factors. The mean absolute error uses the same scale as the data being measured. This is known as a scale-dependent accuracy measure and therefore cannot be used to make comparisons between series using different scales. The mean absolute error is a common measure of forecast error in time series analysis, sometimes used in confusion with the more standard definition of mean

absolute deviation.

- MPE (Mean Percentage Error)

  The Mean Percentage Error (MPE) is the computed average of percentage errors by which forecasts of a model differ from actual values of the quantity being forecast.
  The formula for the mean percentage error is:

$$\text{MPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{a_t - f_t}{a_t}$$

  where a_t is the actual value of the quantity being forecast, f_t is the forecast, and n is the number of different times for which the variable is forecast.

  Because actual rather than absolute values of the forecast errors are used in the formula, positive and negative forecast errors can offset each other; as a result the formula can be used as a measure of the bias in the forecasts.

  A disadvantage of this measure is that it is undefined whenever a single actual value is zero.

- RMSE (Root Mean Square Error)

  It is the square root of the mean square error. It is also always positive and is in the range of the data.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} [y_i - x_i]^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n} (e_i)^2}{n}}$$

  Where, y_i is forecased value
  x_i is actual value, and
  n is total number of values in test set.
  It is in the power of unity and hence is more interpretable as compared to MSE. RMSE is also more penalizing for larger errors.

[34]:
```
mape = np.mean(np.abs(y_pred - y_true) /np.abs(y_true)) # Mean absolute
 ↪percentage error -
mae = np.mean(np.abs(y_pred - y_true)) # Mean absolute error
mpe = np.mean((y_pred - y_true)/y_true) # Mean percentage error
rmse = np.mean((y_pred - y_true)**2)**.5 # RMSE
corr = np.corrcoef (y_pred, y_true) [0,1]
# Correlation Coefficient
mins = np.amin(np.hstack([y_pred[:,None], y_true[:,None]]), axis=1)
maxs = np.amax(np.hstack([y_pred[:,None], y_true[:,None]]), axis=1)
minmax = 1 - np.mean(mins/maxs)    # minmax
import pprint
```

```
pprint.pprint({'mape':mape,'mae':mae,'mpe':mpe,'rmse':rmse,'corr':
 ↪corr,'minmax' :minmax})
```

```
{'corr': 0.9221536375600266,
 'mae': 1.5866697328508435,
 'mape': 0.02387056601358746,
 'minmax': 0.023097222903119108,
 'mpe': 0.012199567557602268,
 'rmse': 2.051420628815606}
```

If model have MAPE less than 5% then model is good fitted.
For our model MAPE is 2.38%.

Residual analysis

[35]:
```
# Plot residual errors
residuals = model_fit.resid
plt.figure(figsize=(16,8))
fig, ax = plt.subplots(1,2)
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```

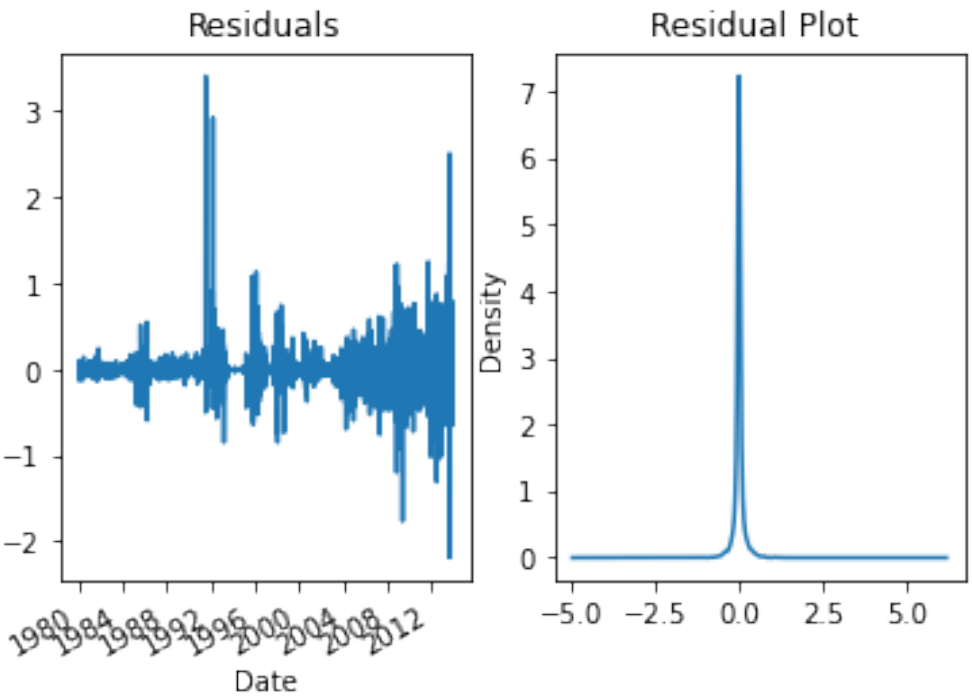

Figure 3.4.0.1: Plot of Residuals

By using simple line plot of residuals, We can say that residuals are stationary.

[36]:
```
df_res=pd.DataFrame()
df_res['Residuals']=residuals
df_res
```

[36]:

Date	Residuals
1980-01-03	-0.056207
1980-01-04	0.094927
1980-01-07	-0.009665
1980-01-08	-0.003900
1980-01-09	-0.054411
...	...
2013-12-09	-0.331310
2013-12-10	-0.172435
2013-12-11	0.180877

```
2013-12-12    0.622931
2013-12-13    0.426498

[8742 rows x 1 columns]
```

[37]:
```
# acf plot of residuals
plot_acf(residuals, lags = 50)
plt.xlabel('Lags')
plt.ylabel('autocorrelation')
plt.title('Autocorrelation plot ')
plt.show()
```

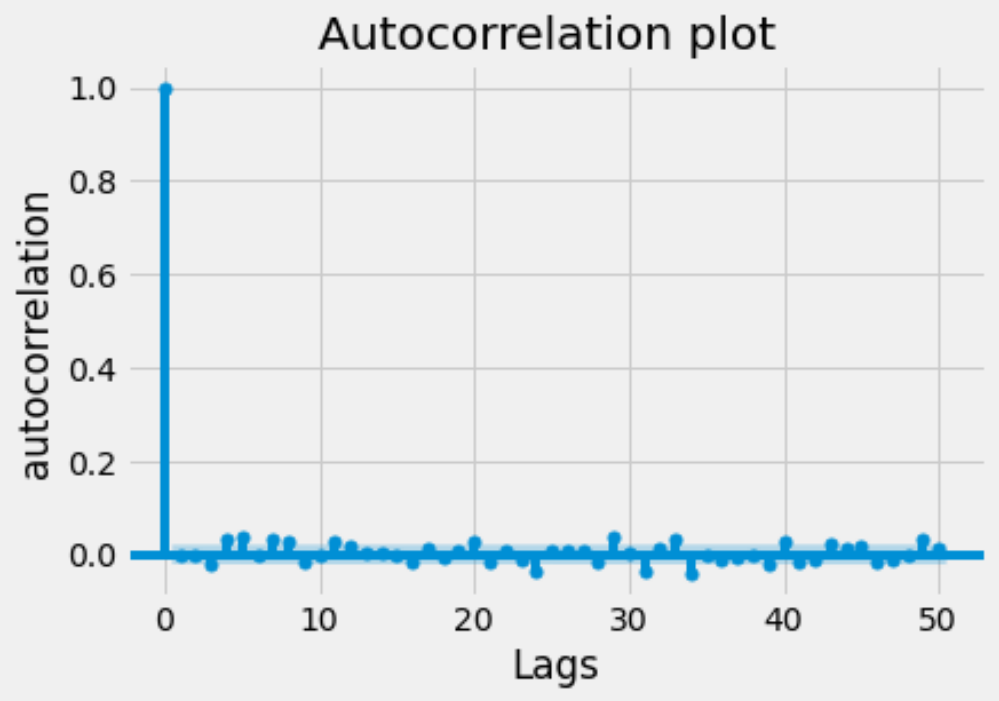

Figure 3.4.0.2: ACF Plot of residuals

Series of residuals is stationary. We will check this by using ADF test.

ADF test:

Null Hypothesis (H0): series has a unit root, that is the series is not stationary.

Alternate Hypothesis (H1): series does not have a unit root, that is the series is stationary.

[38]:
```
from statsmodels.tsa.stattools import adfuller
result = adfuller(df_res['Residuals'][1:])
print(result)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#shows series is stationary
```

```
(-15.765181315573749, 1.1787994856207317e-28, 33, 8707, {'1%':
-3.431101260930503, '5%': -2.861872007212997, '10%': -2.566946722477775},
-6802.044036599882)
ADF Statistic: -15.765181
p-value: 0.000000
```

the p-value is 0.00000 which is less than 0.05 level of significance, hence we reject the null hypothesis and conclude that the series is stationary.

[39]:
```
model.plot_diagnostics(figsize=(15,8))
plt.show
```

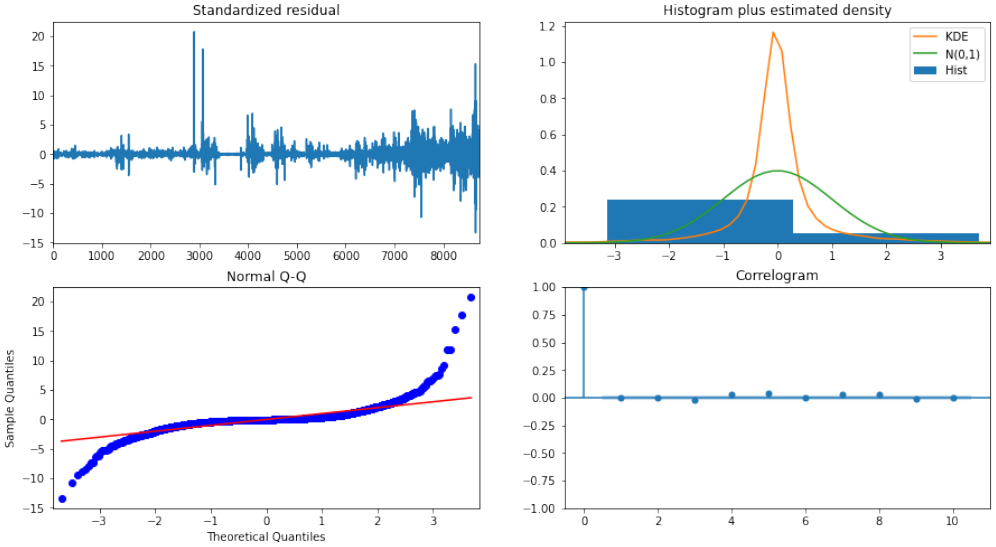

Figure 3.4.0.3: Plot of residual diagnostics

residuals are stationary but not normal and by using correlogram we can say that residuals are not correlated.

LJUNG BOX TEST TO CHECK WHETHER RESIDUALS ARE CORRELATED AT DIFFERENT LAGS: Ljung Box test is a test for testing the absence of serial autocorrelation, up to a specified lag k.

Null hypothesis H0: residuals are uncorrelated.

Alternate hypothesis Ha: residuals are correlated.

$Q(m) = n * (n+2) \sum_{j=1}^{m} \frac{r_j^2}{n-j}$

Where,

r_j = the accumulated sample autocorrelations,

m = the time lag.

If p value is less than level of significance 0.05, then we reject the null hypothesis and conclude that residuals are correlated else we fail to reject the null hypothesis and conclude that residuals are uncorrelated. Here, we will reject null hypothesis and conclude that residuals are uncorrelated.

[40]:
```
#ljung box test

print(sm.stats.acorr_ljungbox(residuals, lags=[1,2,3], return_df=True))
```

```
    lb_stat  lb_pvalue
1  0.000189   0.989043
2  0.136623   0.933969
3  4.957313   0.174950
```

Chapter 4

Forecasting the Price of Rupee using Long Short-term Memory model

Now, we are familiar with statistical modelling on time series, but machine learning is all the rage right now, so it is essential to be familiar with some machine learning models as well. We shall start with the most popular model in time series domain - Long Short-term Memory model.

LSTM is a class of recurrent neural network. So before we can jump to LSTM, it is essential to understand neural networks and recurrent neural networks.

- Neural Networks

An artificial neural network is a layered structure of connected neurons, inspired by biological neural networks. It is not one algorithm but combinations of various algorithms which allows us to do complex operations on data.

- Recurrent Neural Networks It is a class of neural networks tailored to deal with temporal data. The neurons of RNN have a cell state/memory, and input is processed according to this internal state, which is achieved with the help of loops with in the neural network. There are recurring module(s) of 'tanh' layers in RNNs that allow them to retain information. However, not for a long time, which is why we need LSTM models.
  For this we have reffered book: Krollner et al. [2010]

4.1 Intrododuction to LSTM

LSTM recurrent neural networks

It is special kind of recurrent neural network that is capable of learning long term dependencies in data. LSTM stands for Long short-term memory. LSTM cells are used in recurrent neural networks that learn to predict the future from sequences of variable lengths. Note that recurrent neural networks work with any kind of sequential data and, unlike ARIMA and Prophet, are not restricted to time series.

The main idea behind LSTM cells is to learn the important parts of the sequence seen so far and forget the less important ones. This is achieved by the so-called gates, i.e., functions that have different learning objectives such as:

A compact representation of the time series seen so far how to combine new input with the past representation of the series what to forget about the series what to output as a prediction for the next time step.

Observe that while the error on the training dataset decreases over subsequent epochs, this is not the case for the error on the validation set which reaches its minimum in the second epoch and then fluctuates. This shows that the LSTM model is too advanced for a rather small dataset and is prone to overfitting. Despite adding regularization terms such as dropout, we can't still avoid overfitting. LSTM-based recurrent neural networks are probably the most powerful approach to learning from sequential data and time series are only a special case. The potential of LSTM based models is fully revealed when learning from massive datasets where we can detect complex patterns. Unlike ARIMA or Prophet, they do not rely on specific assumptions about the data such as time series stationarity or the existence of a Date field. A disadvantage is that LSTM based RNNs are difficult to interpret and it is challenging to gain intuition into their behaviour. Also, careful hyperparameter tuning is required in order to achieve good results.

4.2 LSTM Model building

Now we will fit the LSTM model for our data.

Importing Libraries

[41]:
```
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, LSTM
import warnings
warnings.filterwarnings('ignore')
```

Importing Data File

Information about Dataset

Dataset is downloaded from www.investing.com

- Price : Closing Price of Day.
- Open : It is the price at which the financial security opens in the market when trading begins.
- High : It is the highest price at which a stock traded during a period.
- Low : It is the minimum price of a stock in a period.
- Change : Change of Prcentage compare with Yesterdays value.

[42]:
```
data = pd.read_csv("C:/Users/Shubham/OneDrive/Desktop/data.
 ↪csv",index_col="Date")
data.index=pd.to_datetime(data.index)
data
```

[42]:

Date	Price	Open	High	Low	Change %
1980-01-02	8.000	8.000	8.000	8.000	0.00%
1980-01-03	7.950	7.950	7.950	7.950	-0.63%
1980-01-04	8.050	8.050	8.050	8.050	1.26%
...	...	...	...	...	...
2022-04-27	76.605	76.768	76.806	76.490	-0.07%
2022-04-28	76.660	76.540	76.732	76.419	0.07%
2022-04-29	76.520	76.618	76.694	76.271	-0.18%

[10928 rows x 5 columns]

[46]:
```
## Splitting Data into train and test
df = data.filter(['Price'])
```

```
dataset = df.values
train_data_len = math.ceil(len(dataset)*.80)
train_data_len
```

[46]: 8743

[47]:
```
## Feature Scaling
scaler = MinMaxScaler(feature_range=(0,1))
scaler_data = scaler.fit_transform(dataset)
scaler_data
```

[47]:
```
array([[0.00461794],
       [0.00389639],
       [0.00533949],
       ...,
       [0.99466051],
       [0.99545422],
       [0.99343387]])
```

[48]:
```
train_data = scaler_data[0:train_data_len , :]
x_train = []
y_train = []
for i in range(60 , len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

- LSTM model

[49]:
```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.
 ↪shape[1],1)))
```

```
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
history=model.fit(x_train, y_train, epochs=1, batch_size=1)
```

[50]:
```
test_data = scaler_data[train_data_len - 60:, :]
x_test = []
y_test = dataset[train_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

4.3 Forecasting the Rupee's price

Now we will forecast the rupee's value using fitted LSTM model.

[51]:
```
pred = model.predict(x_test)
pred = scaler.inverse_transform(pred)
```

[52]:
```
x=valid.Price
x.columns='Price'
```

[53]:
```
x1=x.reset_index()
```

[54]:
```
x1['Predicted Price']=pred
x1
```

Forecasted prices are given below:

[54]:

	Date	Price	Predicted Price
0	2013-12-16	61.740	61.079670
1	2013-12-17	62.020	61.157833

```
2     2013-12-18  62.100        61.257458
3     2013-12-19  62.150        61.356441
4     2013-12-20  62.000        61.445358
...          ...     ...              ...
2180  2022-04-25  76.720        74.726585
2181  2022-04-26  76.660        74.811066
2182  2022-04-27  76.605        74.882286
2183  2022-04-28  76.660        74.927658
2184  2022-04-29  76.520        74.966164

[2185 rows x 3 columns]
```

[55]:
```
x1.to_csv('LSTM predicted output.csv')
```

plot of forecasted values and actual values

[56]:
```
train = data[:train_data_len]
valid = data[train_data_len:]
valid['Predictions'] = pred
plt.figure(figsize=(16,8))
plt.title('  Plot of forecasted value')
plt.xlabel('Date')
plt.ylabel('Price INR (Rs)')
plt.plot(train['Price'])
plt.plot(valid[['Price','Predictions']])
plt.legend(['Train','Valid','Predictions'], loc='lower right')
plt.show()
```

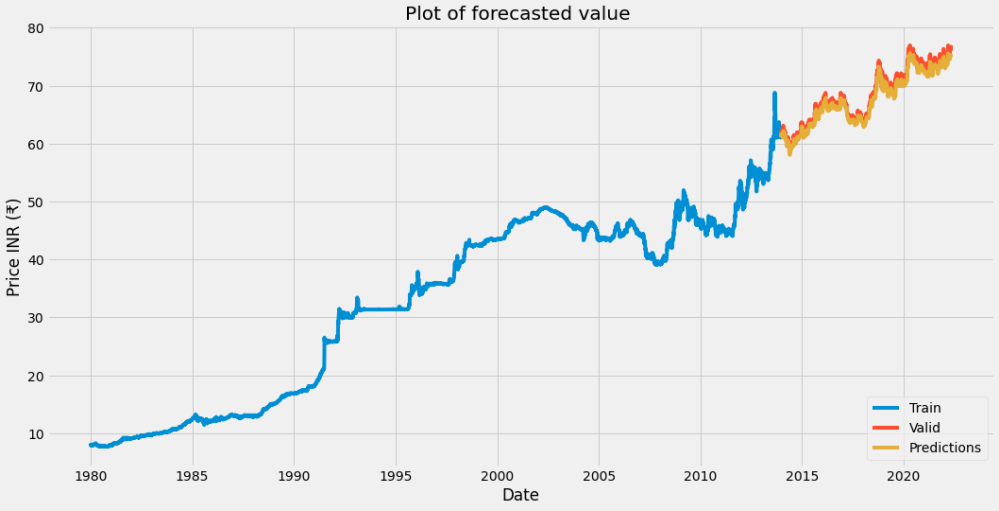

Figure 4.3.0.1: Plot of forecasted value

fig 6.3.1 is plot of valid as well as predicted rupee's price.

[57]:
```
plt.figure(figsize=(16,8))
plt.plot(valid[['Price','Predictions']])
plt.legend(['Valid','Predictions'], loc='lower right')
plt.title(' Plot of forecasted value')
plt.xlabel('Date')
plt.ylabel('Price INR (Rs)')
plt.show()
```

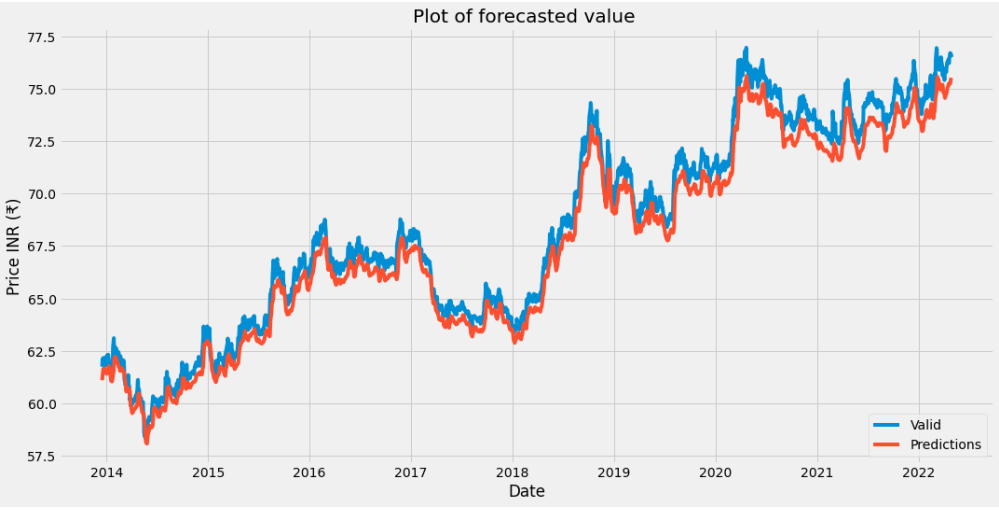

Figure 4.3.0.2: Plot of forecasted value with valid observations

4.4 Diagnostic Checking

We will check the adequecy of our model by using four measures:

[58]:
```
y_pred= x1['Predicted Price']
y_true= x1['Price']


mape = np.mean(np.abs(y_pred - y_true) /np.abs(y_true)) # Mean absolute
percentage error -
mae = np.mean(np.abs(y_pred - y_true)) # Mean absolute error
mpe = np.mean((y_pred - y_true)/y_true) # Mean percentage error
rmse = np.mean((y_pred - y_true)**2)**.5 # RMSE
corr = np.corrcoef (y_pred, y_true) [0,1]


# Correlation Coefficient
mins = np.amin(np.hstack([y_pred[:,None], y_true[:,None]]), axis=1)
maxs = np.amax(np.hstack([y_pred[:,None], y_true[:,None]]), axis=1)
minmax = 1 - np.mean(mins/maxs)    # minmax


import pprint
pprint.pprint({'mape':mape,'mae':mae,'mpe':mpe,'rmse':rmse,'corr':
 ↪corr,'minmax' :minmax})
```

```
{'corr': 0.9974360389874146,
 'mae': 0.9723149551217016,
 'mape': 0.013943989524638089,
 'minmax': 0.013943794336773063,
 'mpe': -0.013877795474065266,
 'rmse': 1.0807111176405806}
```

If model have MAPE less than 5% then model is good fitted.
For our model MAPE is 1.39%.
Here, all diagnostic measures of LSTM model are less than all diagnostic measures ARIMA model. Hence, LSTM model is better to forecast the rupees price.

Chapter 5

Factors affecting strongly on change in value of Indian Rupee

The aim of this analysis is to find the most effective regression model that can be used to predict the US dollars to Indian rupee exchange rate. The eight variables were carefully chosen, reading papers and intuition. we will fit the model for each factor, so we can predict the price if we know value of one variable.for this analysis we will use simple linear regression.

for this we reffered the book: Montgomery et al. [2021]

5.1 Factors affecting on value of Indian rupee

1.Gold Price:

Gold and currencies have a correlation between them. The value of a country's currency has a correlation with the gold reserves of that country. Also, the gold rate of that country affects the strength of its currency. However, this concept is not recent, where gold has impacted the currencies across the world for decades. Let us understand how gold affects current value across the world. India is one of the major importers of gold from other countries. In simple words, when the country imports more gold, the value of the currency dereases. On the contrary, when the export of gold is higher, the value of that country's currency is higher. The gold prices fluatuates daily and depend on the international market. Therefore, if India is exporting more than importing, the value of the Indian rupee will increase when the gold prices increase. However, India is a major importer of gold. So when the gold rates increase across the globe , the value of the Indian rupee declines.

2. Crude Oil Price:

The rise in crude oil prices has a clear impact on the Indian rupee. On 24 May 2018, the rupee closed at 68.34 against the US dollar. This is a near 18-month low for the rupee, and only 0.6% away from its all-time low of 68.825, according to a Livemint report . In addition, if crude oil prices remain at these high levels, the rupee is further expected to depreciate by the year end. Rupee depreciation has a reverberating effect on the Indian economy and even the stock market. To arrest the rupee's fall, the RBI often takes a few steps

3. Foreign direct investment :

Foreign direct investment refers to direct investment equity flows in the reporting economy. It is the sum of equity capital, reinvestment of earnings, and other capital. Direct investment is a category of cross-border investment associated with a resident in one economy having control or a significant degree of influence on the management of an enterprise that is resident in another economy. Ownership of 10 percent or more of the ordinary shares of voting stock is the criterion for determining the existence of a direct investment relationship. Data are in current U.S. dollars.

India foreign direct investment for 2019 was $50.61B, a 20.17% increase from 2018.

India foreign direct investment for 2018 was $42.12B, a 5.38% increase from 2017.

India foreign direct investment for 2017 was $39.97B, a 10.1% decline from 2016.

India foreign direct investment for 2016 was $44.46B, a 1.02% increase from 2015.

4. Petroleum Prices:

The rise in Petroleum prices has a clear impact on the Indian rupee. In addition, if Petroleum prices remain at these high levels, the rupee is further expected to depreciate by the year end. Rupee depreciation has a reverberating effect on the Indian economy and even the stock market. To arrest the rupee's fall, the RBI often takes a few steps

5. Inflation Rate:

Inflation is more likely to have a significant negative effect, rather than a significant positive effect, on a currency's value and foreign exchange rate. A very low rate of inflation does not guarantee a favorable exchange rate for a country, but an extremely high inflation rate is very likely to impact the country's exchange rates with other nations negatively. Low-interest rates spur consumer spending and economic growth, and generally, they have positive influences on currency value. If consumer spending increases to the point where demand exceeds supply, inflation may ensue, which is not necessarily a bad outcome. But low-interest rates do not commonly attract foreign investment.

Inflation as measured by the consumer price index reflects the annual percentage change in the cost to the average consumer of acquiring a basket of goods and services that may be fixed or changed at specified intervals, such as yearly. The Laspeyres formula is generally used.

India inflation rate for 2020 was 6.62%, a 2.9% increase from 2019.

India inflation rate for 2019 was 3.72%, a 0.22% decline from 2018.

India inflation rate for 2018 was 3.95%, a 0.62% increase from 2017.

India inflation rate for 2017 was 3.33%, a 1.62% decline from 2016.

6. GDP:

What is GDP? :-

The first basic concept of GDP was invented at the end of the 18th century. The modern concept was developed by the American economist Simon Kuznets in 1934 and adopted as the main measure of a country's economy at the Bretton Woods conference in 1944. GDP stands for "Gross Domestic Product".The gross domestic product is the value of all final goods and services produced in the economy in the given period of time. GDP is a very strong measure to gauge the economic health of a country and it reflects the sum total of the production of a country and as such comprises all purchases of goods and services produced by a country. It is used as an indicator by almost all the governments and economic decision-makers for planning and policy formulation.The imported goods are not considerd in GDP only those goods and services are consider in India.GDP is calculated according to the financial year i.e 1 April to 31 March.Gross domestic product or GDP, tells us the country's current aggregate production of goods and services. It is often considered the best measure of how well the economy is performing. GDP summarizes the aggregate of all economic activities in a given period of time. In any economy, however, goods and services produced are not homogenous. It is not possible to add, for example, 10 barrels of petroleum with 10 million matric tons of wheat. So, as a trick, quantities and volumes of all respective goods and services are multiplied by their prices and then summed up. This gives the money value of GDP.Prices however include indirect business taxes (IBT) i.e. sales taxes and excise duties. So this GDP is not a true measure of the productive activities in the economy. In order to get a true measure of GDP we deduct IBT from GDP. This is called GDP at factor cost. For all practical purposes the government uses data on GDP at factor cost.

Gross :-

"Gross" (in "Gross Domestic Product") indicates that products are counted regardless of their subsequent use. A product can be used for consumption, for investment, or to replace an asset. In all cases, the product's final "sales receipt" will be added to the total GDP figure.

Domestic :-

"Domestic" (in "Gross Domestic Product") indicates that the inclusion criterion is geographical: goods and services counted are those produced within the country's border, regardless of the nationality of the producer. For example, the production of a German-owned factory in the United States will be counted as part of United States' GDP.

Product :-

"Product" (in "Gross Domestic Product") stands for production, or economic output, of final goods and services sold on the market.

7. Imports:

Imports of goods and services represent the value of all goods and other market services received from the rest of the world. They include the value of merchandise, freight, insurance, transport, travel, royalties, license fees, and other services, such as communication, construction, financial, information, business, personal, and government services. They exclude compensation of employees and investment income (formerly called factor services) and transfer payments. Data are in current U.S. dollars.

India imports for 2020 was $482.45B, a 19.8% decline from 2019.

India imports for 2019 was $601.58B, a 5.86% decline from 2018.

India imports for 2018 was $639.01B, a 9.79% increase from 2017.

India imports for 2017 was $582.02B, a 21.21% increase from 2016.

8. Exports:

Exports of goods and services represent the value of all goods and other market services provided to the rest of the world. They include the value of merchandise, freight, insurance, transport, travel, royalties, license fees, and other services, such as communication, construction, financial, information, business, personal, and government services. They exclude compensation of employees and investment income (formerly called factor services) and transfer payments. Data are in current U.S. dollars.

India exports for 2020 was $474.15B, a 10.37% decline from 2019.

India exports for 2019 was $529.02B, a 1.79% decline from 2018.

India exports for 2018 was $538.64B, a 8.1% increase from 2017.

India exports for 2017 was $498.26B, a 13.33% increase from 2016.

5.2 EDA of factors

[59]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sympy as sym
import scipy.stats as ss
from warnings import filterwarnings
from sklearn.preprocessing import StandardScaler
filterwarnings("ignore")
#init_printing()   # the function init_printing() will enable Latex␣
 ↪pretty printing in the notebook for sympy.
```

Information about dataset:
dataset is downloaded from www.macrotrends.net
Dataset contains 8 variables.

- **year**: From 1971 to 2020 years are given.
- **Price**: Daily prices are given.
- **Gold Price RsPrice**: Yearly Gold prices are given.
- **Import (Billions of US dollar)**: Yearly import of India is given.
- **Export (Billions of US dollar)**:Yearly export of India is given.
- **Inflation Rate of INDIA**: Yearly Inflation Rate of India is given.
- **GDP (Billions of dollar)**:Yearly GDP of India is given.
- **Forign Investment Inflows (Billions of US dollar)**:Yearly Forign Investment Inflows of India is given.

[60]:
```
data=pd.read_csv("regression.csv",index_col="Year")
data.head()
```

[60]:

Year	Price Rs	Gold Price Rs	Import (Billions of US $)	\
1971	7.4900	193.0	2.70	

```
1972    7.5900          202.0                       2.65
1973    7.7400          278.0                       4.04
1974    8.1000          506.0                       5.99
1975    8.4483          540.0                       6.55
```

```
Year         Export (Billions of US $)  Inflation Rate (%)  INDIA  \
1971                          2.47                       3.08
1972                          2.88                       6.44
1973                          3.60                      16.94
1974                          4.81                      28.60
1975                          5.56                       5.75
```

```
Year        GDP (Billions of $)     Forign Investment Inflows (Billions␣
↪of US $)
1971                67.35                                          0.05
1972                71.46                                          0.02
1973                85.52                                          0.04
1974                99.53                                          0.06
1975                98.47                                          0.01
```

[61]:
```
print(data.shape)
```

```
(50, 7)
```

There are 50 observations with 7 factors (columns).

[62]:
```
print(data.describe().T)
```

```
                                count          mean  \
Price Rs                         50.0     32.814002
Gold Price Rs                    50.0   9630.245800
```

Import (Billions of US $)	50.0	164.549200
Export (Billions of US $)	50.0	144.902000
Inflation Rate (%) INDIA	50.0	7.770200
GDP (Billions of $)	50.0	792.119800
Forign Investment Inflows (Billions of US $)	50.0	11.965400

	std	min	\
Price Rs	21.230247	7.49	
Gold Price Rs	11956.759802	193.00	
Import (Billions of US $)	212.677363	2.65	
Export (Billions of US $)	185.498386	2.47	
Inflation Rate (%) INDIA	4.939200	-7.63	
GDP (Billions of $)	841.229440	67.35	
Forign Investment Inflows (Billions of US $)	17.382944	0.01	

	25%	50%	\
Price Rs	10.467275	33.90985	
Gold Price Rs	1842.500000	4317.00000	
Import (Billions of US $)	17.162500	42.31500	
Export (Billions of US $)	12.350000	39.93500	
Inflation Rate (%) INDIA	4.730000	7.11500	
GDP (Billions of $)	213.685000	376.59000	
Forign Investment Inflows (Billions of US $)	0.072500	2.15500	

	75%	max
Price Rs	46.782475	74.0861
Gold Price Rs	12075.000000	48651.0000
Import (Billions of US $)	336.085000	639.0100
Export (Billions of US $)	268.582500	538.6400
Inflation Rate (%) INDIA	10.010000	28.6000

```
GDP (Billions of $)                              1212.280000   2870.5000
Forign Investment Inflows (Billions of US $)       24.922500     55.5600
```

This is summary statistics of all factors.

[63]:
```
data.isnull().sum()
```

[63]:
```
Price Rs                                    0
Gold Price Rs                               0
Import (Billions of US $)                   0
Export (Billions of US $)                   0
Inflation Rate (%)  INDIA                   0
GDP (Billions of $)                         0
Forign Investment Inflows (Billions of US $)    0
dtype: int64
```

There are no missing values.

[64]:
```
sns.pairplot(data,diag_kind='kde')
```

[64]:

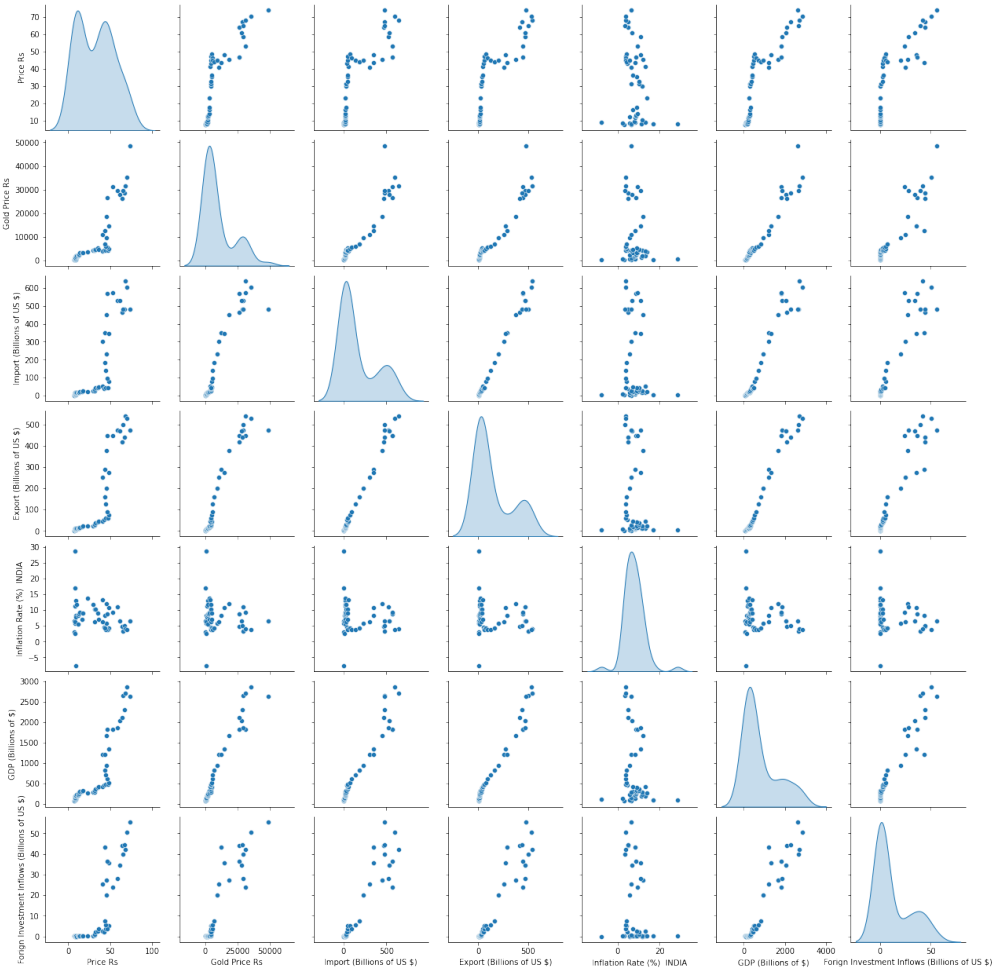

Figure 5.2.0.1: Pair plot of all variables

We can observe all pair plot of all variables with each other.

[65]:
```
sns.heatmap(data.corr(),annot=True)
```

[65]:

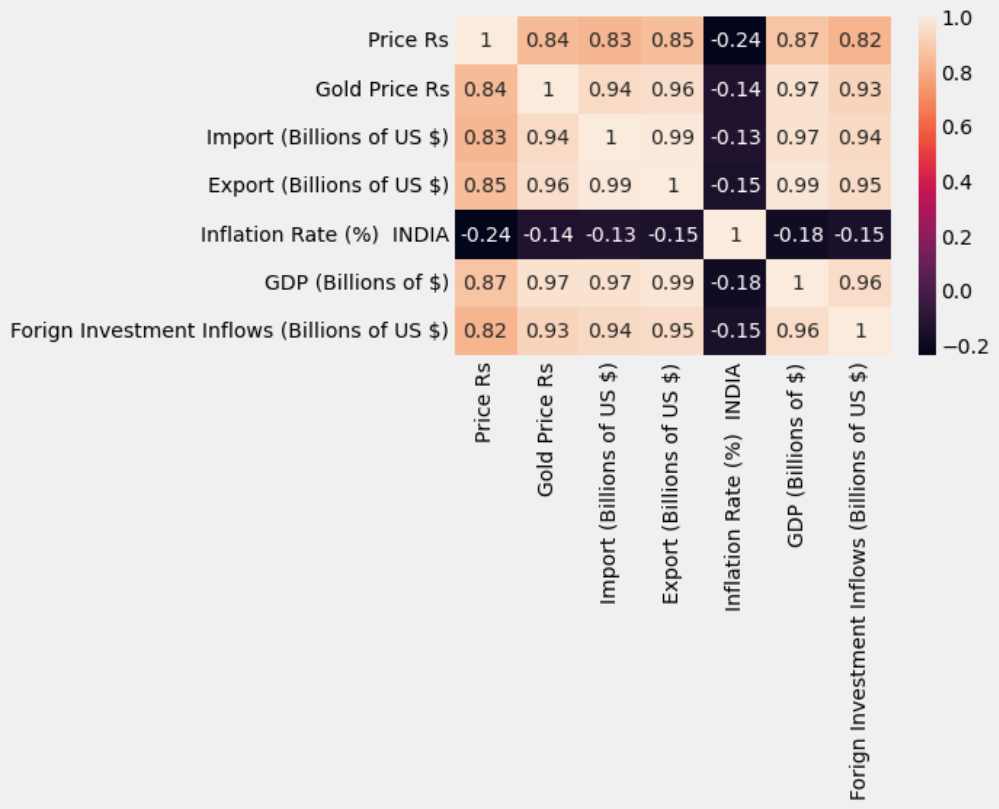

Figure 5.2.0.2: Correlation heatmap

All factors are corelated with Price except Inflation Rate.
Also there is multicolinearity present in this data.

5.3 Simple Linear Regression Model Fitting

SIMPLE LINEAR REGRESSION MODEL :

simple linear regression model , that is, a model with a single regressor x that has a relationship with a response y that is a straight line. This simple linear regression model is

$y = \beta 0 + \beta 1x + \varepsilon$

where the intercept $\beta 0$ and the slope $\beta 1$ are unknown constants and ε is a random error component. The errors are assumed to have mean zero and unknown variance σ^2

The parameters $\beta 0$ and $\beta 1$ are usually called regression coefficients . These coefficients have a simple and often useful interpretation. The slope $\beta 1$ is the change in the mean of the distribution of y produced by a unit change in x. If the range of data on x includes x = 0, then the intercept $\beta 0$ is the mean of the distribution of the response y when x = 0. If the range of x does not include zero, then $\beta 0$ has no practical interpretation.

Fitted simple linear regression model is

$\hat{y} = \beta 0 + \beta 1 * x$

$\beta 0$ and $\beta 1$ are the least - squares estimators of the intercept and slope, respectively

Relation between value of Indian rupee and Gold Price

[66]:
```
x=data.iloc[:,1:2]
y=data['Price Rs'].values
plt.title('Scatter Plot ')
plt.xlabel('Gold Price (Rs) ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[66]:

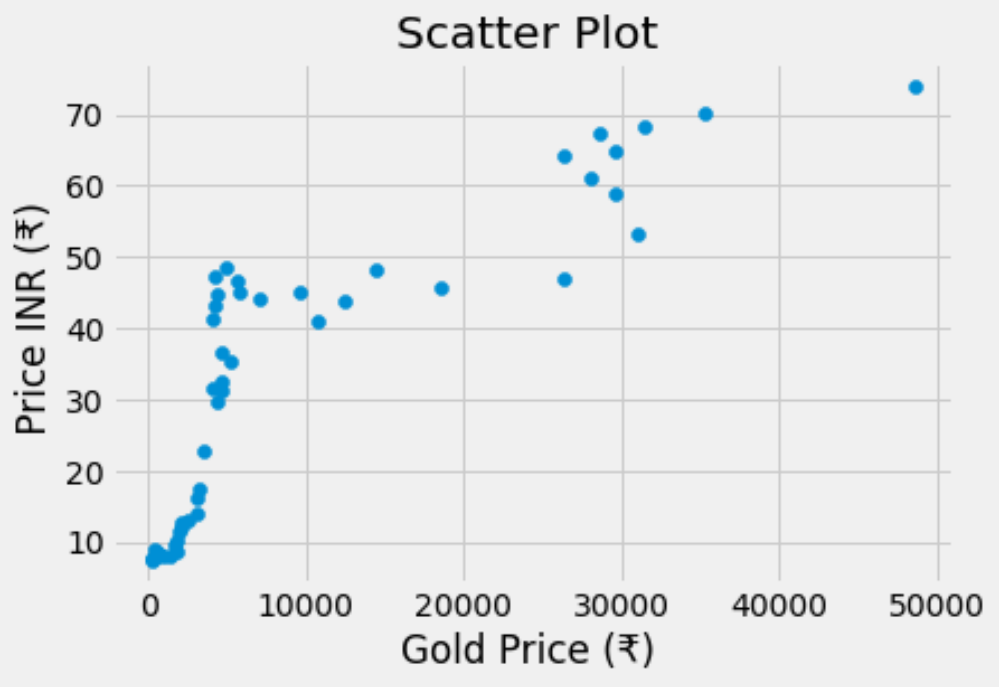

Figure 5.3.0.1: Scatter plot of Price INR vs Gold Price

There is less linear relationship between both of them hence we will use root transformation.

[67]:
```
x=data.iloc[:,1:2]
y=data['Price Rs'].values
import numpy as geek
x=geek.sqrt(x)
x=geek.sqrt(x)
plt.title('Scatter Plot ')
plt.xlabel('Gold Price (Rs)^(1/4) ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[67]:

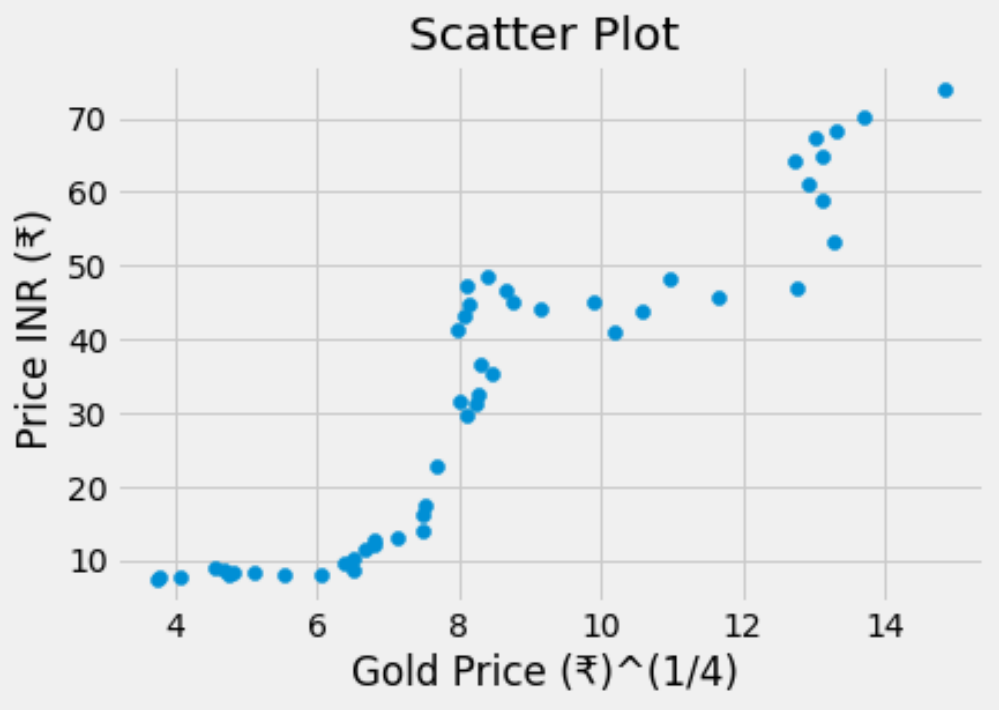

Figure 5.3.0.2: Scatter plot of Price INR vs Gold Price after transformation

We will fit least square regression model.

[68]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪1,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
ypred=model.predict(x_test)
print(model.score(x_test,y_test))
print(model.coef_)
print(model.intercept_)  #$\beta_0$
```

```
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train)   #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.729917609439268
[19.51592479]
33.73827333333334
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.867
Model:	OLS	Adj. R-squared:	0.864
Method:	Least Squares	F-statistic:	280.7
Covariance Type:	nonrobust	Prob (F-statistic):	1.84e-20
Df Model:	1	Log-Likelihood:	-155.35
No. Observations:	45	AIC:	314.7
Df Residuals:	1	BIC:	318.3

	coef	std err	t	P>|t|	[0.025	0.975]
const	33.7383	1.165	28.963	0.000	31.389	36.087
x1	19.5159	1.165	16.754	0.000	17.167	21.865

Omnibus:	1.000	Durbin-Watson:	1.731
Prob(Omnibus):	0.607	Jarque-Bera (JB):	1.044
Skew:	0.312	Prob(JB):	0.593
Kurtosis:	2.591	Cond. No.	1.00

specified.

Model accuracy is 72.99 %.

R-squared value is 0.867 i.e. 86.7% variation in value of Indian Rupee is explained by the fitted model.

Fitted model is $\hat{y} = 30.7383 + 19.5159 * x^{1/4}$

Relation between value of Indian rupee and Imports of India

[69]:
```
x=data.iloc[:,2:3]
y=data['Price Rs'].values
plt.title('Scatter Plot ')
plt.xlabel('Import (Billions of US $)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[69]:

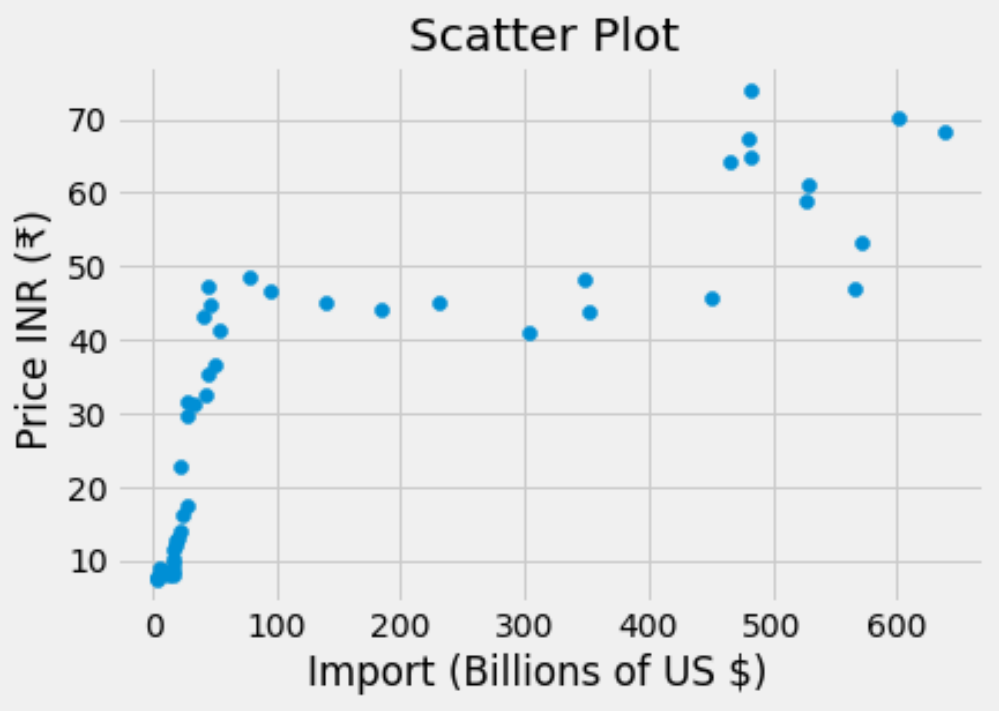

Figure 5.3.0.3: Scatter plot of Price INR vs Import

There is less linear relationship between both of them hence we will use root transforma-

tion.

[70]:
```
x=data.iloc[:,2:3]
y=data['Price Rs'].values
import numpy as geek
x=geek.sqrt(x)
x=geek.sqrt(x)
plt.title('Scatter Plot ')
plt.xlabel('Import (Billions of US $)^(1/4)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[70]:

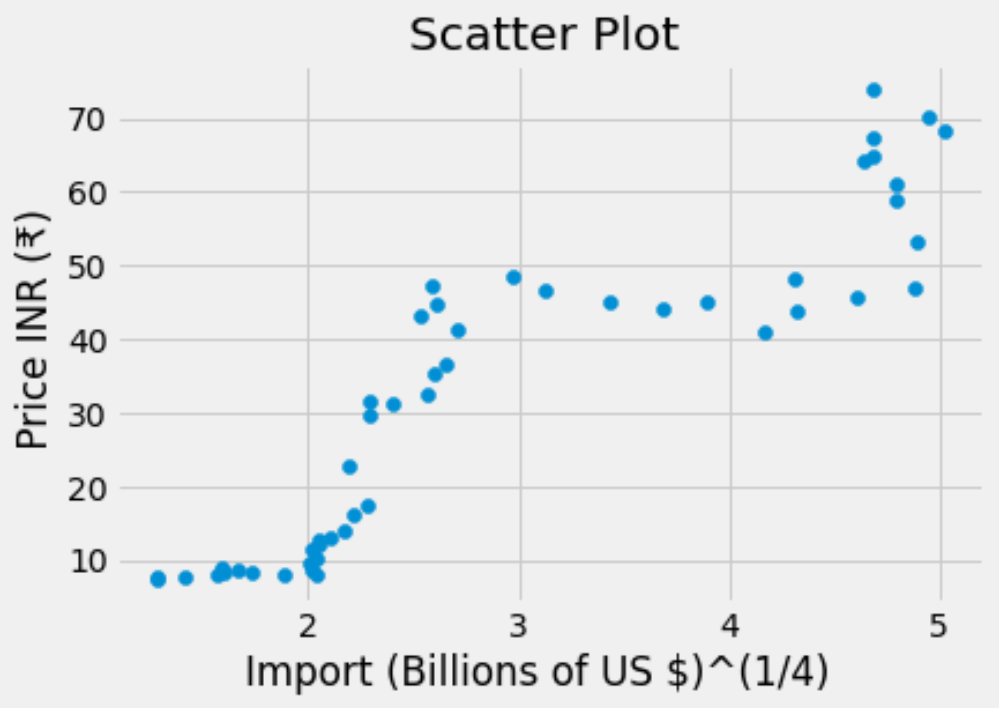

Figure 5.3.0.4: Scatter plot of Price INR vs Import after transformation

We will fit least square regression model.

[71]:
```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪1,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
ypred=model.predict(x_test)
print(model.score(x_test,y_test))
print(model.coef_)
print(model.intercept_)  #$\beta_0$
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train)   #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.7107283661870665
[19.17791139]
33.73827333333334
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y        R-squared:                  0.837
Model:                            OLS        Adj. R-squared:             0.834
Method:                 Least Squares        F-statistic:                221.4
Df Model:                           1        Prob(F-statistic):       1.44e-18
Covariance Type:            nonrobust        Log-Likelihood:           -159.90
```

```
No. Observations:              45        AIC:                     323.8
Df Residuals:                  43        BIC:                     327.4
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         33.7383      1.289     26.177      0.000      31.139      36.337
x1            19.1779      1.289     14.880      0.000      16.579      21.777
==============================================================================
Omnibus:                        1.145        Durbin-Watson:           2.088
Prob(Omnibus):                  0.564        Jarque-Bera (JB):        1.167
Skew:                           0.299        Prob(JB):                0.558
Kurtosis:                       2.485        Cond. No.                1.00
==============================================================================
```

Model accuracy is 71.07 %.

R-squared value is 0.837 i.e. 83.7% variation in value of Indian Rupee is explained by the fitted model.

Fitted model is $\hat{y}$ = 33.7383 + 19.1779 * $x^{1/4}$

Relation between value of Indian rupee and Exports of India

[72]:
```
x=data.iloc[:,3:4]
y=data['Price Rs'].values
plt.title('Scatter Plot ')
plt.xlabel('Export (Billions of US $)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[72]:

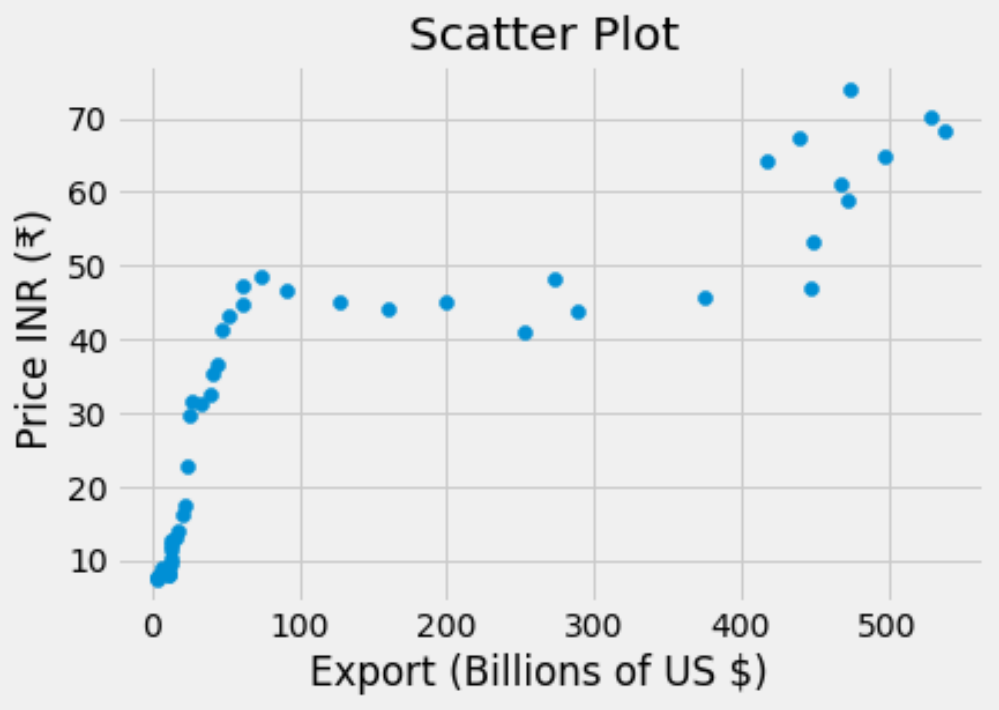

Figure 5.3.0.5: Scatter plot of Price INR vs Export

There is less linear relationship between both of them hence we will use root transformation.

[73]:
```
x=data.iloc[:,3:4]
y=data['Price Rs'].values
import numpy as geek
x=geek.sqrt(x)
x=geek.sqrt(x)
x=geek.sqrt(x)
plt.title('Scatter Plot ')
plt.xlabel('Export (Billions of US $)^(1/8)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[73]:

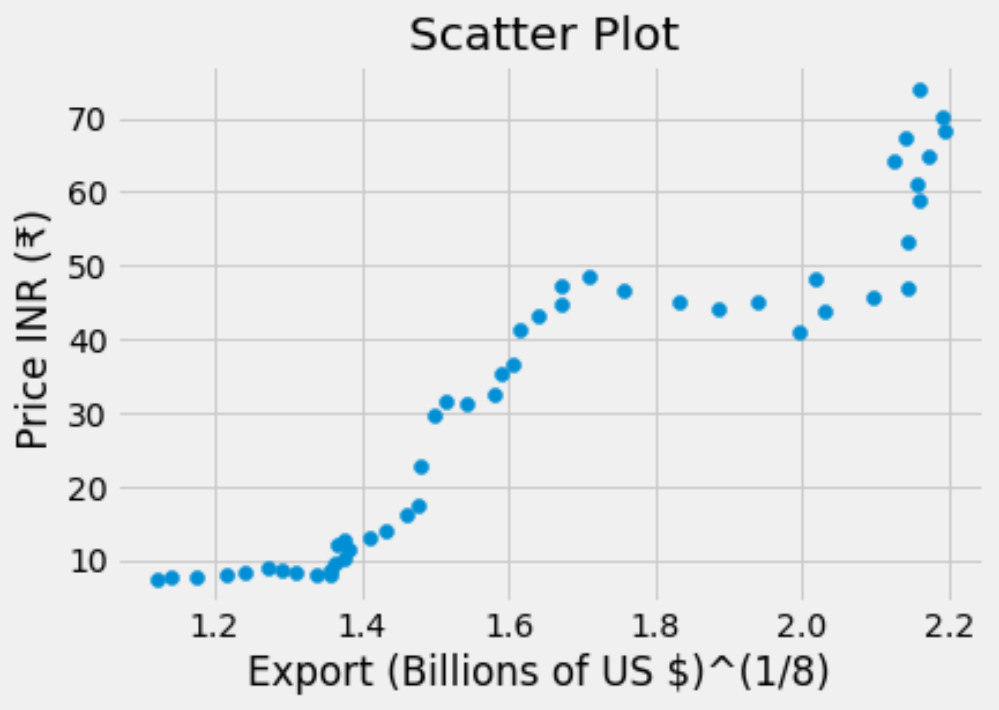

Figure 5.3.0.6: Scatter plot of Price INR vs Export after transformation

We will fit least square regression model.

[74]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪1,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
ypred=model.predict(x_test)
print(model.score(x_test,y_test))
print(model.coef_)
print(model.intercept_)  #$\beta_0$
```

```
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train)   #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.8527629313427398
[19.7525083]
33.73827333333332
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.888
Model:	OLS	Adj. R-squared:	0.886
Method:	Least Squares	F-statistic:	342.0
Df Model:	1	Prob(F-statistic):	4.36e-22
Covariance Type:	nonrobust	Log-Likelihood:	-151.44
No. Observations:	45	AIC:	306.9
Df Residuals:	43	BIC:	310.5

	coef	std err	t	P>|t|	[0.025	0.975]
const	33.738	1.068	31.587	0.000	31.584	35.892
x1	19.7525	1.068	18.493	0.000	17.598	21.907

Omnibus:	1.600	Durbin-Watson:	1.973
Prob(Omnibus):	0.449	Jarque-Bera (JB):	1.104
Skew:	-0.021	Prob(JB):	0.576
Kurtosis:	2.234	Cond. No.	1.000

Model accuracy is 85.28 %.

R-squared value is 0.888 i.e. 88.8% variation in value of Indian Rupee is explained by the fitted model.

Fitted model is $\hat{y} = 33.7383 + 19.7525 * x^{1/8}$

Relation between value of Indian rupee and Inflation rate of India

[75]:
```
x=data.iloc[:,4:5]
y=data['Price Rs'].values
plt.title('Scatter Plot ')
plt.xlabel('Import (Inflation rate %)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[75]:

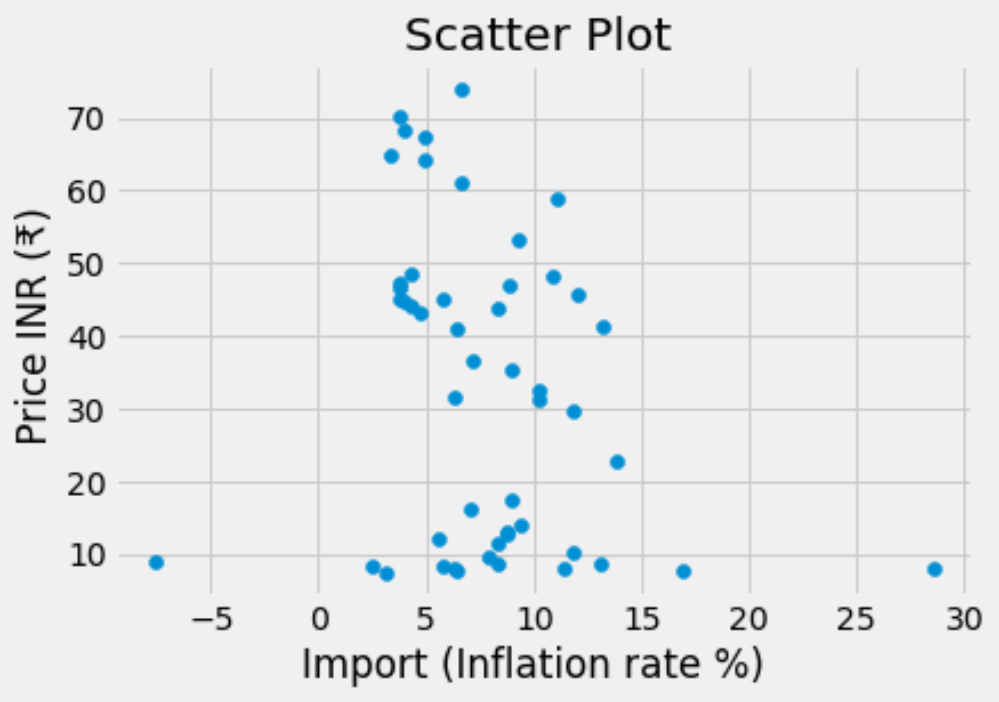

Figure 5.3.0.7: Scatter plot of Price INR vs Inflation Rate

From plot we can say that there is no linear relationship between value of indian rupee and inflation rate.

Relation between value of Indian rupee and GDP of India

[76]:
```
x=data.iloc[:,5:6]   #axis=3 represents columns
y=data['Price Rs'].values
plt.title('Scatter Plot ')
plt.xlabel(' GDP (Billions of $)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[76]:

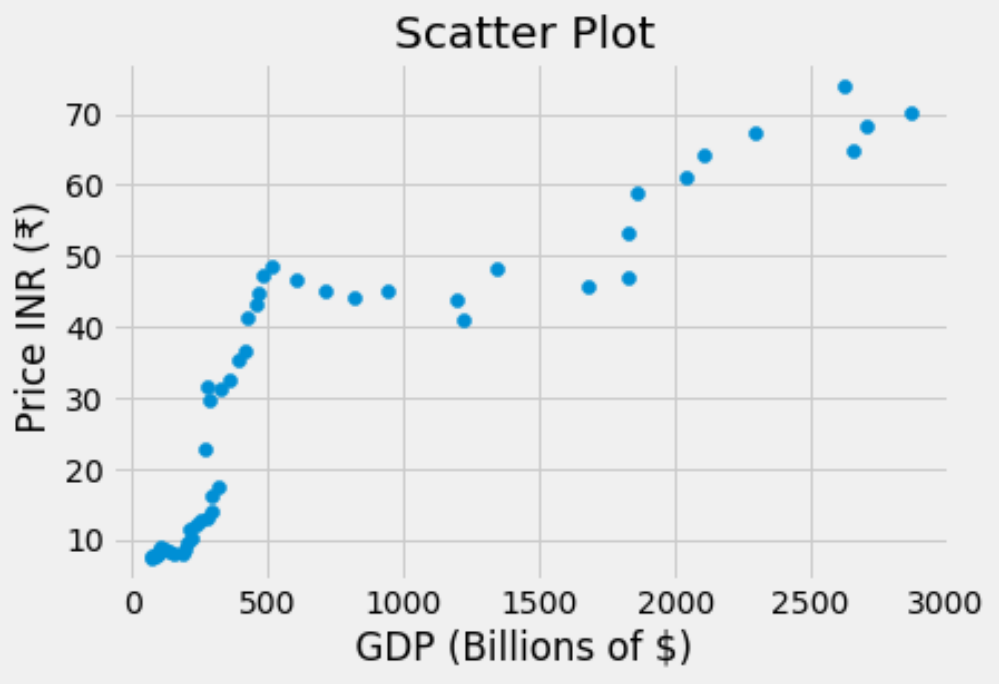

Figure 5.3.0.8: Scatter plot of Price INR vs GDP

There is less linear relationship between both of them hence we will use root transformation.

[77]:
```
x=data.iloc[:,5:6]   #axis=3 represents columns
y=data['Price Rs'].values
import numpy as geek
x=geek.sqrt(x)
plt.title('Scatter Plot ')
```

```
plt.xlabel(' GDP (Billions of $)^(1/2)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[77]:

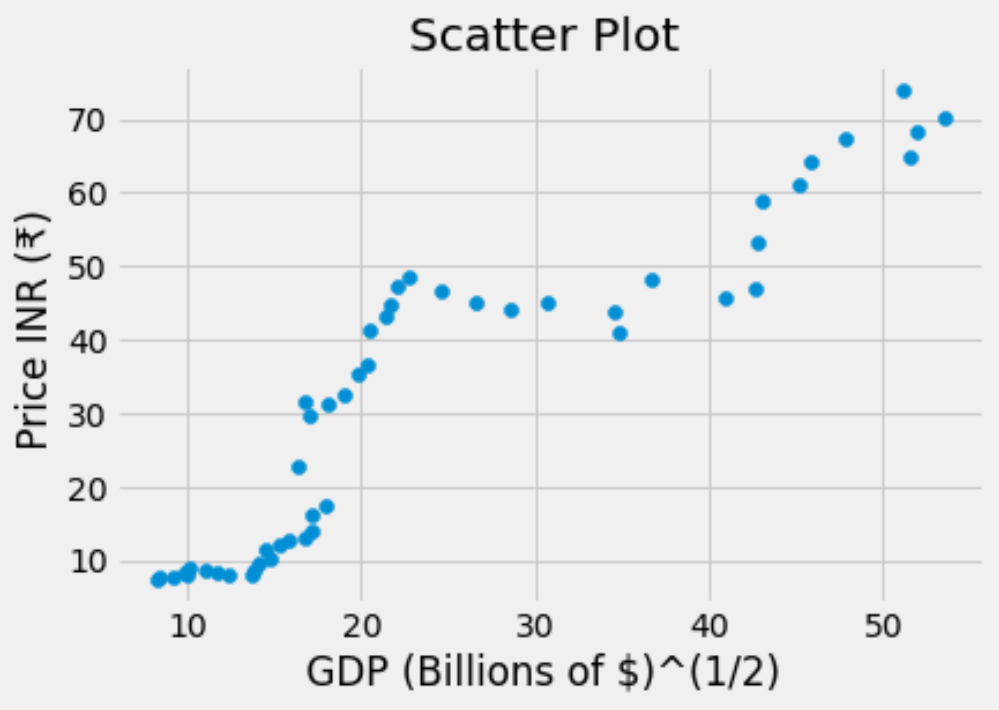

Figure 5.3.0.9: Scatter plot of Price INR vs GDP after transformation

We will fit least square regression model.

[78]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪1,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
```

```
model.fit(x_train,y_train)
ypred=model.predict(x_test)
print(model.score(x_test,y_test))
print(model.coef_)
print(model.intercept_)  #$\beta_0$
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train)   #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.7848800274540959
[19.30684822]
33.738273333333325
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.849
Model:	OLS	Adj. R-squared:	0.845
Method:	Least Squares	F-statistic:	241.2
Df Model:	1	Prob(F-statistic):	3.05e-19
Covariance Type:	nonrobust	Log-Likelihood:	-158.28
No. Observations:	45	AIC:	320.6
Df Residuals:	43	BIC:	324.2

	coef	std err	t	P>|t|	[0.025	0.975]
const	33.7383	1.243	27.137	0.000	31.231	36.246
x1	19.3068	1.243	15.529	0.000	16.800	21.814

Omnibus:	4.552	Durbin-Watson:	1.855
Prob(Omnibus):	0.103	Jarque-Bera (JB):	4.142
Skew:	0.671	Prob(JB):	0.126
Kurtosis:	2.361	Cond. No.	1.000

==========

Model accuracy is 78.49 %.

R-squared value is 0.849 i.e. 84.9% variation in value of Indian Rupee is explained by the fitted model.

Fitted model is $\hat{y} = 33.7383 + 19.3068 * x^{1/2}$

Relation between value of Indian rupee and Forign Investment Inflows of India

[79]:
```
x=data.iloc[:,6:7]   #axis=3 represents columns
y=data['Price Rs'].values
plt.title('Scatter Plot ')
plt.xlabel(' Forign Investment Inflows (Billions of US $)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[79]:

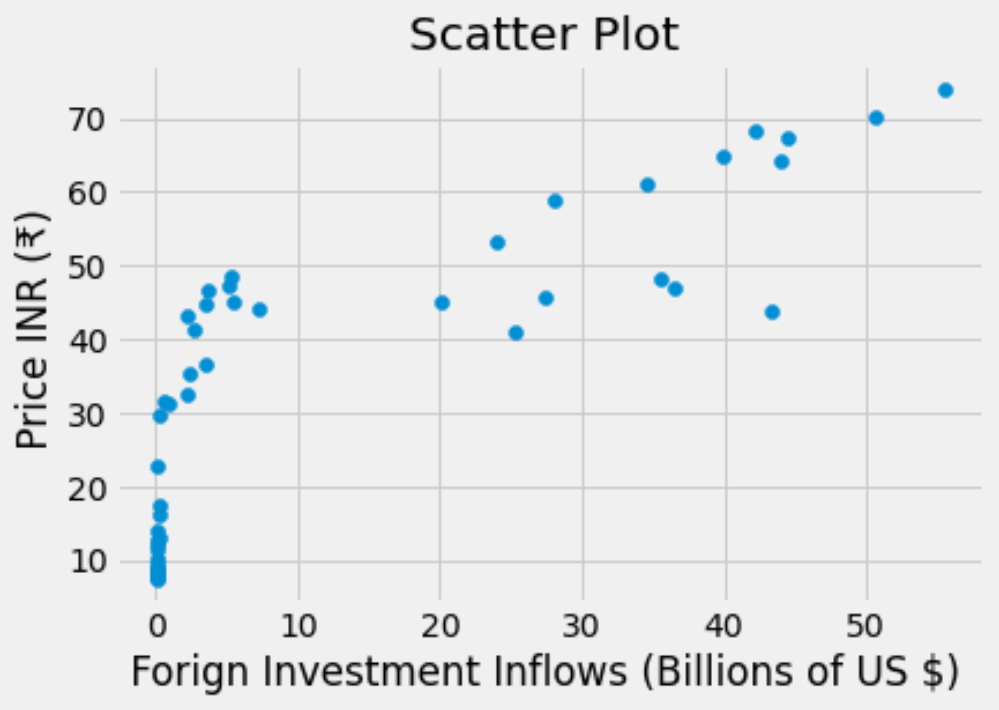

Figure 5.3.0.10: Scatter plot of Price INR vs Forign Investment Inflows

There is less linear relationship between both of them hence we will use root transformation.

[80]:
```
x=data.iloc[:,6:7]   #axis=3 represents columns
y=data['Price Rs'].values
import numpy as geek
x=geek.sqrt(x)
x=geek.sqrt(x)
plt.title('Scatter Plot ')
plt.xlabel(' Forign Investment Inflows (Billions of US $)^(1/4)  ')
plt.ylabel(' Price INR (Rs)')
plt.scatter(x,y)
```

[80]:

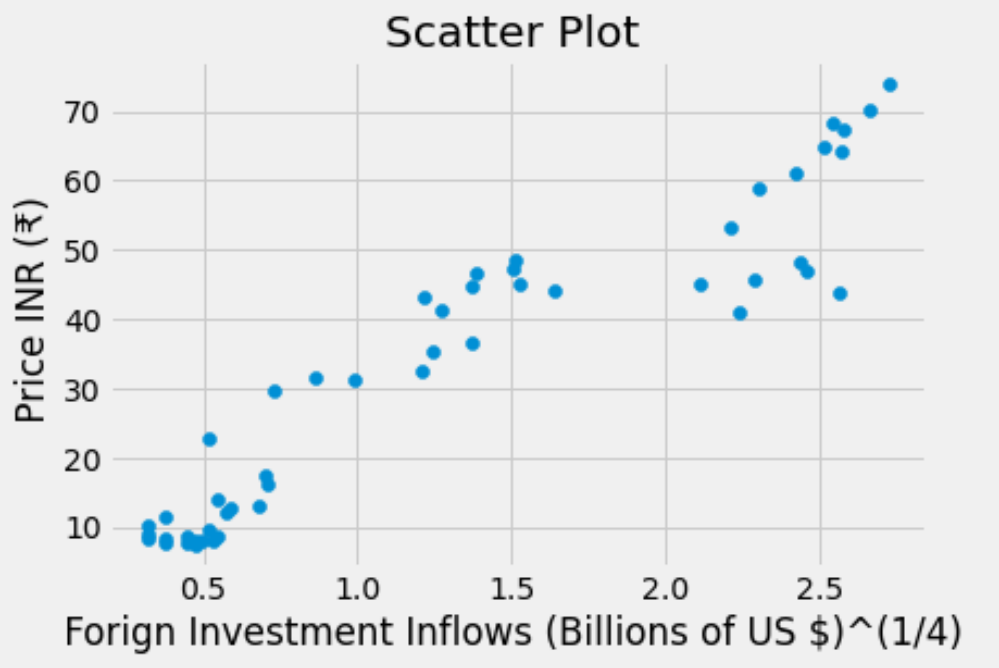

Figure 5.3.0.11: Scatter plot of Price INR vs Forign Investment Inflows after transformation

We will fit least square regression model.

[81]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪1,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
ypred=model.predict(x_test)
print(model.score(x_test,y_test))
print(model.coef_)
print(model.intercept_)  #$\beta_0$
```

```
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train)   #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.870564742801588
[19.68558553]
33.73827333333333
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.882
Model:	OLS	Adj. R-squared:	0.880
Method:	Least Squares	F-statistic:	322.3
Df Model:	1	Prob(F-statistic):	1.35e-21
Covariance Type:	nonrobust	Log-Likelihood:	-152.62
No. Observations:	45	AIC:	309.2
Df Residuals:	43	BIC:	312.9

	coef	std err	t	P>|t|	[0.025	0.975]
const	33.7383	1.096	30.770	0.000	31.527	35.950
x1	19.6856	1.096	17.954	0.000	17.474	21.897

Omnibus:	1.556	Durbin-Watson:	2.416
Prob(Omnibus):	0.459	Jarque-Bera (JB):	1.512
Skew:	-0.405	Prob(JB):	0.470
Kurtosis:	2.610	Cond. No.	1.000

Model accuracy is 87.06 %.

R-squared value is 0.882 i.e. 88.2% variation in value of Indian Rupee is explained by the fitted model.

Fitted model is $\hat{y}$ = 33.7383 + 19.6856 * $x^{1/4}$

5.4 Multiple Linear Regression Model Fitting

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable. multiple regression model is of the form, $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$

Perform Multiple linear regression with Price as the respose and all other variables as the predictor.

[82]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sympy as sym
import scipy.stats as ss
from warnings import filterwarnings
from sklearn.preprocessing import StandardScaler
filterwarnings("ignore")
#init_printing()   # the function init_printing() will enable Latex␣
 ↪pretty printing in the notebook for sympy.
```

[83]:
```
x=data.iloc[:,1:]   #axis=1 represents columns
y=data['Price Rs'].values
```

Model building:

[84]:
```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪2,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
ypred=model.predict(x_test)
print(model.score(x_test,y_test))
print(model.coef_)
print(model.intercept_)  #$\beta_0$
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train)   #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
 ↪#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.5248926564800774
[ -1.33639586 -24.99579084  31.05358217  -0.95201676  15.19925556
  -1.0597539 ]
33.502812500000005
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y         R-squared:                       0.818
Model:                            OLS         Adj. R-squared:                  0.785
Method:                 Least Squares         F-statistic:                     24.68
Covariance Type:            nonrobust         Prob (F-statistic):           7.00e-11
```

Df Model:	6	Log-Likelihood:	-145.58
No. Observations:	40	AIC:	305.2
Df Residuals:	33	BIC:	317.0

	coef	std err	t	P>|t|	[0.025	0.975]
const	33.5028	1.604	20.892	0.000	30.240	36.765
x1	-1.3364	7.067	-0.189	0.851	-15.715	13.042
x2	-24.9958	21.464	-1.165	0.253	-68.664	18.672
x3	31.0536	29.404	1.056	0.299	-28.770	90.877
x4	-0.9520	1.696	-0.561	0.578	-4.402	2.498
x5	15.1993	13.153	1.156	0.256	-11.560	41.959
x6	-1.0598	6.151	-0.172	0.864	-13.574	11.454

Omnibus:	4.330	Durbin-Watson:	1.717
Prob(Omnibus):	0.115	Jarque-Bera (JB):	2.572
Skew:	0.406	Prob(JB):	0.276
Kurtosis:	2.059	Cond. No.	50.600

Interpretation

The model obtained is:

Price =33.5028 + (-1.3364 × Gold price) + (-24.9958× Import)+(31.0536 × Export)+(-0.9520 × Inflation rate) + (15.1993 × GDP) + (-1.0598 * Foriegn envestment inflows)

The coefficient of R2 value the model is good fitted. The value of R square in the given model is 81.2%, showing that about 81.2%, of total variation in Manufacturing in India can be explained by independent variables

Since calculated F value is 24.68 which is greater than the p_value 7.00e-11, hence the model is accepted.

Test of significance of overall regression model. F-test
We can evaluate the significance of our model.
H0: $\beta 1 = \beta 2 = \beta 3 = \beta 4 = \beta 5 =, \beta 6 = 0$vs H1: β j != 0 for at least on j.
Here F-statistics value is 24.68 And P_value is 7.00e-11
There is linear relationship between Price y and at least one of the regressors from the x1, x2, x3, x4, x5, x6 explaining variation in y. Hence, we reject H0 at 5 % level of significance.
Which predictor appears to have statistically significant relationship to the response?
From t-test we can conclude that the predictors have significant relationship to the response or not.
H0: β i =0 vs H1: β i != 0 for at least one i. i=1,2,3,4,5,6. VS
H0: β i =0 vs H1: β i != 0 for at least one i. i=1,2,3,4,5,6.

Here, all variables are non significant. This is may due to Multicolinearity presence.So we will try to remove multicolinearity using Principle Component Analysis.

[85]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
#the independent variables set.
X=data[['Gold Price Rs','Import (Billions of US $)','Export (Billions of␣
 ↪US $)','Inflation Rate (%)  INDIA','GDP (Billions of $)','Forign␣
 ↪Investment Inflows (Billions of US $)']]
#VIF Dataframe
vif_data=pd.DataFrame()
vif_data['features']=X.columns
#Calculate VIF for each feature.
vif_data['VIF']=[variance_inflation_factor(X.values,i)
                 for i in range(len(X.columns))]
print(vif_data)
```

```
                                features         VIF
0                             Gold Price Rs   27.891165
1                 Import (Billions of US $)  256.083482
2                 Export (Billions of US $)  501.047863
3                 Inflation Rate (%)  INDIA    1.914040
4                       GDP (Billions of $)   85.472961
5  Forign Investment Inflows (Billions of US $)   16.660756
```

Here is the VIF is greater than 5 or 10 so the Multicollinearity is present in the above

regressors. We used here PCA technique.
Principal Component Analysis(PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. PCA is statastical procedure that allows you to summarize the information content in large data tables by means of a smaller set of summary indices that can be more easily visualized and analyzed. Using PCA can identify the correlation between the data points.

[86]:
```
from sklearn.preprocessing import StandardScaler
X_std=StandardScaler().fit_transform(X)
from sklearn.decomposition import PCA
pca=PCA().fit(X_std)
#plotting the cummulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel("Number of component")
plt.ylabel("Variance (%)")   #for each component
plt.title("Auto dataset Explained Variance")
plt.show()
```

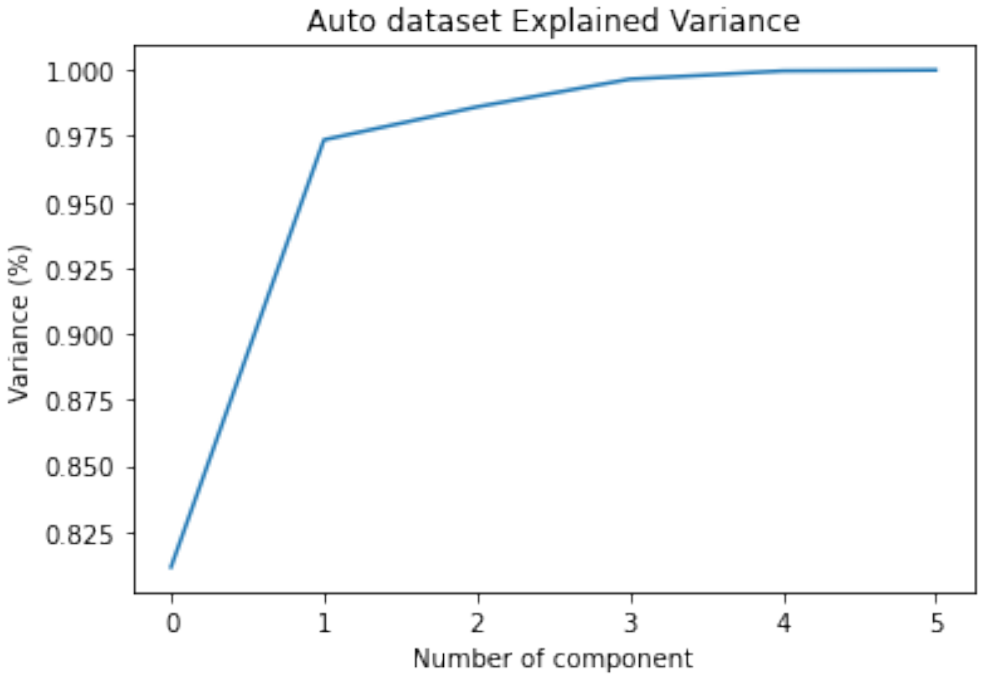

Figure 5.4.0.1: PCA plot

Used for standardized means the inputs by removing mean value from every value for that column and dividing it by the variance that exists for that column. Plot the overall Variance explained by each component.

When we apply PCA the only difference is rather than plotting individual contributions We plot overall cumulative sum of the contributions, So the sum total of contribution turns out to be 1 or 100% the here the first Two principal component value is greather than 0.9735 and other values are add up to the total of 1

[87]:
```
np.cumsum(pca.explained_variance_ratio_)     # cumulative contribution␣
↪captured by variance
```

[87]:
```
array([0.81150443, 0.97355066, 0.98603604, 0.99654919, 0.99967864,
       1.        ])
```

Finding principal componenets By taking 2 components we fit the model again.

[88]:
```
from sklearn.decomposition import PCA
x=StandardScaler().fit_transform(x)
pca=PCA(n_components=2)
x_pca = pca.fit_transform(x)
x_pca.shape
pca.explained_variance_ratio_
```

[88]:
```
array([0.81150443, 0.16204623])
```

[89]:
```
x_train_pca,x_test_pca,y_train,y_test=train_test_split(x_pca,y,test_size=0.
↪2,random_state=1)
from sklearn.linear_model import LinearRegression
model_2=LinearRegression()
model_2.fit(x_train_pca,y_train)
print(model_2.score(x_test_pca,y_test))
print(model_2.coef_)
```

```
print(model_2.intercept_)
import statsmodels.api as sm
x_train_lm=sm.add_constant(x_train_pca)    #dependent variable is also␣
 ↪concerned by the constant term 'a'.Hence we add constant term.
 ↪#x_train_lm
Result=sm.OLS(y_train,x_train_lm).fit()
print(Result.summary())
```

```
0.9391406745367646
[ 8.72354672 -0.78629772]
31.974766269899202
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.896
Model:	OLS	Adj. R-squared:	0.881
Method:	Least Squares	F-statistic:	61.01
Covariance Type:	nonrobust	Prob (F-statistic):	1.93e-12
Df Model:	2	Log-Likelihood:	-148.77
No. Observations:	40	AIC:	103.5
Df Residuals:	37	BIC:	108.6

	coef	std err	t	P>|t|	[0.025	0.975]
const	31.9748	1.666	19.194	0.000	28.599	35.350
x1	8.7235	0.799	10.922	0.000	7.105	10.342
x2	6.7863	0.985	6.883	0.000	5.120	11.547

Omnibus:	5.359	Durbin-Watson:	1.906
Prob(Omnibus):	0.069	Jarque-Bera (JB):	4.792
Skew:	0.770	Prob(JB):	0.0911

```
Kurtosis:                       2.289       Cond. No.                         2.800
==============================================================================
```

Interpretation

The model obtained is:
Price = 31.9748 + (8.7235 × Gold price) + (6.7863 × Import)
The coefficient of R2 value the model is good fitted. The value of R square in the given model is 0.896, showing that about 89.6%, of total variation in Manufacturing in India can be explained by independent variables.

Since calculated F value is 61.01 which is greater than the p_value 1.93e-12, hence the model is accepted.

here both varibles are significant.

Chapter 6

Conclusions

- To check whether the data is stationary or not, we saw the autocor- relation plot which showed slow decay which meant that our data is non stationary then we conducted ADF and KPSS test which also concluded our data being non stationary. So to convert the non stationary data to stationary data we carried out first differencing from which we got stationary data and using auto arima command we got the model ARIMA(3,1,2). Here the value of p=3, d=1 and q=2. Using this model we forecasted further values and the MAPE was 2.38%. By MAPE 97.63% model is accurate. Also residual are uncorrelated. so we can say that our fitted ARIMA(3,1,2) model is better
- We have used LSTM model and the MAPE was 1.4% i.e. 98.6% model is accurate. Also residual are uncorrelated. In comparison between ARIMA(3,1,2) and LSTM by using MAPE measure LSTM is better.
- All regression Models of all factors:
  - Fitted model is $\hat{Price}$ = 30.7383 + 19.5159 * (Gold Price Rs)$^{1/4}$
  - Fitted model is $\hat{Price}$ = 33.7383 + 19.1779 * Import (Billions of US dollar)$^{1/4}$
  - Fitted model is $\hat{Price}$ = 33.7383 + 19.7525 * Export (Billions of US dollar)$^{1/8}$
  - There is no linear relationship in Inflation Rate and Value of Indian rupee
  - Fitted model is $\hat{Price}$ = 33.7383 + 19.3068 * GDP (Billions of dollar) $^{1/2}$

  - Fitted model is $\hat{Price}$ = 33.7383 + 19.6856 * Forign Investment Inflows (Billions of US dpllar)$^{1/4}$

- We have fitted the multiple regression model for price. Mostly 5 factor affects on Price and that fitted model is:

  Price =33.5028 + (-1.3364 * Gold price) + (-24.9958 * Import)+(31.0536 * Export)+(-0.9520 * Inflation rate) + (15.1993 * GDP) + (-1.0598 * Foriegn envestment inflows) But observed that there was multicolinearity present in data, Hence we used the Principle Component Analysis.

- After removing multicolinearity we got two component and the we have fitted the model:

  Price = 31.9748 + (8.7235 × Gold price) + (6.7863 × Import)

Bibliography

Peter J Brockwell and Richard A Davis. *Introduction to time series and forecasting*. Springer, 2002.

AS Babu and SK Reddy. Exchange rate forecasting using arima. *Neural Network and Fuzzy Neuron, Journal of Stock & Forex Trading*, 4(3):01–05, 2015.

Bjoern Krollner, Bruce J Vanstone, Gavin R Finnie, et al. Financial time series forecasting with machine learning techniques: a survey. In *ESANN*, 2010.

Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.