Chapter 1

INTRODUCTION

In economies that are dependent on fossil fuel revenues, realization of long-term plans, mid-term and annual budgeting requires a fairly accurate estimation of the amount of consumption and its price fluctuations. Fossil fuels and in general, energy carriers play a key in various economic sectors and sub-sectors. Therefore, fossil fuels are of great importance for consumers, decision-makers and planners due to their value of income, consumption etc. Crude oil is a non renewable source of energy which took millions of years to form and hence it makes the quantity of crude oil available for consumption fixed and limited. But increase in consumption of crude oil in 20th century due to advancement of technology and increase in global human population there is a fast depletion in crude oil resources. Hence there is utmost need at present to study the pattern of crude oil consumption all over the world so that necessary steps and measures can be taken for wise and judicious consumption of resources to prevent it from premature exhaustion. Considering the importance of Crude Oil and the problems in this area, we can utilize the large amounts of data available showing its past consumption across the world. The forecasts can be used for judiciously handling its use in future.

1.1 Objective

Considering the importance of Crude Oil and the problems in this area, we can utilize the large amounts of data available showing its past consumption across the world. The forecasts can be used for judiciously handling its use in future.Models based on Statistical Time Series Analysis methods and Machine Learning approaches that handle large data well are proposed to forecast world crude oil consumption and country wise consumption for India, China and USA for next 5 years. Knowing the past data, we can predict the future with high approximation.

Chapter 2

TIME SERIES ANALYSIS

Time series analysis comprises methods for analyzing time series data to extract meaningful statistics and other characteristics of time series data. It focuses on comparing values of a single time series or multiple dependent time series at different points in time. The analysis of time series is based on the assumption that successive values in the data file represent consecutive measurements taken at equally spaced time intervals.

There are two main goals of time series analysis:
(a) identifying the nature of the phenomenon represented by the sequence of observations,
(b) forecasting (predicting future values of the time series variable).

Dataset that we usually get may not be necessarily be time series data.There are generally four broad classification of data :

FIGURE 2.1
Real life time series events

2.1 Time Series Data

A time series is a sequence of data points, typically consisting of successive measurements made over a time interval. Examples of time series are solar activity, ocean tides, stock market behavior, and the spread of disease. Time series are often plotted using line charts. Time series data are found in any domain of applied science and engineering which involves time-based measurements.

An example could be the set of monthly profits (both positive and negative) earned by Samsung between the 1st of October 2016 and the 1st of December 2016.

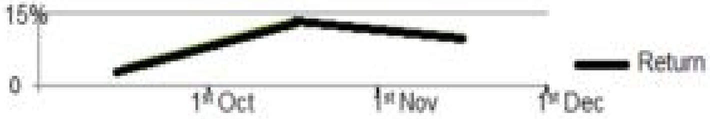


FIGURE 2.2
The above figure shows a sample time series data.

2.2 Cross-sectional data

Cross-sectional data, or a cross section of a study population, in statistics and econometrics is a type of data collected by observing many subjects (such as individuals, firms, countries, or regions) at the same point of time, or without regard to differences in time. Analysis of cross-sectional data usually consists of comparing the differences among the subjects.

Cross-sectional data differs from time series data, in which the same small-scale or aggregate entity is observed at various points in time. Another type of data, panel data (or longitudinal data), combines both cross-sectional and time series data ideas and looks at how the subjects (firms, individuals, etc.) change over time. Panel data differs from pooled cross-sectional data across time, because it deals with the observations on the same subjects in different times whereas the latter observes different subjects in different time periods. Panel analysis uses panel data to examine changes in variables over time and differences in variables between the subjects.

A good example of cross-sectional data can be the stock returns earned by shareholders of Microsoft, IBM, and Samsung as for the year ended 31st December 2015:

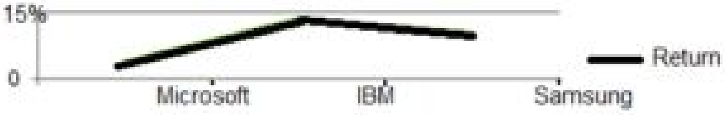


FIGURE 2.3
The above figure shows a sample cross-sectional data.

2.3 Pooled data

Pooled data is just a type of data set which is combination of both Time series data and Cross-Sectional data.

2.4 Panel data

If we were to study a particular characteristic or phenomenon across several entities over a period of time, we would end up with what's referred to as panel data.

For example, suppose we study the GDP of 3 developing countries for a period spanning 3 years, from 2015 to 2017:

Country	Year	GDP
Kenya	2015	–
Kenya	2016	–
Kenya	2017	–
India	2015	–
India	2016	–
India	2017	–
Brazil	2015	–
Brazil	2016	–
Brazil	2017	–

Here, we would study a group of entities (Kenya, India, and Brazil) over a period of time (3 yrs).This would constitute panel data.

FIGURE 2.4

The above figure shows a sample panel data.

2.5 Important terms used in Time series analysis

2.5.1 Trend

Trend represents a general systematic linear or (most often) nonlinear component that changes over time and does not repeat or at least does not repeat within the time range captured by our data (e.g., a plateau followed by a period of exponential growth).
A trend "changing direction" when it might go from an increasing trend (Positive/up secular data) to a decreasing trend (negative/down secular trend)

2.5.2 Seasonality

A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, or day of the week). Seasonality is always of a fixed and known period. When seasonality is removed from time series data its called 'Seasonally adjusted data'

2.5.3 Dependence

Dependence refers to the association of two observations with the same variable, at prior time points.

2.5.4 Stationarity

Stationarity shows the mean value of the series that remains constant over a time period; if past effects accumulate and the values increase toward infinity, then stationarity is not met.
Stationary time series is one whose properties do not depend on the time at which the series is observed. So time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times.

2.5.5 Specification

May involve the testing of the linear or non-linear relationships of dependent variables by using models such as ARIMA, ARCH, GARCH, VAR, Co-integration, etc.

2.5.6 Differencing

Used to make the series stationary, to De-trend, and to control the auto-correlations; however, some time series analyses do not require differencing and over-differenced series can produce inaccurate estimates.

2.5.7 Cyclic

A cyclic pattern exists when data exhibit rises and falls that are not of fixed period. The duration of these fluctuations is usually of at least 2 years.

Cyclic Vs Seasonal

If the fluctuations are not of fixed period then they are cyclic; if the period is unchanging and associated with some aspect of the calendar, then the pattern is seasonal. The average length of cycles is longer than the length of a seasonal pattern

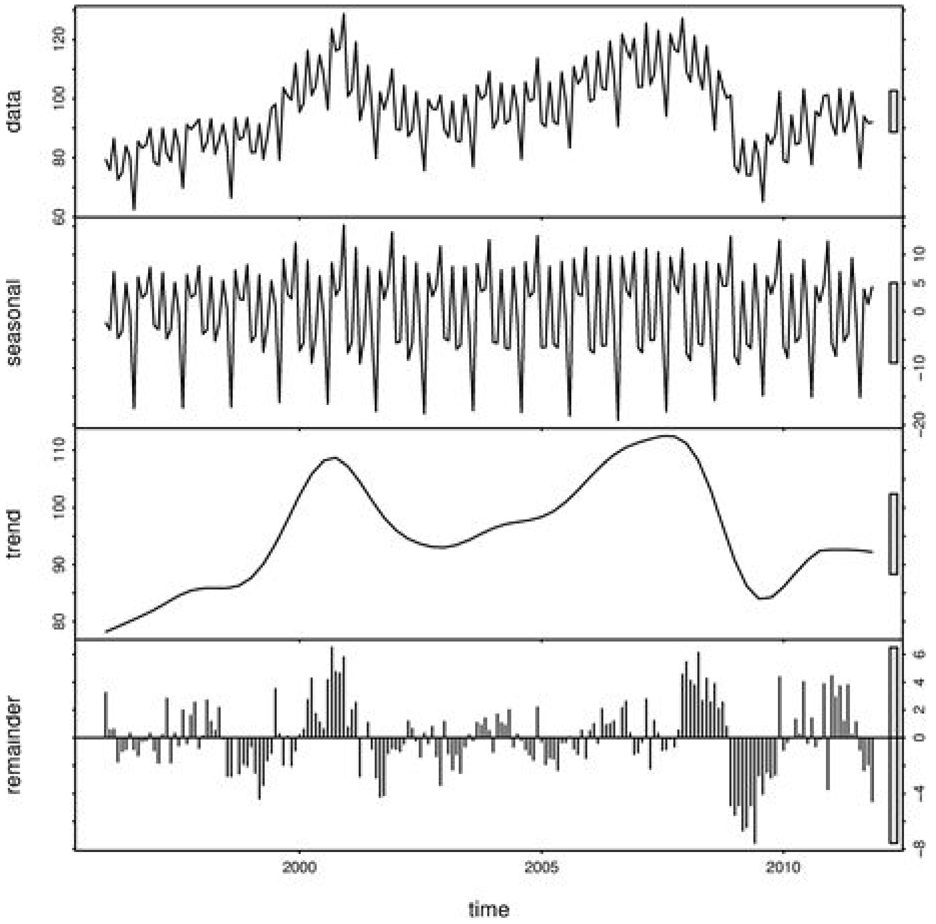


FIGURE 2.5

The above figure shows a comparison between cyclic and seasonal data.

Chapter 3

TIME SERIES FORECASTING

3.1 Introduction

Making predictions about the future is called extrapolation in the classical statistical handling of time series data.More modern fields focus on the topic and refer to it as time series forecasting.Forecasting involves taking models fit on historical data and using them to predict future observations.Descriptive models can borrow for the future (i.e. to smooth or remove noise), they only seek to best describe the data.An important distinction in forecasting is that the future is completely unavailable and must only be estimated from what has already happened.

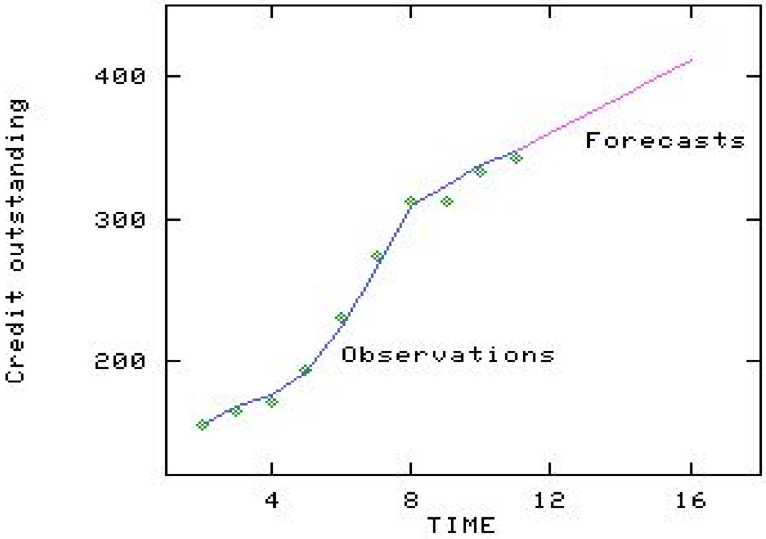
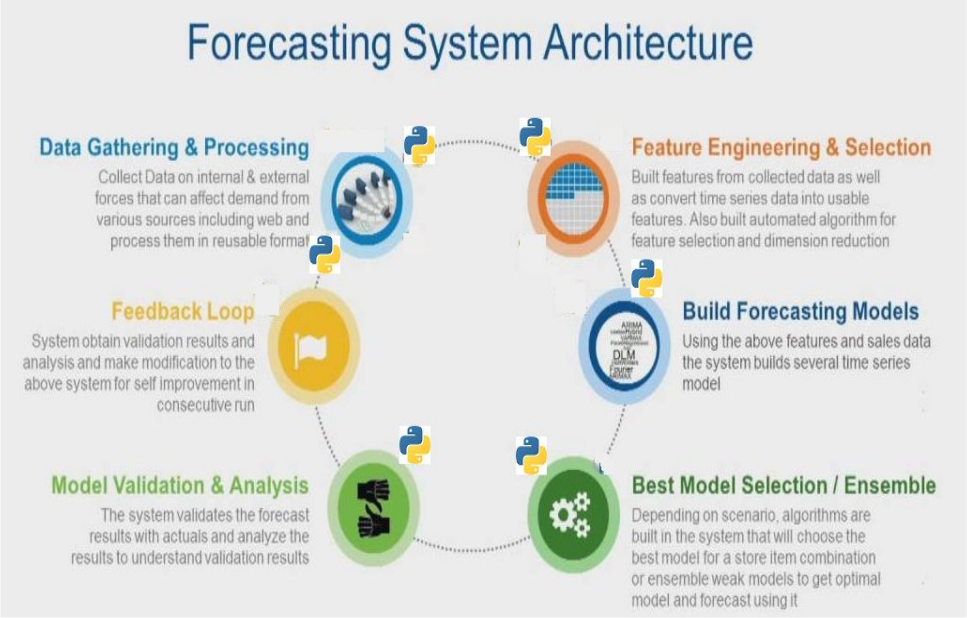


FIGURE 3.1
The above figure shows time series forecasting for a sample observation.



Forecasting System Architecture

Data Gathering & Processing

Collect Data on internal & external forces that can affect demand from various sources including web and process them in reusable format

Feature Engineering & Selection

Built features from collected data as well as convert time series data into usable features. Also built automated algorithm for feature selection and dimension reduction

Feedback Loop

System obtain validation results and analysis and make modification to the above system for self improvement in consecutive run

Build Forecasting Models

Using the above features and sales data the system builds several time series model

Model Validation & Analysis

The system validates the forecast results with actuals and analyze the results to understand validation results

Best Model Selection / Ensemble

Depending on scenario, algorithms are built in the system that will choose the best model for a store item combination or ensemble weak models to get optimal model and forecast using it

FIGURE 3.2
The above figure shows system architecture of time series forecasting.

3.2 Forecasting Methods

There are broadly two categories of methods available for time series forecasting classical and machine learning methods.

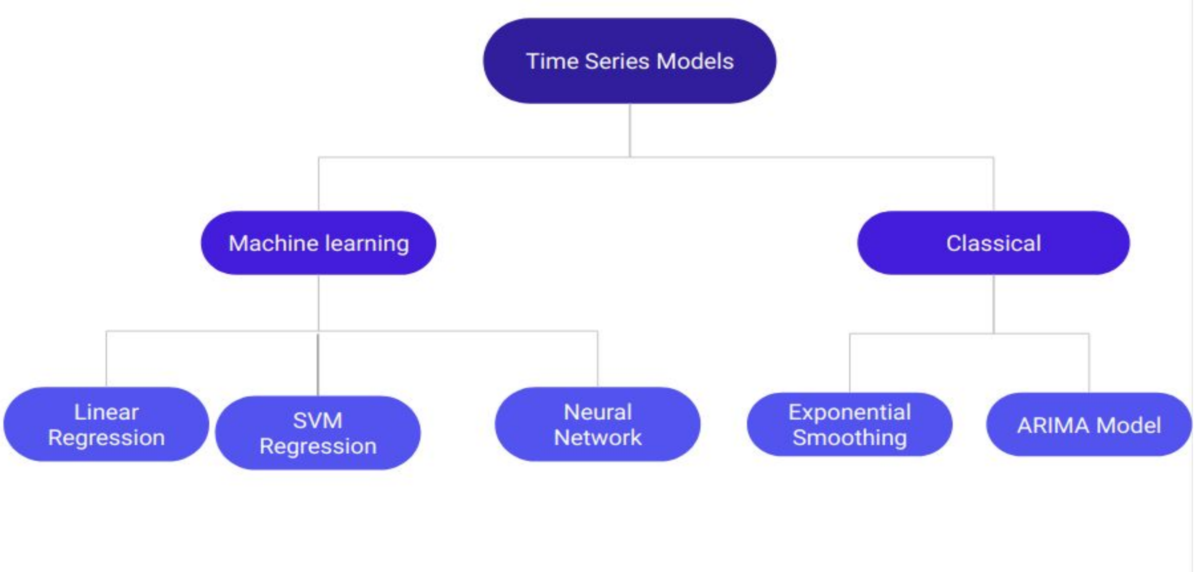


FIGURE 3.3
The above figure shows hierarchical diagram of various forecasting methods .

3.2.1 Classical Methods

Classical time series forecasting methods may be focused on linear relationships, nevertheless, they are sophisticated and perform well on a wide range of problems, assuming that your data is suitably prepared and the method is well configured.

3.2.1.1 Exponential Smoothing

Exponential smoothing has become very popular as a forecasting method for a wide variety of time series data. Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight.The method is suitable for univariate time series without trend and seasonal components.

The specific formula for simple exponential smoothing is:

$St = \alpha *Xt + (1- \alpha)*St-1$ where $0< \alpha <1$

When applied recursively to each successive observation in the series, each new smoothed value (forecast) is computed as the weighted average of the current observation and the previous smoothed observation; the previous smoothed observation was computed in turn from the previous observed value and the smoothed value before the previous observation, and so on. Thus, in effect, each smoothed value is the weighted average of the previous observations, where the weights decrease exponentially depending on the value of parameter α (alpha). If α is equal to 1 (one) then the previous observations are ignored entirely; if α is equal to 0 (zero), then the current observation is ignored entirely, and the smoothed value consists entirely of the previous smoothed value (which in turn is computed from the smoothed observation before it, and so on; thus all smoothed values will be equal to the initial smoothed value *S0*). Values of α in-between will produce intermediate results.

$$F_t = F_{t-1} + \alpha(A_{t-1} - F_{t-1})$$

where: F_t = new forecast

F_{t-1} = previous period forecast

A_{t-1} = previous period *actual* demand

α = smoothing (weighting) constant

FIGURE 3.4
The above figure shows exponential smoothing formula.

3.2.1.2 ARIMA Model

In real-life research and practice, patterns of the data are unclear, individual observations involve considerable error, and we still need not only to uncover the hidden patterns in the data but also generate forecasts. The ARIMA methodology developed by Box and Jenkins (1976) allows us to do just that; it has gained enormous popularity in many areas and research practice confirms its power and flexibility

Autoregressive moving average model. The general model introduced by Box and Jenkins (1976) includes autoregressive as well as moving average parameters, and explicitly includes differencing in the formulation of the model. Specifically, the three types of parameters in the model are: the autoregressive parameters (*p*), the number of differencing passes (*d*), and moving average parameters (*q*).

- **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 (e.g. 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 a residual error from a moving average model applied to lagged observations.

The input series for ARIMA needs to be stationary.

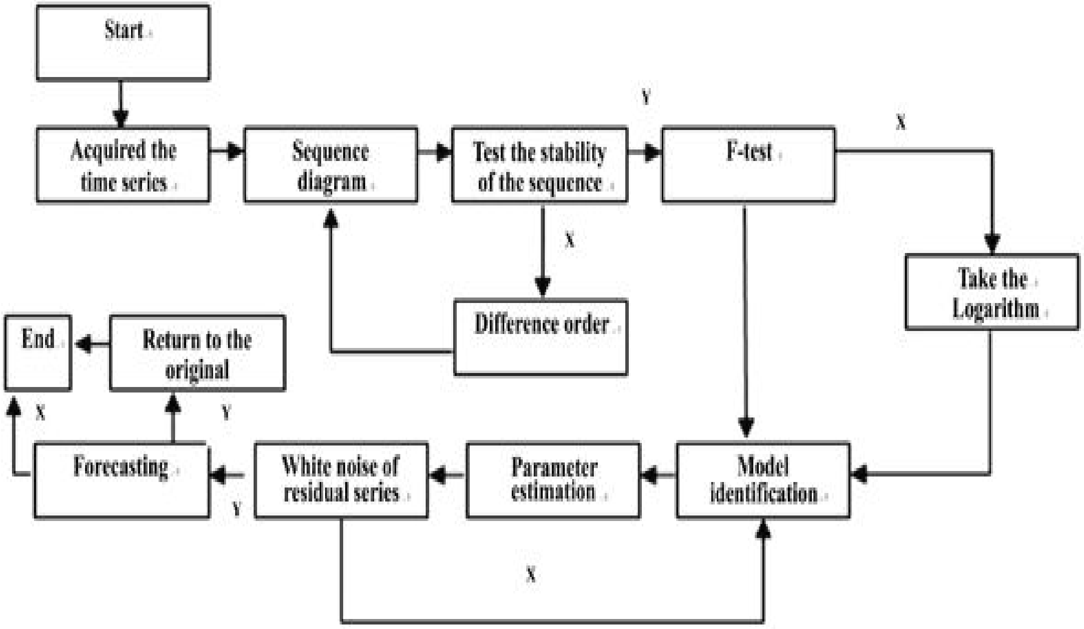


FIGURE 3.5
The above figure shows ARIMA based forecasting model.

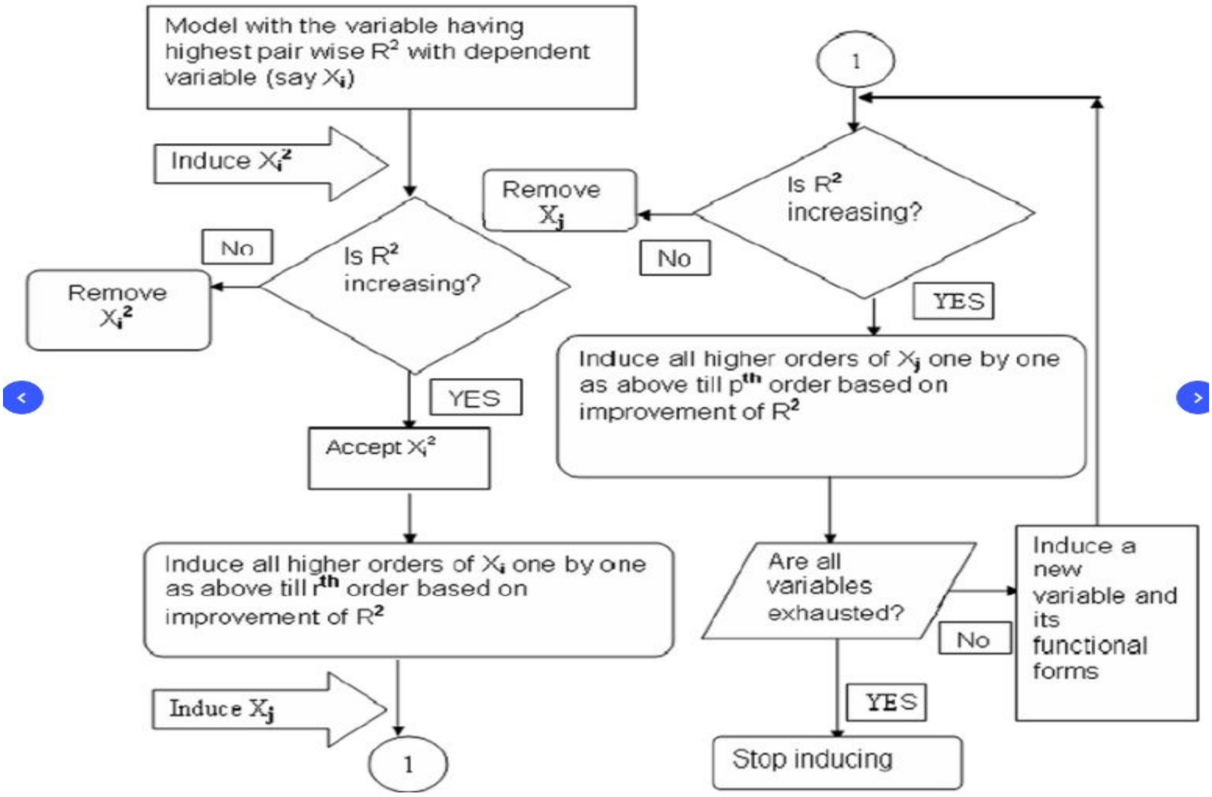
3.2.2 Machine Learning Methods

3.2.2.1 Linear Regression

Regression is the process of fitting models to data. Linear regression assumes that the relationship between the dependent variable *yi* and the independent variable *xi* is linear: *yi* = *a*+ *b xi*.

Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit.

The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.



Model with the variable having highest pair wise R² with dependent variable (say Xᵢ)
Induce Xᵢ²
No
Is R² increasing?
Remove Xᵢ²
YES
Accept Xᵢ²
Induce all higher orders of Xᵢ one by one as above till rᵗʰ order based on improvement of R²
Induce Xⱼ
1
1
Is R² increasing?
Remove Xⱼ
No
YES
Induce all higher orders of Xⱼ one by one as above till pᵗʰ order based on improvement of R²
Are all variables exhausted?
No
Induce a new variable and its functional forms
YES
Stop inducing

FIGURE 3.6

The above figure shows Linear regression based forecasting model.

3.2.2.2 Polynomial Regression

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x), and has been used to describe nonlinear phenomena such as the growth rate of tissues, the distribution of carbon isotopes in lake sediments,and the progression of disease epidemics.Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression.

The explanatory (independent) variables resulting from the polynomial expansion of the "baseline" variables are known as higher-degree terms. Such variables are also used in classification settings.The goal of regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable (or vector of independent variables) x. In simple linear regression, the model

$$y = \beta_0 + \beta_1 x + \varepsilon,$$

is used, where ε is an unobserved random error with mean zero conditioned on a scalar variable *x*. In this model, for each unit increase in the value of *x*, the conditional expectation of *y* increases by β_1 units.In many settings, such a linear relationship may not hold. For example, if we are modeling the yield of a chemical synthesis in terms of the temperature at which the synthesis takes place, we may find that the yield improves by increasing amounts for each unit increase in temperature. In this case, we might propose a quadratic model of the form:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon.$$

We can extend the this formulae upto power of n.Hence the generalised formulae of polynomial regression becomes:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

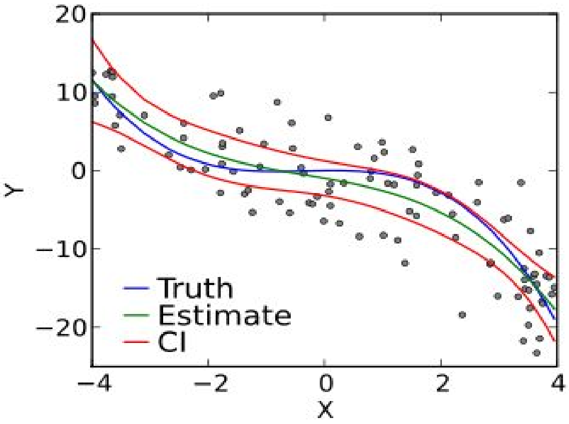


FIGURE 3.7
The above figure shows polynomial regression based forecasting model.

3.2.2.3 Support Vector Machine Regression

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences.In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. The main idea is always the same: to minimize error,individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

The interesting part about SVR is that you can deploy a non-linear kernel. In this case you end making non-linear regression, i.e. fitting a curve rather than a line. This process is based on the kernel trick and the representation of the solution/model in the dual rather than in the primal. That is, the model is represented as combinations of the training points rather than a function of the features and some weights. At the same time the basic algorithm remains the same: the only real change in the process of going non-linear is the kernel function, which changes from a simple inner product to some non linear function.

Linear SVR

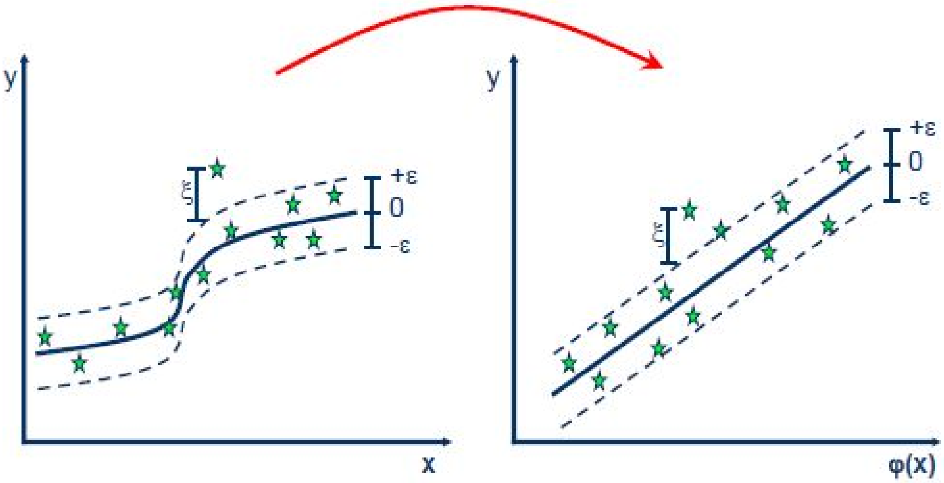
$$y = \sum_{i=1}^{N} \left(\alpha_i - \alpha_i^*\right) \cdot \left\langle x_i , x \right\rangle + b$$

Non-linear SVR

The kernel functions transform the data into a higher dimensional feature space to make it possible to perform the linear separation.

$$y = \sum_{i=1}^{N} \left(\alpha_i - \alpha_i^*\right) \cdot \langle \varphi(x_i), \varphi(x) \rangle + b$$

$$y = \sum_{i=1}^{N} \left(\alpha_i - \alpha_i^*\right) \cdot K(x_i, x) + b$$



Kernel functions

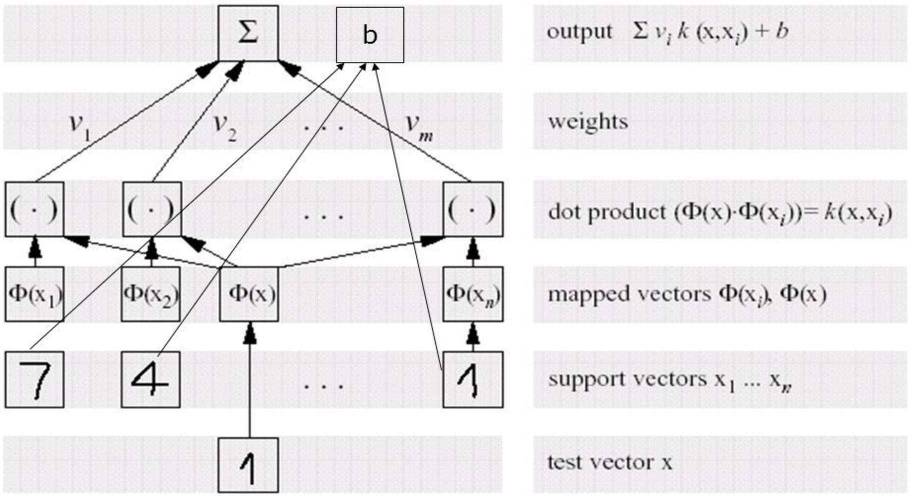
Polynomial

$$k(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i . \mathrm{x}_j)^d$$

Gaussian Radial Basis function

$$k(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(- \frac{\left\| \mathrm{x}_i - \mathrm{x}_j \right\|^2}{2\sigma^2} \right)$$

Architecture of SV Regression Machine



Σ
b
output $\Sigma\, v_i\, k\,(\mathrm{x},\mathrm{x}_i) + b$
v_1
v_2
. . .
v_m
weights
(·)
(·)
. . .
(·)
dot product $(\Phi(\mathrm{x})\cdot\Phi(\mathrm{x}_i)) = k(\mathrm{x},\mathrm{x}_i)$
$\Phi(\mathrm{x}_1)$
$\Phi(\mathrm{x}_2)$
$\Phi(\mathrm{x})$
$\Phi(\mathrm{x}_n)$
mapped vectors $\Phi(\mathrm{x}_i)$, $\Phi(\mathrm{x})$
7
4
. . .
1
support vectors $\mathrm{x}_1 \ldots \mathrm{x}_n$
1
test vector x

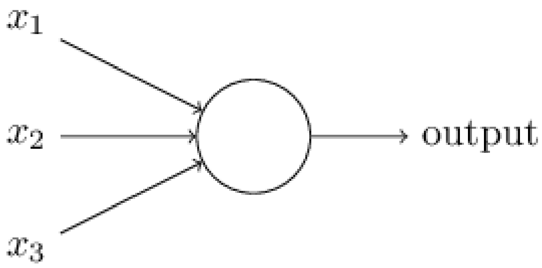
similar to regression in a three-layered neural network!?

FIGURE 3.8
The above figure shows SVR based forecasting model.

3.2.2.4 Neural Network

Perceptron is simplest neural network which were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter

Pitts. A perceptron takes several binary inputs, x1,x2,…x1,x2,…, and produces a single binary output:



In the example shown the perceptron has three inputs, x1,x2,x3x1,x2,x3. In general it could have more or fewer inputs. Rosenblatt proposed a simple rule to compute the output. He introduced *weights*, w1,w2,…w1,w2,…, real numbers expressing the importance of the respective inputs to the output.

The neuron's output, 00 or 11, is determined by whether the weighted sum ∑jwjxj is less than or greater than some *threshold value*. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

output={0 if ∑wjxj ≤ threshold

1 if ∑wjxj > threshold}

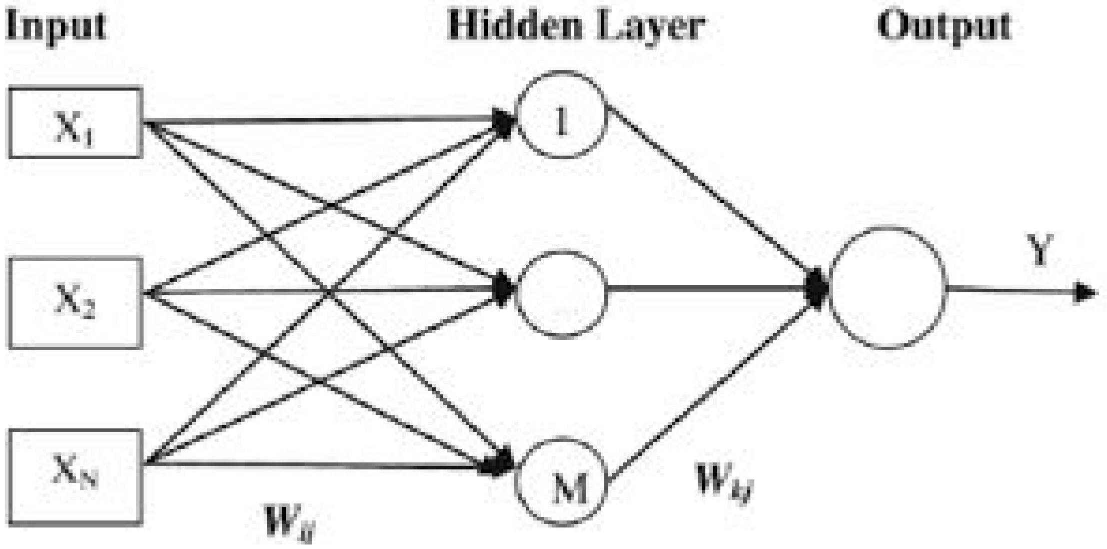


FIGURE 3.9
The above figure shows Neural network based forecasting model.

What does a neural network consist of?

A typical neural network has anything from a few dozen to hundreds, thousands, or even millions of artificial neurons called units arranged in a series of layers, each of which connects to the layers on either side. Some of them, known as input units, are designed to receive various forms of information from the outside world that the network will attempt to learn about, recognize, or otherwise process. Other units sit on the opposite side of the network and signal how it responds to the information it's learned; those are known as output units. In between the input units and output units are one or more layers of hidden units, which, together, form the majority of the artificial brain. Most neural networks are fully connected, which means each hidden unit and each output unit is connected to every unit in the layers either side. The connections between one unit and another are represented by a number called a weight, which can be either positive (if one unit excites another) or negative (if one unit suppresses or inhibits another). The higher the weight, the more influence one unit has on another. (This corresponds to the way actual brain cells trigger one another across tiny gaps called synapses.)

How does a neural network learn things?

Information flows through a neural network in two ways. When it's learning (being trained) or operating normally (after being trained), patterns of information are fed into the network via the input units, which trigger the layers of hidden units, and these in turn arrive at the output units. This common design is called a feedforward network. Not all units "fire" all the time. Each unit receives inputs from the units to its left, and the inputs are multiplied by the weights of the connections they travel along. Every unit adds up all the inputs it receives in this way and (in the simplest type of network) if the sum is more than a certain threshold value, the unit "fires" and triggers the units it's connected to (those on its right).

For a neural network to learn, there has to be an element of feedback involved—just as children learn by being told what they're doing right or wrong. In fact, we all use feedback, all the time. Think back to when you first learned to play a game like ten-pin bowling. As you picked up the heavy ball and rolled it down the alley, your brain watched how quickly the ball moved and the line it followed, and noted how close you came to knocking down the skittles. Next time it was your turn, you remembered what you'd done wrong before, modified your movements accordingly, and hopefully threw the ball a bit better. So you used feedback to compare the outcome you wanted with what actually happened, figured out the difference between the two, and used that to change what you did next time ("I need to throw it harder," "I need to roll slightly more to the left," "I need to let go later," and so on). The bigger the difference between the intended and actual outcome, the more radically you would have altered your moves.

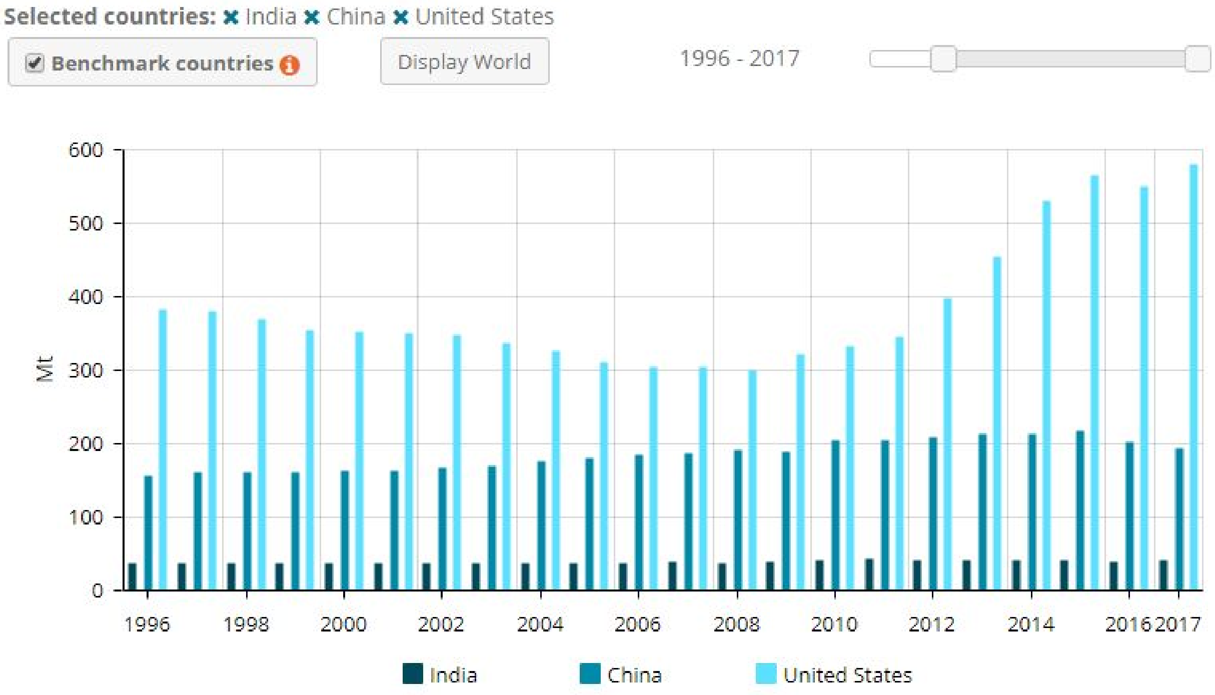
Neural networks learn things in exactly the same way, typically by a feedback process called **backpropagation**(sometimes abbreviated as "backprop"). This involves comparing the output a network produces with the output it was meant to produce, and using the difference between them to modify the weights of the connections between the units in the network, working from the output units through the hidden units to the input units—going backward, in other words. In time, backpropagation causes the network to learn, reducing the difference between actual and intended output to the point where the two exactly coincide, so the network figures things out exactly as it should.

Chapter 4

DATASET COLLECTION

4.1 Dataset choice

For this project we are using dataset of global crude oil consumption from 1996-2017 and country wise crude oil consumption for India, China and USA from Global Energy Statistical Yearbook 2018 provided by Enerdata.



Selected countries: ✖ India ✖ China ✖ United States

☑ Benchmark countries

1996 - 2017

FIGURE 4.1

Breakdown of crude oil consumption by country for India, China and USA in (Mt)

Source: https://yearbook.enerdata.net/crude-oil/world-production-statitistics.html

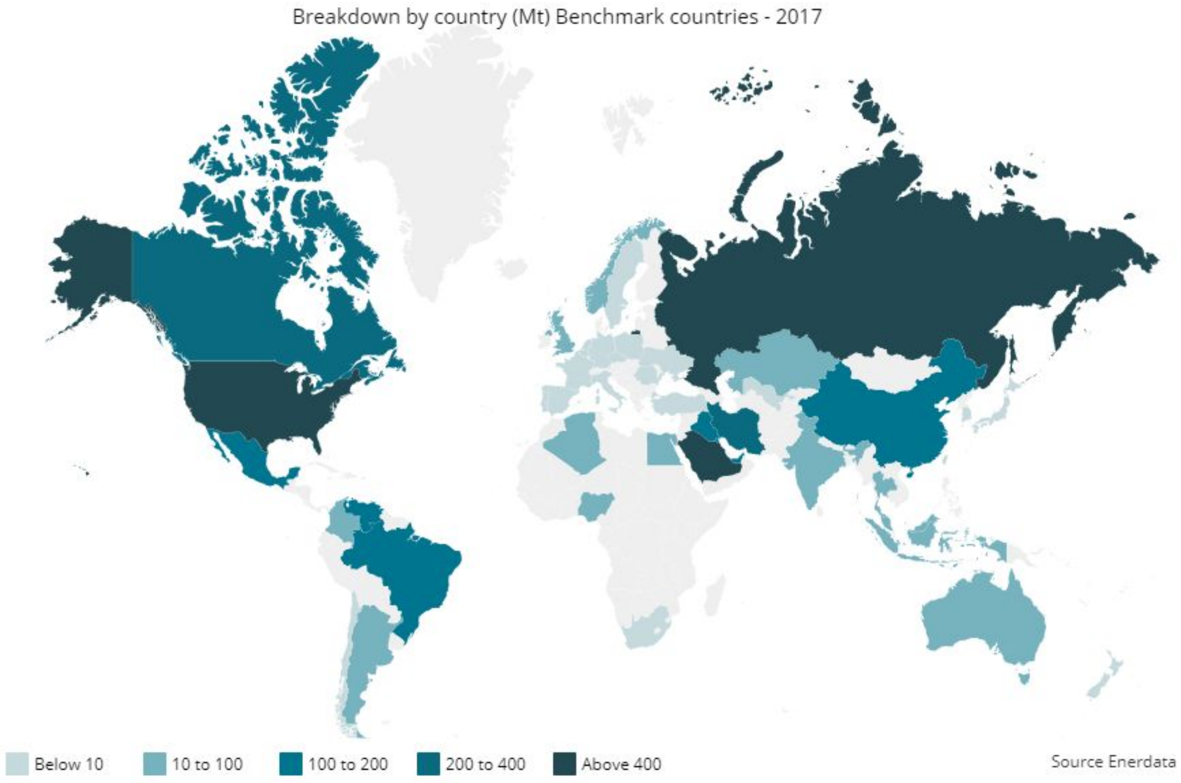
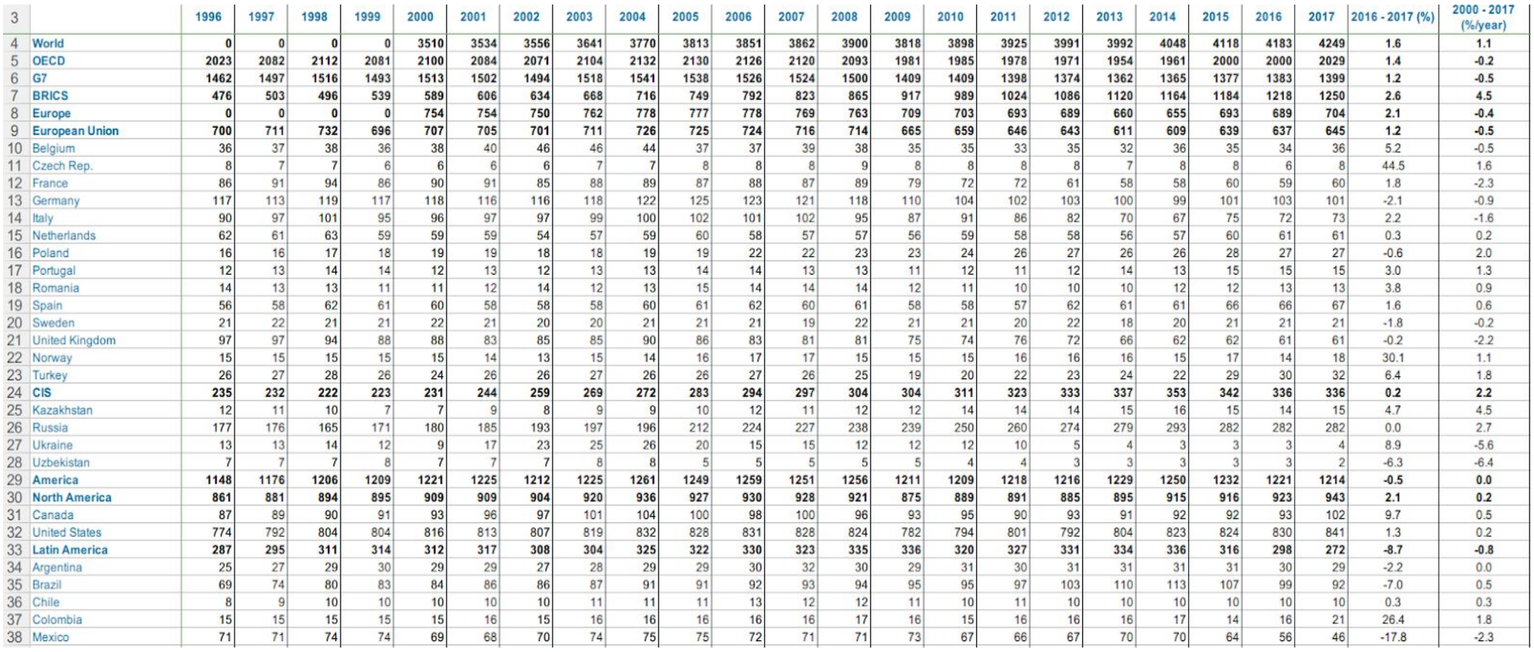


FIGURE 4.2

Source: https://yearbook.enerdata.net/crude-oil/world-production-statitistics.html



3		1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2016 - 2017 (%)	2000 - 2017 (%/year)
4	**World**	**0**	**0**	**0**	**0**	**3510**	**3534**	**3556**	**3641**	**3770**	**3813**	**3851**	**3862**	**3900**	**3818**	**3898**	**3925**	**3991**	**3992**	**4048**	**4118**	**4183**	**4249**	**1.6**	**1.1**
5	**OECD**	**2023**	**2082**	**2112**	**2081**	**2100**	**2084**	**2071**	**2104**	**2132**	**2130**	**2126**	**2120**	**2093**	**1981**	**1985**	**1978**	**1971**	**1954**	**1961**	**2000**	**2000**	**2029**	**1.4**	**-0.2**
6	**G7**	**1462**	**1497**	**1516**	**1493**	**1513**	**1502**	**1494**	**1518**	**1541**	**1538**	**1526**	**1524**	**1500**	**1409**	**1409**	**1398**	**1374**	**1362**	**1365**	**1377**	**1383**	**1399**	**1.2**	**-0.5**
7	**BRICS**	**476**	**503**	**496**	**539**	**589**	**606**	**634**	**668**	**716**	**749**	**792**	**823**	**865**	**917**	**989**	**1024**	**1086**	**1120**	**1164**	**1184**	**1218**	**1250**	**2.6**	**4.5**
8	**Europe**	**0**	**0**	**0**	**0**	**754**	**754**	**750**	**762**	**778**	**777**	**778**	**769**	**763**	**709**	**703**	**693**	**689**	**660**	**655**	**693**	**689**	**704**	**2.1**	**-0.4**
9	**European Union**	**700**	**711**	**732**	**696**	**707**	**705**	**701**	**711**	**726**	**725**	**724**	**716**	**714**	**665**	**659**	**646**	**643**	**611**	**609**	**639**	**637**	**645**	**1.2**	**-0.5**
10	Belgium	36	37	38	36	38	40	46	46	44	37	37	39	38	35	35	33	35	32	36	35	34	36	5.2	-0.5
11	Czech Rep.	8	7	7	6	6	6	6	7	7	8	8	8	9	8	8	8	8	7	8	8	6	8	44.5	1.6
12	France	86	91	94	86	90	91	85	88	89	87	88	87	89	79	72	72	61	58	58	60	59	60	1.8	-2.3
13	Germany	117	113	119	117	118	116	116	118	122	125	123	121	118	110	104	102	103	100	99	101	103	101	-2.1	-0.9
14	Italy	90	97	101	95	96	97	97	99	100	102	101	102	95	87	91	86	82	70	67	75	72	73	2.2	-1.6
15	Netherlands	62	61	63	59	59	59	54	57	59	60	58	57	57	56	59	58	58	56	57	60	61	61	0.3	0.2
16	Poland	16	16	17	18	19	19	18	18	19	19	22	22	23	23	24	26	27	26	26	28	27	27	-0.6	2.0
17	Portugal	12	13	14	14	12	13	12	13	13	14	14	13	13	11	12	11	12	14	13	15	15	15	3.0	1.3
18	Romania	14	13	13	11	11	12	14	12	13	15	14	14	14	12	11	10	10	10	12	12	13	13	3.8	0.9
19	Spain	56	58	62	61	60	58	58	58	60	61	62	60	61	58	58	57	62	61	61	66	66	67	1.6	0.6
20	Sweden	21	22	21	21	22	21	20	20	21	21	21	19	22	21	21	20	22	18	20	21	21	21	-1.8	-0.2
21	United Kingdom	97	97	94	88	88	83	85	85	90	86	83	81	81	75	74	76	72	66	62	62	61	61	-0.2	-2.2
22	Norway	15	15	15	15	15	14	13	15	14	16	17	17	15	15	15	16	16	16	15	17	14	18	30.1	1.1
23	Turkey	26	27	28	26	24	26	26	27	26	26	27	26	25	19	20	22	23	24	22	29	30	32	6.4	1.8
24	**CIS**	**235**	**232**	**222**	**223**	**231**	**244**	**259**	**269**	**272**	**283**	**294**	**297**	**304**	**304**	**311**	**323**	**333**	**337**	**353**	**342**	**336**	**336**	**0.2**	**2.2**
25	Kazakhstan	12	11	10	7	7	9	8	9	9	10	12	11	12	12	14	14	14	15	16	15	14	15	4.7	4.5
26	Russia	177	176	165	171	180	185	193	197	196	212	224	227	238	239	250	260	274	279	293	282	282	282	0.0	2.7
27	Ukraine	13	13	14	12	9	17	23	25	26	20	15	15	12	12	12	10	5	4	3	3	3	4	8.9	-5.6
28	Uzbekistan	7	7	7	8	7	7	7	8	8	5	5	5	5	5	4	4	3	3	3	3	3	2	-6.3	-6.4
29	**America**	**1148**	**1176**	**1206**	**1209**	**1221**	**1225**	**1212**	**1225**	**1261**	**1249**	**1259**	**1251**	**1256**	**1211**	**1209**	**1218**	**1216**	**1229**	**1250**	**1232**	**1221**	**1214**	**-0.5**	**0.0**
30	**North America**	**861**	**881**	**894**	**895**	**909**	**909**	**904**	**920**	**936**	**927**	**930**	**928**	**921**	**875**	**889**	**891**	**885**	**895**	**915**	**916**	**923**	**943**	**2.1**	**0.2**
31	Canada	87	89	90	91	93	96	97	101	104	100	98	100	96	93	95	90	93	91	92	92	93	102	9.7	0.5
32	United States	774	792	804	804	816	813	807	819	832	828	831	828	824	782	794	801	792	804	823	824	830	841	1.3	0.2
33	**Latin America**	**287**	**295**	**311**	**314**	**312**	**317**	**308**	**304**	**325**	**322**	**330**	**323**	**335**	**336**	**320**	**327**	**331**	**334**	**336**	**316**	**298**	**272**	**-8.7**	**-0.8**
34	Argentina	25	27	29	30	29	29	27	28	29	29	30	32	30	29	31	30	31	31	31	31	30	29	-2.2	0.0
35	Brazil	69	74	80	83	84	86	86	87	91	91	92	93	94	95	95	97	103	110	113	107	99	92	-7.0	0.5
36	Chile	8	9	10	10	10	10	10	11	11	11	13	12	12	11	10	11	10	10	10	10	10	10	0.3	0.3
37	Colombia	15	15	15	15	15	16	15	16	16	16	16	16	17	16	15	16	16	16	17	14	16	21	26.4	1.8
38	Mexico	71	71	74	74	69	68	70	74	75	75	72	71	71	73	67	66	67	70	70	64	56	46	-17.8	-2.3

FIGURE 4.3

Dataset snippet

Chapter 5

SOFTWARE REQUIREMENTS

5.1 Languages used

5.1.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. One of the aspects that makes Python such a popular choice in general, is its abundance of libraries and frameworks that facilitate coding and save development time. Machine learning and deep learning are exceptionally well catered for. NumPy, used for scientific computation, SciPy for advanced computation, and scikit-learn for data mining and data analysis, are among the most popular libraries, working alongside such heavy-hitting frameworks as TensorFlow. Python’s extensive selection of machine learning-specific libraries and frameworks simplify the development process and cut development time. Python’s simple syntax and readability promote rapid testing of complex algorithms, and make the language accessible to non-programmers. It also reduces the cognitive overhead on developers, freeing up their mental resources so that they can concentrate on problem-solving and achieving project goals.

5.2 Software Tools

5.2.1 NumPy

NumPy is a library for the Python Programming Language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high -level mathematical functions to operate on these arrays. Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

5.2.2 Pandas

Pandas is a software library written for Python for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

It provides various functionalities like:

- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- DataFrame object for data manipulation with integrated indexing.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation[4] and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.
- Provides data filtration.

5.2.3 Scikit-Learn

Scikit Learn is a free machine learning for the Python programming language. It features various regression algorithms and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

5.2.4 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source.

5.2.5 Tensorflow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

5.2.6 Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebooks documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context.

Chapter 6

Implementation and Coding

6.1 Preprocessing Dataset

Before implementing any regression model our main task was to preprocess the dataset.

Steps involved in preprocessing of dataset.

1. Making matrix of independent variables
2. Making vector of Dependent variable
3. Using mean value to compromise for any missing data value

6.2 Global crude oil consumption

For implementing this project we choose linear regression as our first model.

Steps involved in implementation of the model:

1.Preprocessing of dataset
2.Dividing entire data set into two sets(Training set and Test set) to avoid overfitting.
3.Implementing linear regression model on the Training set to train our model.
4.Checking the accuracy of our model on test data set.
5.Forecasting of consumption in next five year after getting a satisfactory result in testing phase.

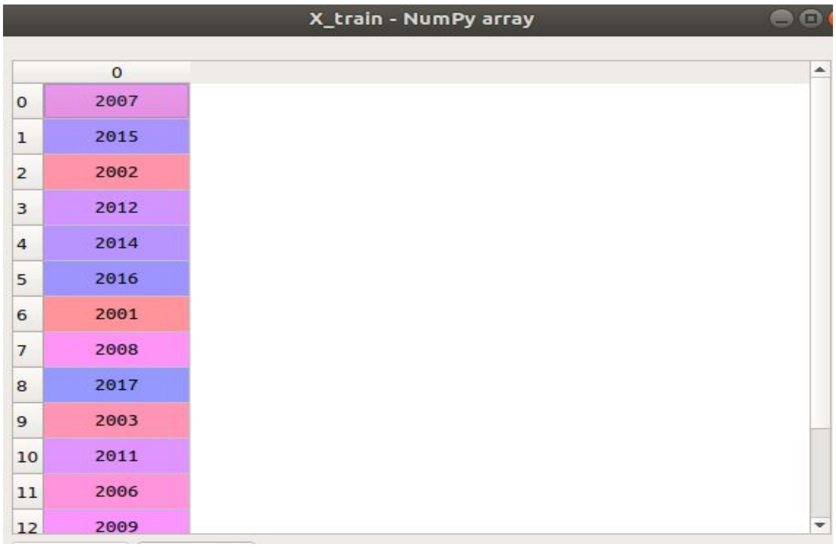
To avoid over feeding phenomenon we divided our dataset into test data set and training dataset.

Code for Preprocessing of dataset:

```
 dataset = pd.read_csv("data_formatted.csv")
   forecasting_dataset = pd.read_csv("forecasting.csv")
   """
   We are creating matrix of independent variable and vector of dependent variable
   """
   X = dataset.iloc[:,0:1].values
   Y = dataset.iloc[:,1:2].values
   X1 = forecasting_dataset.iloc[:,0:1].values
   Y1 = forecasting_dataset.iloc[:,1:2].values
```

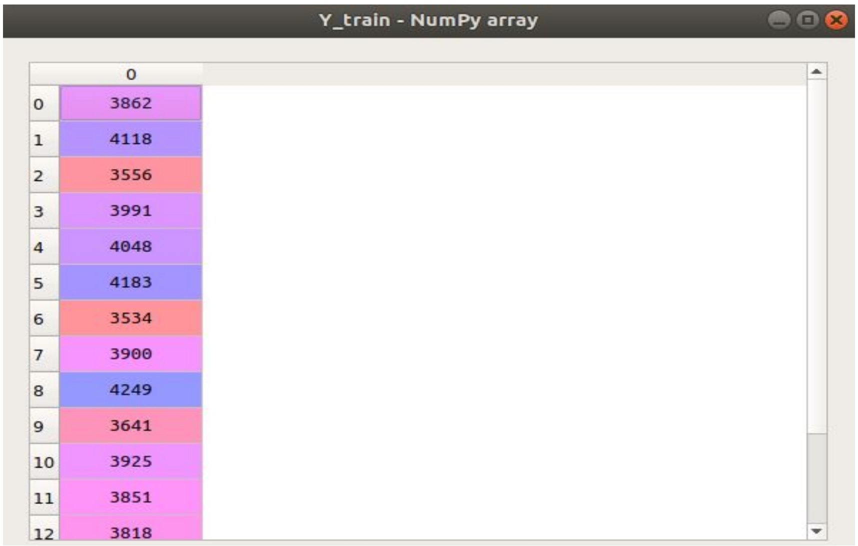
```
#   print(X1)
   #
======================================================================
===========
   # Y = dataset.iloc[:,3].values
   """
   Here we will check for exitance of any missing value and replace that missing value by
mean of the
   column
   """

   imputer = Imputer(missing_values = 'NaN',strategy='mean',axis=0)
   Y = imputer.fit_transform(Y)
   Y = imputer.transform(Y)

   """
   Here we are going to convert years to some label as numeric calculations can't be
   performed on labels
   """
   #from sklearn.preprocessing import LabelEncoder, OneHotEncoder
   #
   #labelencoder_X = LabelEncoder()
   #X[:,0]=labelencoder_X.fit_transform(X[:,0])
   """
   We don't need hot encoding yet because years do have certain weightage
   """

   """
   This is the most crucial step of data preprocessing.Here we are splitting our dataset
   into training and test dataset to avoid overfitting.Here we have chosen random_state of 42
   which is generally most suitable state for unbiased division.
   """

   X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state = 42)
```

train.py



	0
0	2007
1	2015
2	2002
3	2012
4	2014
5	2016
6	2001
7	2008
8	2017
9	2003
10	2011
11	2006
12	2009

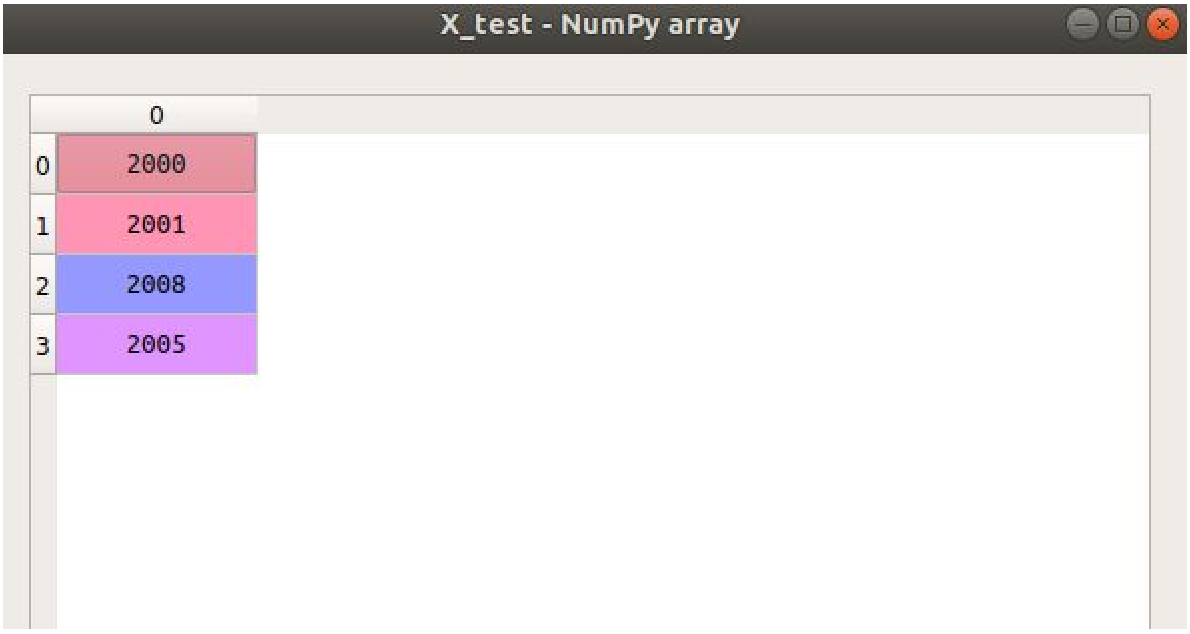
FIGURE 6.1
Training dataset(X_train)



	0
0	3862
1	4118
2	3556
3	3991
4	4048
5	4183
6	3534
7	3900
8	4249
9	3641
10	3925
11	3851
12	3818

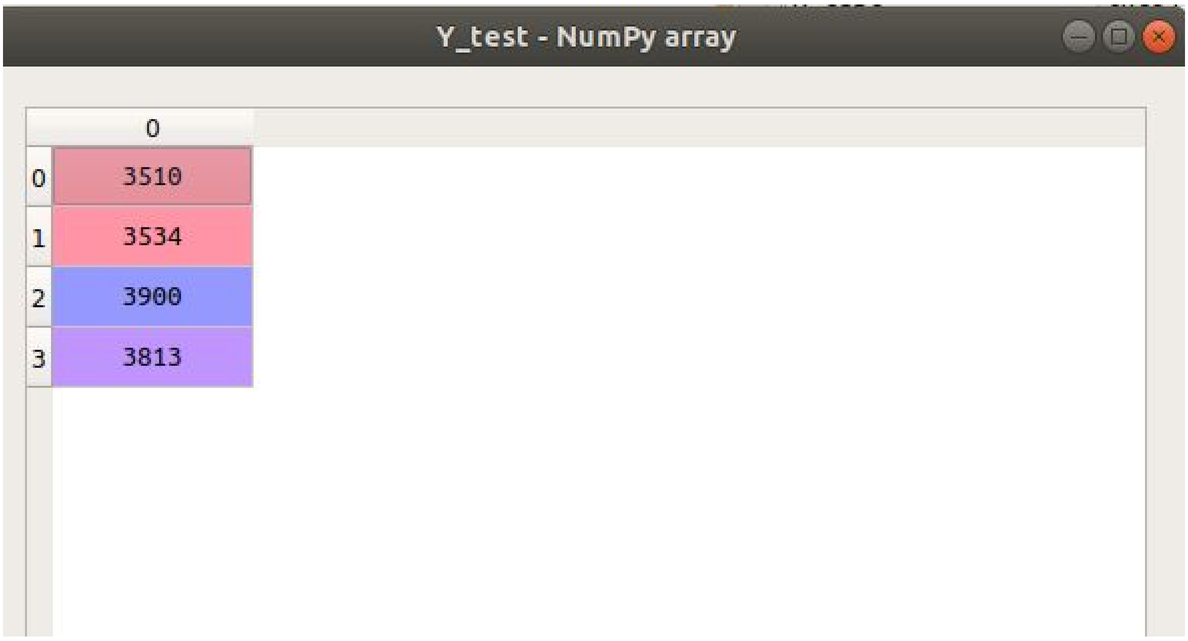
FIGURE 6.2
Training Dataset(Y_train)

test.py



	0
0	2000
1	2001
2	2008
3	2005

FIGURE 6.3
Test Dataset(X_test)



	0
0	3510
1	3534
2	3900
3	3813

FIGURE 6.4
Test Dataset(Y_test)

6.3 Source code:

6.3.1 Linear Regression Model

```
"""
Time series forecasting using Simple Linear Regression

Author:
Alok Kumar
Ayushi Saxena
Lakshita Bhargava
"""



"""



"""
Function for prediciting oil consumption Globaly
"""
def global_forcast():
   dataset = pd.read_csv("data_formatted.csv")
   forecasting_dataset = pd.read_csv("forecasting.csv")
   """
   We are creating matrix of independent variable and vector of dependent variable
   """
   X = dataset.iloc[:,0:1].values
   Y = dataset.iloc[:,1:2].values
   X1 = forecasting_dataset.iloc[:,0:1].values
   Y1 = forecasting_dataset.iloc[:,1:2].values
   #
==============================================================
===========
   # Y = dataset.iloc[:,3].values
   """
   Here we will check for exitance of any missing value and replace that missing value by
mean of the
   column
   """


   imputer = Imputer(missing_values = 'NaN',strategy='mean',axis=0)
   Y = imputer.fit_transform(Y)
   Y = imputer.transform(Y)
```

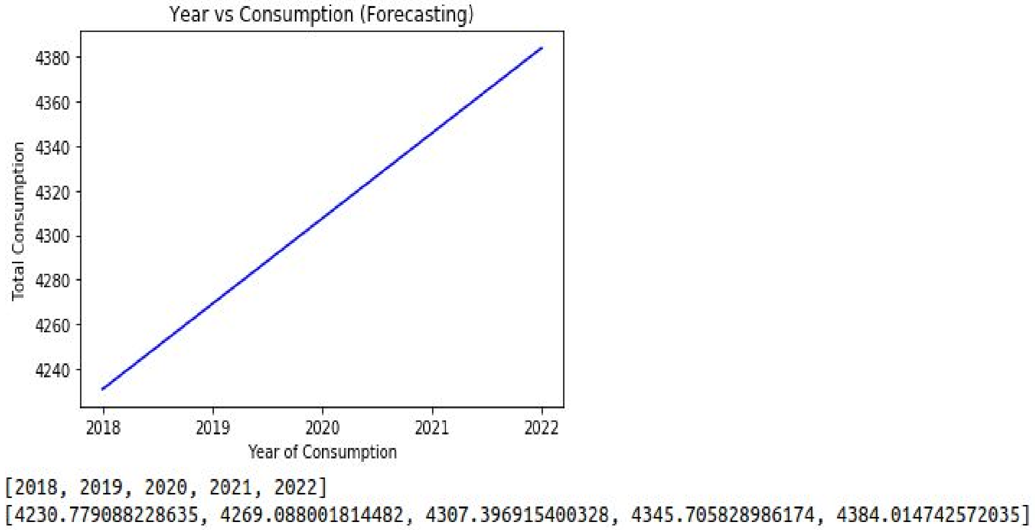
```
"""
Here we are going to convert years to some label as numeric calculations can't be
performed on labels
"""
#from sklearn.preprocessing import LabelEncoder, OneHotEncoder
#
#labelencoder_X = LabelEncoder()
#X[:,0]=labelencoder_X.fit_transform(X[:,0])
"""
We don't need hot encoding yet because years do have certain weightage
"""

"""
This is the most crucial step of data preprocessing.Here we are splitting our dataset
into training and test dataset to avoid overfitting.Here we have choosen random_state of 42
which is generally most suitable state for unbiased division.
"""

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state = 42)

"""
In this section we are going to fit the simple regression model to our training set
For this purpose we are using Linear Regression module inbuild in sklearn library

"""

regressor = LinearRegression()
regressor.fit(X_train,Y_train)

"""
After training our model on training data set we need to
test it on test_data set to see whether it is efficient or not
"""

Y_pred = regressor.predict(X_test)
Y_forecasted = regressor.predict(X1)

"""
After we have tested our model on test data we are now in position to
show some visualization of our model.For this purpose we are going to use  python library
matplotlib
"""

#Visualising the Training set results
print("\n")
print(color.BOLD+"Global Consumption Forecasting"+color.END)
print("\n")
```

```
"""
Line graph representation
"""
plt.scatter(X_train,Y_train,color="red")
plt.plot(X_train,regressor.predict(X_train),color="blue")
plt.title("Year vs Consumption (Training set)")
plt.xlabel("Year of Consumption")
plt.ylabel("Total Consumption")
locator = matplotlib.ticker.MultipleLocator(2)
plt.gca().xaxis.set_major_locator(locator)
formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
plt.gca().xaxis.set_major_formatter(formatter)
plt.show()

"""
Bar graph representation
"""
X_train_list =[]
for x in X_train.flat:
   X_train_list.append(x)
Y_train_predicted = regressor.predict(X_train)
Y_train_list = []
for y in Y_train_predicted.flat:
   Y_train_list.append(y)
#print(X_train_list)
#print(Y_train_list)
plt.scatter(X_train,Y_train,color="red")
plt.bar(X_train_list,Y_train_list)
plt.title("Year vs Consumption (Training set)")
plt.xlabel("Year of Consumption")
plt.ylabel("Total Consumption")
locator = matplotlib.ticker.MultipleLocator(2)
plt.gca().xaxis.set_major_locator(locator)
formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
plt.gca().xaxis.set_major_formatter(formatter)
plt.show()

#plt.bar(list(X_train),list(regressor.predict(X_train)),label="Bar graph")
#plt.show()



#Visualising the test set result
"""
   Tabular form
"""
X_value_list = []
for x in X_test.flat:
        X_value_list.append(x)
    Y_actual_list = []
    for y in Y_test.flat:
        Y_actual_list.append(y)
    Y_predicted_list=[]
    for y in Y_pred.flat:
        Y_predicted_list.append(y)
#
===============================================================
===========
#     print(X_value_list)
#     print(Y_actual_list)
#     print(Y_predicted_list)
#
===============================================================
===========
    actual_list=[]
    predicted_list=[]
    counter=0
    for x in X_value_list:
        actual_list.append((x,math.ceil(Y_actual_list[counter])))
        counter=counter+1
    counter=0
    for x in X_value_list:
        predicted_list.append((x,math.ceil(Y_predicted_list[counter])))
        counter=counter+1
    print("Actual Global oil Consumption Value")
    print(actual_list)
    print("\n")
    print("Predicted Global oil Consumption Value")
    print(predicted_list)
    print("\n")

    """
    Line graph representation
    """

    plt.scatter(X_test,Y_test,color="red")
    plt.plot(X_test,Y_pred,color="blue")
    plt.title("Year vs Consumption (Test set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()
    """
    Bar graph representation
    """

    X_test_list =[]
    for x in X_test.flat:
        X_test_list.append(x)
    Y_test_predicted = regressor.predict(X_test)
    Y_test_list = []
    for y in Y_pred.flat:
        Y_test_list.append(y)
    #print(X_train_list)
    #print(Y_train_list)
    plt.scatter(X_test,Y_test,color="red")
    plt.bar(X_test_list,Y_test_list)
    plt.title("Year vs Consumption (Test set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    #Forecasting the result for next five years

    """
    Line graph representation
    """

    plt.scatter(X1,Y1,color="red")
    plt.plot(X1,regressor.predict(X1),color="blue")
    plt.title("Year vs Consumption (Forecasting)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(1)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    """
    Bar graph representation
    """

    X_forecast_list =[]
#    print(X1)
    for x in X1.flat:
```

```
    X_forecast_list.append(x)
Y_test_predicted = regressor.predict(X_test)
Y_forecast_list = []
for y in Y_forecasted.flat:
    Y_forecast_list.append(y)
#print(X_forecast_list)
#print(Y_forecast_list)
print(X_forecast_list)
print(Y_forecast_list)
plt.scatter(X1,Y1,color="red")
plt.bar(X_forecast_list,Y_forecast_list)
plt.title("Year vs Consumption (Forecasting)")
plt.xlabel("Year of Consumption")
plt.ylabel("Total Consumption")
locator = matplotlib.ticker.MultipleLocator(1)
plt.gca().xaxis.set_major_locator(locator)
formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
plt.gca().xaxis.set_major_formatter(formatter)
plt.show()
```

RESULT:

Global forecast for next five years



```
[2018, 2019, 2020, 2021, 2022]
[4230.779088228635, 4269.088001814482, 4307.396915400328, 4345.705828986174, 4384.014742572035]
```

FIGURE 6.5
Line graph representation of forecasted result

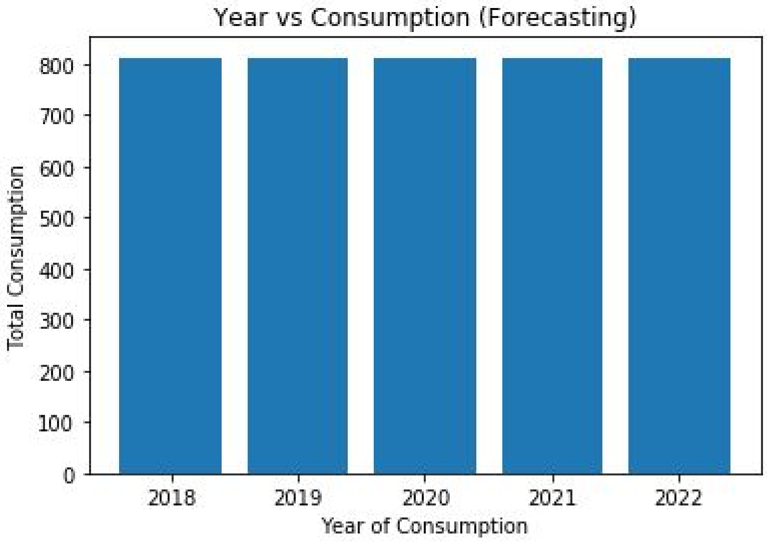


FIGURE 6.5
Bar graph representation of forecasted result

6.3.2 Polynomial Regression Model

Source Code:

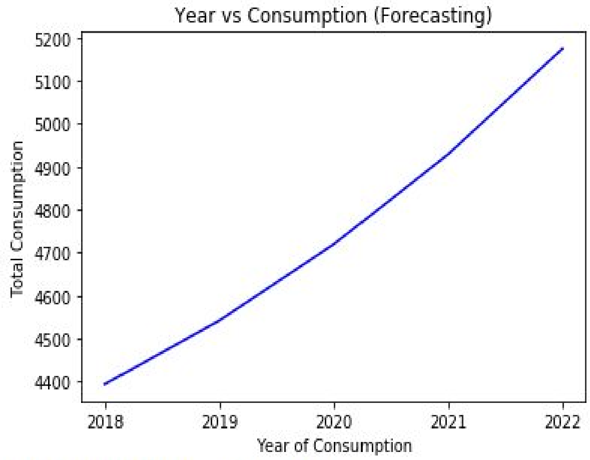
```
"""
Function for prediciting oil consumption Globaly
"""
def global_forcast():
    dataset = pd.read_csv("data_formatted.csv")
    forecasting_dataset = pd.read_csv("forecasting.csv")
    """
    We are creating matrix of independent variable and vector of dependent variable
    """
    X = dataset.iloc[:,0:1].values
    Y = dataset.iloc[:,1:2].values
    X1 = forecasting_dataset.iloc[:,0:1].values
    Y1 = forecasting_dataset.iloc[:,1:2].values
    #
================================================================
===========
    # Y = dataset.iloc[:,3].values
    """
    Here we will check for exitance of any missing value and replace that missing value by
mean of the
    column
    """

    imputer = Imputer(missing_values = 'NaN',strategy='mean',axis=0)
    Y = imputer.fit_transform(Y)
    Y = imputer.transform(Y)

    """
    Here we are going to convert years to some label as numeric calculations can't be
    performed on labels
    """
    #from sklearn.preprocessing import LabelEncoder, OneHotEncoder
    #
    #labelencoder_X = LabelEncoder()
    #X[:,0]=labelencoder_X.fit_transform(X[:,0])
    """
    We don't need hot encoding yet because years do have certain weightage
    """

    """
    This is the most crucial step of data preprocessing.Here we are splitting our dataset
    into training and test dataset to avoid overfitting.Here we have choosen random_state of 42
    which is generally most suitable state for unbiased division.
    """

    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state = 42)

    """
    In this section we are going to fit the polynomial regression model to our training set
    For this purpose we are using Regression module in built in sklearn library and polynomial
features from preprocessing library
    of sklearn
    """
    poly_reg = PolynomialFeatures(degree = 3)
    X_poly = poly_reg.fit_transform(X_train)
    poly_reg.fit(X_poly,Y_train)
    regressor = LinearRegression()
    regressor.fit(X_poly,Y_train)

    """
    After training our model on training data set we need to
    test it on test_data set to see whether it is efficient or not
    """
    X_poly_test = poly_reg.fit_transform(X_test)
    Y_pred = regressor.predict(X_poly_test)
    X_poly_forecasted = poly_reg.fit_transform(X1)
    Y_forecasted = regressor.predict(X_poly_forecasted)

    """
    After we have tested our model on test data we are now in position to
    show some visualization of our model.For this purpose we are going to use  python library
matplotlib
    """

    #Visualising the Training set results
    print("\n")
    print(color.BOLD+"Global Consumption Forecasting"+color.END)
    print("\n")

    """
    Line graph representation
    """
    plt.scatter(X_train,Y_train,color="red")
    plt.plot(X_train,regressor.predict(poly_reg.fit_transform(X_train)),color="blue")
    plt.title("Year vs Consumption (Training set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
```

```
plt.gca().xaxis.set_major_locator(locator)
formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
plt.gca().xaxis.set_major_formatter(formatter)
plt.show()

"""
Bar graph representation
"""
X_train_list =[]
for x in X_train.flat:
   X_train_list.append(x)
Y_train_predicted = regressor.predict(poly_reg.fit_transform(X_train))
Y_train_list = []
for y in Y_train_predicted.flat:
   Y_train_list.append(y)
#print(X_train_list)
#print(Y_train_list)
plt.scatter(X_train,Y_train,color="red")
plt.bar(X_train_list,Y_train_list)
plt.title("Year vs Consumption (Training set)")
plt.xlabel("Year of Consumption")
plt.ylabel("Total Consumption")
locator = matplotlib.ticker.MultipleLocator(2)
plt.gca().xaxis.set_major_locator(locator)
formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
plt.gca().xaxis.set_major_formatter(formatter)
plt.show()

#plt.bar(list(X_train),list(regressor.predict(X_train)),label="Bar graph")
#plt.show()



#Visualising the test set result
"""
   Tabular form
"""
X_value_list = []
for x in X_test.flat:
   X_value_list.append(x)
Y_actual_list = []
for y in Y_test.flat:
   Y_actual_list.append(y)
Y_predicted_list=[]
for y in Y_pred.flat:
   Y_predicted_list.append(y)
```

```
#
=================================================================================
============
#    print(X_value_list)
#    print(Y_actual_list)
#    print(Y_predicted_list)
#
=================================================================================
============
  actual_list=[]
  predicted_list=[]
  counter=0
  for x in X_value_list:
     actual_list.append((x,math.ceil(Y_actual_list[counter])))
     counter=counter+1
  counter=0
  for x in X_value_list:
     predicted_list.append((x,math.ceil(Y_predicted_list[counter])))
     counter=counter+1
  print("Actual Global oil Consumption Value")
  print(actual_list)
  print("\n")
  print("Predicted Global oil Consumption Value")
  print(predicted_list)
  print("\n")

  """
  Line graph representation
  """

  plt.scatter(X_test,Y_test,color="red")
  plt.plot(X_test,Y_pred,color="blue")
  plt.title("Year vs Consumption (Test set)")
  plt.xlabel("Year of Consumption")
  plt.ylabel("Total Consumption")
  locator = matplotlib.ticker.MultipleLocator(2)
  plt.gca().xaxis.set_major_locator(locator)
  formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
  plt.gca().xaxis.set_major_formatter(formatter)
  plt.show()

  """
  Bar graph representation
  """

  X_test_list =[]
  for x in X_test.flat:
     X_test_list.append(x)
  Y_test_predicted = regressor.predict(poly_reg.fit_transform(X_test))
  Y_test_list = []
  for y in Y_pred.flat:
     Y_test_list.append(y)
  #print(X_train_list)
  #print(Y_train_list)
  plt.scatter(X_test,Y_test,color="red")
  plt.bar(X_test_list,Y_test_list)
  plt.title("Year vs Consumption (Test set)")
  plt.xlabel("Year of Consumption")
  plt.ylabel("Total Consumption")
  locator = matplotlib.ticker.MultipleLocator(2)
  plt.gca().xaxis.set_major_locator(locator)
  formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
  plt.gca().xaxis.set_major_formatter(formatter)
  plt.show()

  #Forecasting the result for next ten years

  """
  Line graph representation
  """

  plt.scatter(X1,Y1,color="red")
  plt.plot(X1,regressor.predict(poly_reg.fit_transform(X1)),color="blue")
  plt.title("Year vs Consumption (Forecasting)")
  plt.xlabel("Year of Consumption")
  plt.ylabel("Total Consumption")
  locator = matplotlib.ticker.MultipleLocator(1)
  plt.gca().xaxis.set_major_locator(locator)
  formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
  plt.gca().xaxis.set_major_formatter(formatter)
  plt.show()

  """
  Bar graph representation
  """

  X_forecast_list =[]
  for x in X1.flat:
     X_forecast_list.append(x)
#   Y_test_predicted = regressor.predict(X_test)
  Y_forecast_list = []
  for y in Y_forecasted.flat:
     Y_forecast_list.append(y)
  print(X_forecast_list)
  print(Y_forecast_list)
  plt.scatter(X1,Y1,color="red")
  plt.bar(X_forecast_list,Y_forecast_list)
  plt.title("Year vs Consumption (Forecasting)")
  plt.xlabel("Year of Consumption")
  plt.ylabel("Total Consumption")
  locator = matplotlib.ticker.MultipleLocator(1)
  plt.gca().xaxis.set_major_locator(locator)
  formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
  plt.gca().xaxis.set_major_formatter(formatter)
  plt.show()
```

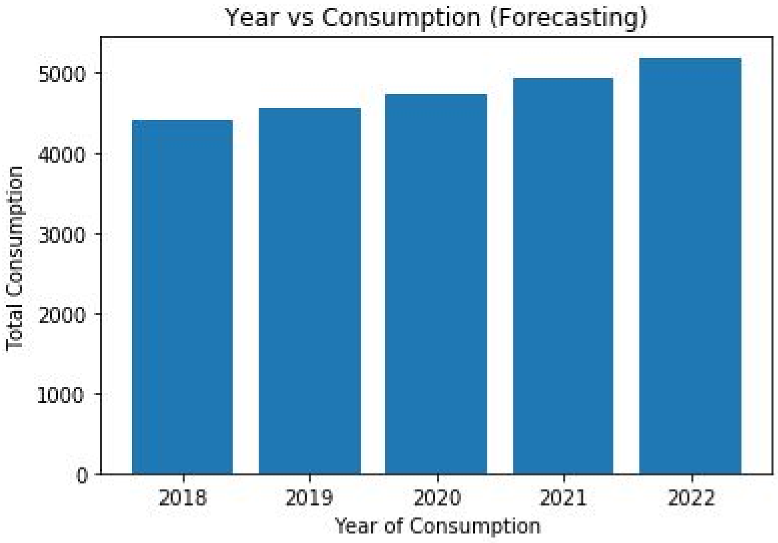
RESULT:

Global Forecast for next five years



```
[2018, 2019, 2020, 2021, 2022]
[4394.190323352814, 4541.86482667923, 4719.050760746002, 4928.779089450836, 5174.080763339996]
```

FIGURE 6.6
Line graph representation of forecasted result



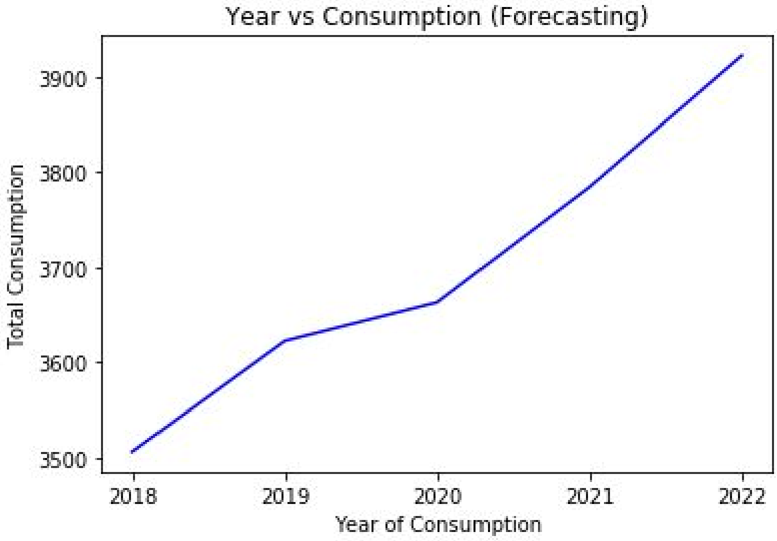
6.3.3 Support vector regression Model

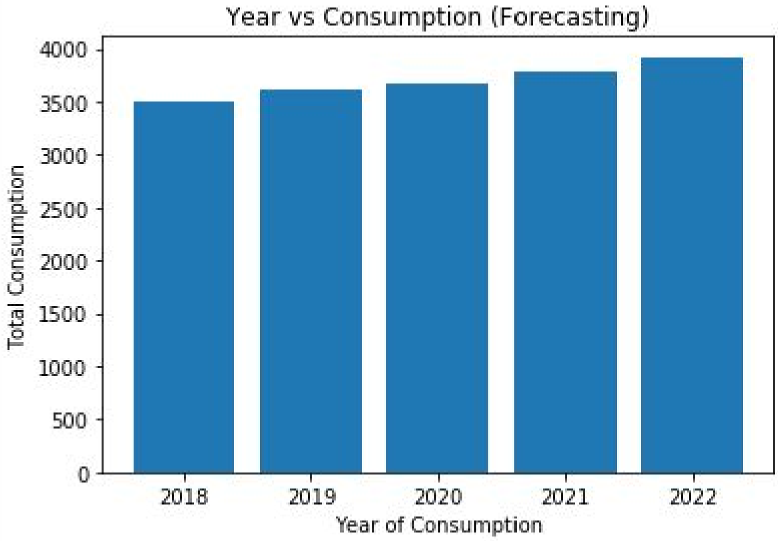
SOURCE CODE

```
def global_forcast():
    dataset = pd.read_csv("data_formatted.csv")
    forecasting_dataset = pd.read_csv("forecasting.csv")
    """
    We are creating matrix of independent variable and vector of dependent variable
    """
    X = dataset.iloc[:,0:1].values
    Y = dataset.iloc[:,1:2].values
    X1 = forecasting_dataset.iloc[:,0:1].values
    Y1 = forecasting_dataset.iloc[:,1:2].values
#   print(X1)
    #
==============================================================================
    # Y = dataset.iloc[:,3].values
    """
    Here we will check for exitance of any missing value and replace that missing value by mean of the
    column
    """

    imputer = Imputer(missing_values = 'NaN',strategy='mean',axis=0)
    Y = imputer.fit_transform(Y)
    Y = imputer.transform(Y)

    """
    Here we are going to convert years to some label as numeric calculations can't be
    performed on labels
    """
    #from sklearn.preprocessing import LabelEncoder, OneHotEncoder
    #
    #labelencoder_X = LabelEncoder()
    #X[:,0]=labelencoder_X.fit_transform(X[:,0])
    """
    We don't need hot encoding yet because years do have certain weightage
    """

    """
    This is the most crucial step of data preprocessing.Here we are splitting our dataset
    into training and test dataset to avoid overfitting.Here we have choosen random_state of 42
    which is generally most suitable state for unbiased division.
    """

    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state = 42)

    """
        As SVR library does not have inbuild feature scaling so we need to perfrom
        the feature scaling manually in this case.
    """
    print(X1)
    sc_X = StandardScaler()
    sc_Y = StandardScaler()
    sc_X1 = StandardScaler()
    sc_Y1 = StandardScaler()
    sc_X2 = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    Y_train = sc_Y.fit_transform(Y_train)
    X_test = sc_X1.fit_transform(X_test)
    Y_test = sc_Y1.fit_transform(Y_test)
    X1 = sc_X2.fit_transform(X1)
  #  Y1 = sc_Y2.fit_transform(Y1)

    """
    In this section we are going to fit the SVR model to our training set
    For this purpose we are using SVM module inbuild in sklearn library

    """

    regressor = SVR(kernel = 'rbf')
    regressor.fit(X_train,Y_train)

    """
    After training our model on training data set we need to
    test it on test_data set to see whether it is efficient or not
    """

    Y_pred = regressor.predict(X_test)
    Y_forecasted = regressor.predict(X1)

    """
    After we have tested our model on test data we are now in position to
    show some visualization of our model.For this purpose we are going to use  python library matplotlib
    """

    #Visualising the Training set results
    print("\n")
    print(color.BOLD+"Global Consumption Forecasting"+color.END)
    print("\n")

    """
    Line graph representation
    """
    Y_train_transform = sc_Y.inverse_transform(Y_train)
    X_train_transform = sc_X.inverse_transform(X_train)
    plt.scatter(X_train_transform,Y_train_transform,color="red")
    plt.plot(X_train_transform,sc_Y.inverse_transform(regressor.predict(X_train)),color="blue")

    plt.title("Year vs Consumption (Training set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    """
    Bar graph representation
    """
    X_train_list =[]
    for x in X_train_transform.flat:
        X_train_list.append(x)
    Y_train_predicted_transform = sc_Y.inverse_transform(regressor.predict(X_train))
    Y_train_list = []
    for y in Y_train_predicted_transform.flat:
        Y_train_list.append(y)
    #print(X_train_list)
    #print(Y_train_list)
    plt.scatter(X_train_transform,Y_train_transform,color="red")
    plt.bar(X_train_list,Y_train_list)
    plt.title("Year vs Consumption (Training set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    #plt.bar(list(X_train),list(regressor.predict(X_train)),label="Bar graph")
    #plt.show()



    #Visualising the test set result
    """
        Tabular form
    """
    Y_test_transform = sc_Y1.inverse_transform(Y_test)
    Y_pred_transform = sc_Y1.inverse_transform(Y_pred)
    X_test_transform = sc_X1.inverse_transform(X_test)
    print(X_test_transform)
    X_value_list = []
    for x in X_test_transform.flat:
        X_value_list.append(x)
    Y_actual_list = []
    for y in Y_test_transform.flat:
        Y_actual_list.append(y)
    Y_predicted_list=[]
    for y in Y_pred_transform.flat:
        Y_predicted_list.append(y)
    print(X_value_list)
# =============================================================================
#     print(X_value_list)
#     print(Y_actual_list)
#     print(Y_predicted_list)
# =============================================================================
    actual_list=[]
    predicted_list=[]
    counter=0
    for x in X_value_list:
        actual_list.append((x,math.ceil(Y_actual_list[counter])))
        counter=counter+1
    counter=0
    for x in X_value_list:
        predicted_list.append((x,math.ceil(Y_predicted_list[counter])))
        counter=counter+1
    print("Actual Global oil Consumption Value")
    print(actual_list)
    print("\n")
    print("Predicted Global oil Consumption Value")
    print(predicted_list)
    print("\n")

    """
    Line graph representation
    """

    plt.scatter(X_test_transform,Y_test_transform,color="red")
    plt.plot(X_test_transform,Y_pred_transform,color="blue")
    plt.title("Year vs Consumption (Test set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    """
    Bar graph representation
    """

    X_test_list =[]
    for x in X_test_transform.flat:
        X_test_list.append(x)
    Y_test_predicted = regressor.predict(X_test)
    Y_test_list = []
    for y in Y_pred_transform.flat:
        Y_test_list.append(y)
    #print(X_train_list)
    #print(Y_train_list)
    plt.scatter(X_test_transform,Y_test_transform,color="red")
    plt.bar(X_test_list,Y_test_list)
    plt.title("Year vs Consumption (Test set)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(2)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    #Forecasting the result for next five years

    """
    Line graph representation
    """
#    Y1_transform = sc_Y2.inverse_transform(Y1)
    Y1_pred_transform = sc_Y1.inverse_transform(regressor.predict(X1))
    X1_transform = sc_X2.inverse_transform(X1)
    print(X1_transform)
#    plt.scatter(X1,Y1,color="red")
    plt.plot(X1_transform, Y1_pred_transform,color="blue")
    plt.title("Year vs Consumption (Forecasting)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(1)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()

    """
    Bar graph representation
    """

    X_forecast_list =[]
    for x in X1_transform.flat:
        X_forecast_list.append(x)
    Y_forecast_list = []
    for y in  Y1_pred_transform.flat:
        Y_forecast_list.append(y)
    #print(X_forecast_list)
    #print(Y_forecast_list)
#    plt.scatter(X1,Y1,color="red")
    plt.bar(X_forecast_list,Y_forecast_list)
    plt.title("Year vs Consumption (Forecasting)")
    plt.xlabel("Year of Consumption")
    plt.ylabel("Total Consumption")
    locator = matplotlib.ticker.MultipleLocator(1)
    plt.gca().xaxis.set_major_locator(locator)
    formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
    plt.gca().xaxis.set_major_formatter(formatter)
    plt.show()
```

RESULT:

Global consumption forecasting in next five years





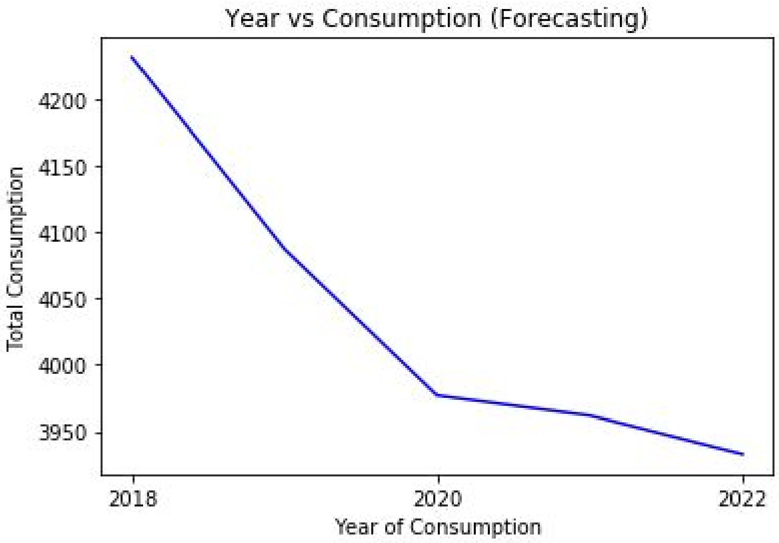
6.3.4 Exponential smoothing model

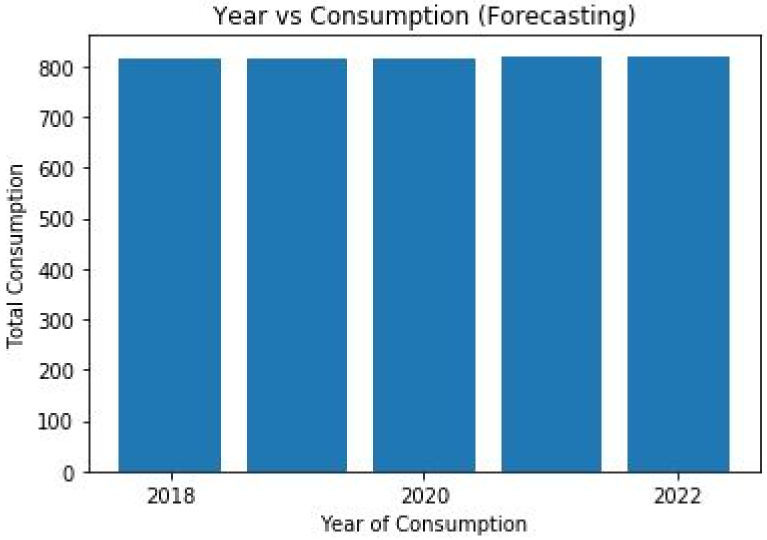
SOURCE CODE:

```
  """

  In this section of the code we will apply the exponential smoothing model on our training data

  Fomulae to be used:

  F(t)=β(A(t-1))+(1-β)F(t-1)

  where β is exponential smoothing constant

  Here we are taking the value of β = 0.2

  """

  x = 0.2

  Y_train_predicted = []

  counter = 0

  for y in Y_train_list:

      if counter == 0:

          Y_train_predicted.append(math.ceil(Y_train_list[counter]))

      else:

          res = x*(Y_train_list[counter-1])+(1-x)*Y_train_predicted[counter-1]

          Y_train_predicted.append(math.ceil(res))

      counter+=1

#   print( Y_train_predicted)

  print("Global oil consumption forecasting")
```

RESULT:

Global Forecasting result for next five years:





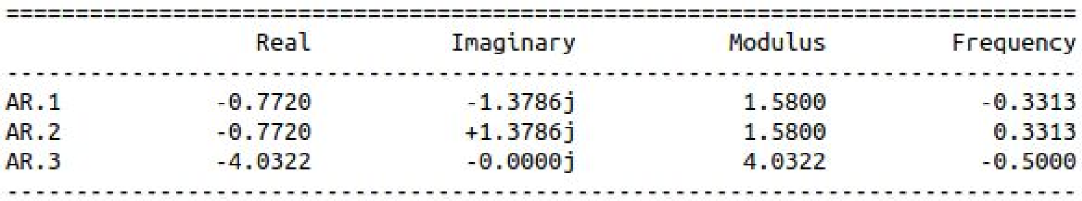
6.3.5 ARIMA model

SOURCE CODE:

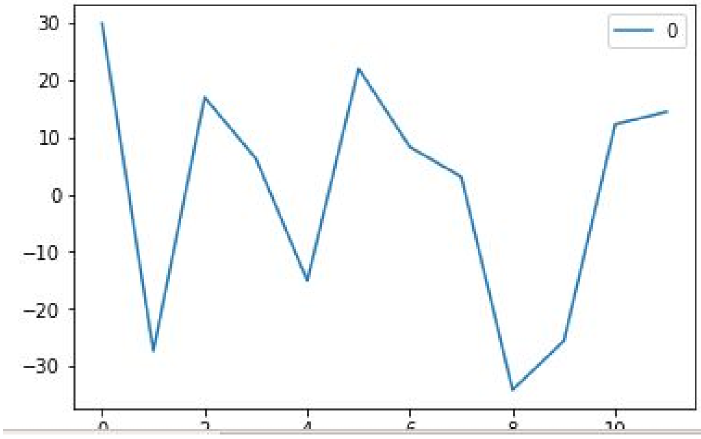
```
model = ARIMA(X_train, order=(5,1,0))

  model_fit = model.fit(disp=0)

  print(model_fit.summary())

  # plot residual errors

  residuals = DataFrame(model_fit.resid)

  residuals.plot()

  pyplot.show()

  residuals.plot(kind='kde')

  pyplot.show()

  print(residuals.describe())

  #Visualizing the result on test set

  X_test_list = []

  for x in X_test.flat:

      X_test_list.append(x)

  Y_pred = [3320,3400,3893,3895]

  Y_forecast = [4394.190,4421.864,4719.0507607,4628.7790,5074.080]

  X_forecast = []

  for x in X1.flat:

      X_forecast.append(x)
```

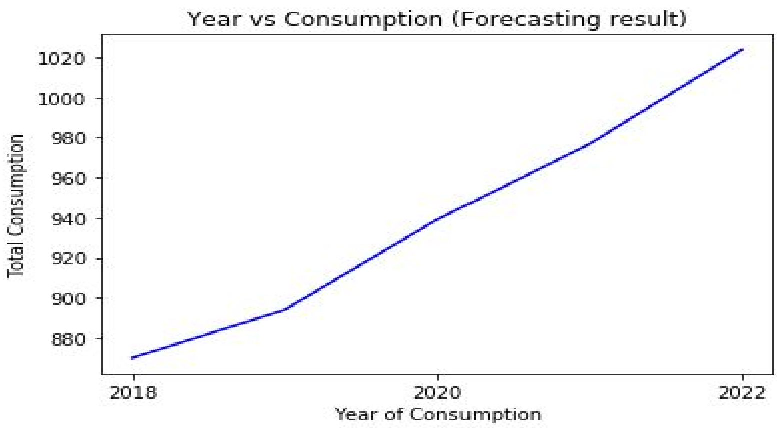
RESULT:

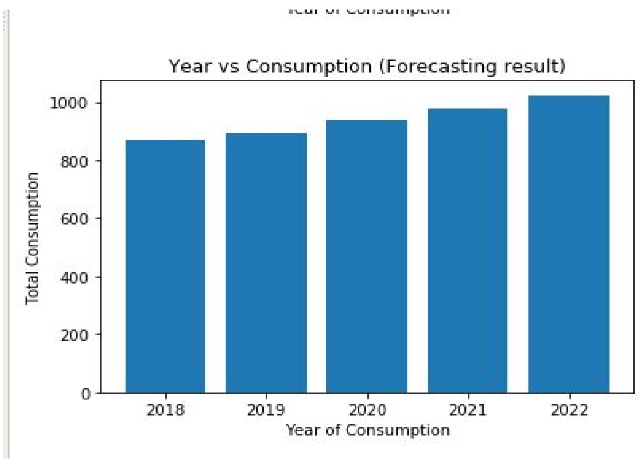
Global oil consumption forecasting in next five years



```
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.7720            -1.3786j           1.5800           -0.3313
AR.2           -0.7720            +1.3786j           1.5800            0.3313
AR.3           -4.0322            -0.0000j           4.0322           -0.5000
-----------------------------------------------------------------------------
```







Chapter 7

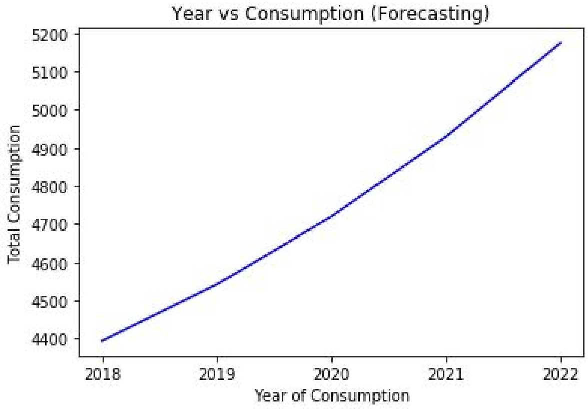
CONCLUSION

7.1 CONCLUSION

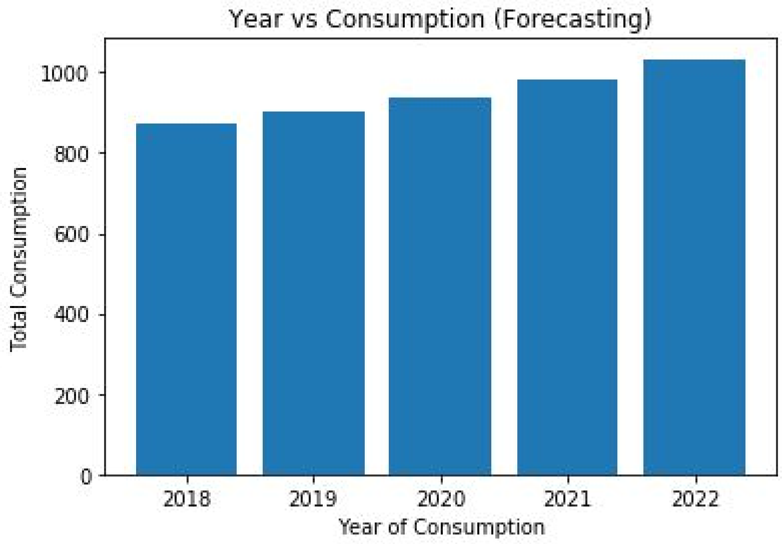
After performing time series forecasting for three different models we got best result with least error for Polynomial regression model.

So we are taking the values predicted by polynomial regression model as our final predicted value.

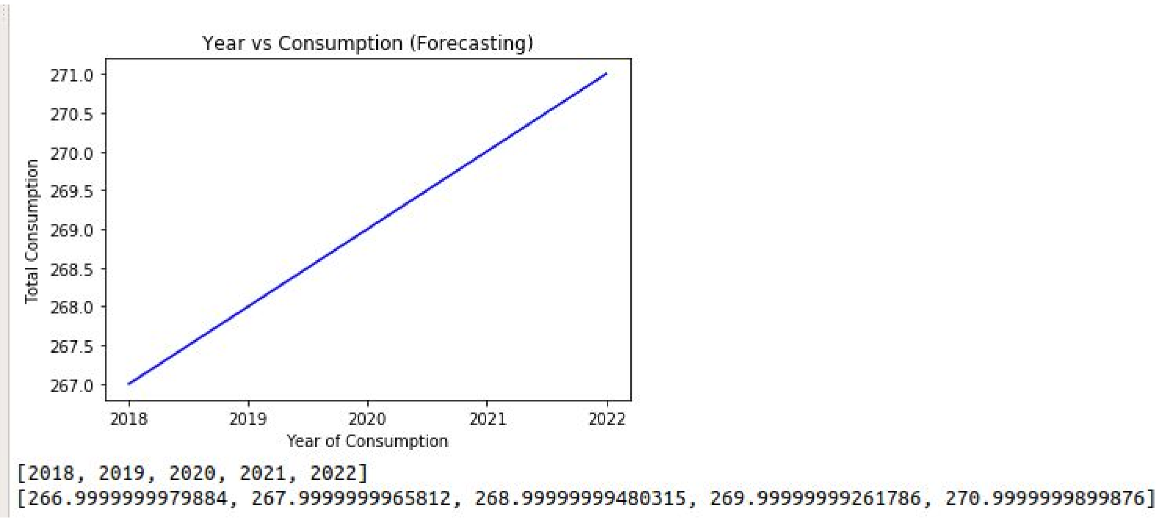
Global Crude oil consumption forecasting for next five years:



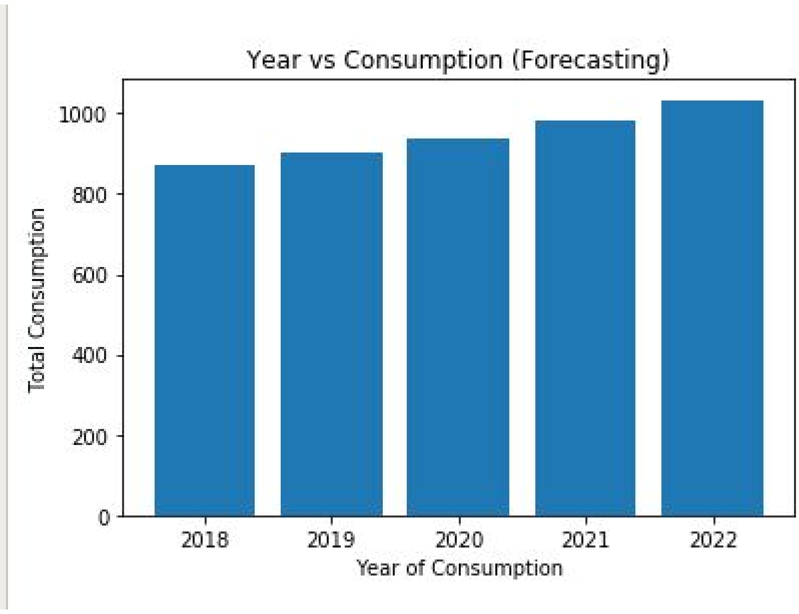
```
[2018, 2019, 2020, 2021, 2022]
[4394.190323352814, 4541.86482667923, 4719.050760746002, 4928.779089450836, 5174.080763339996]
```



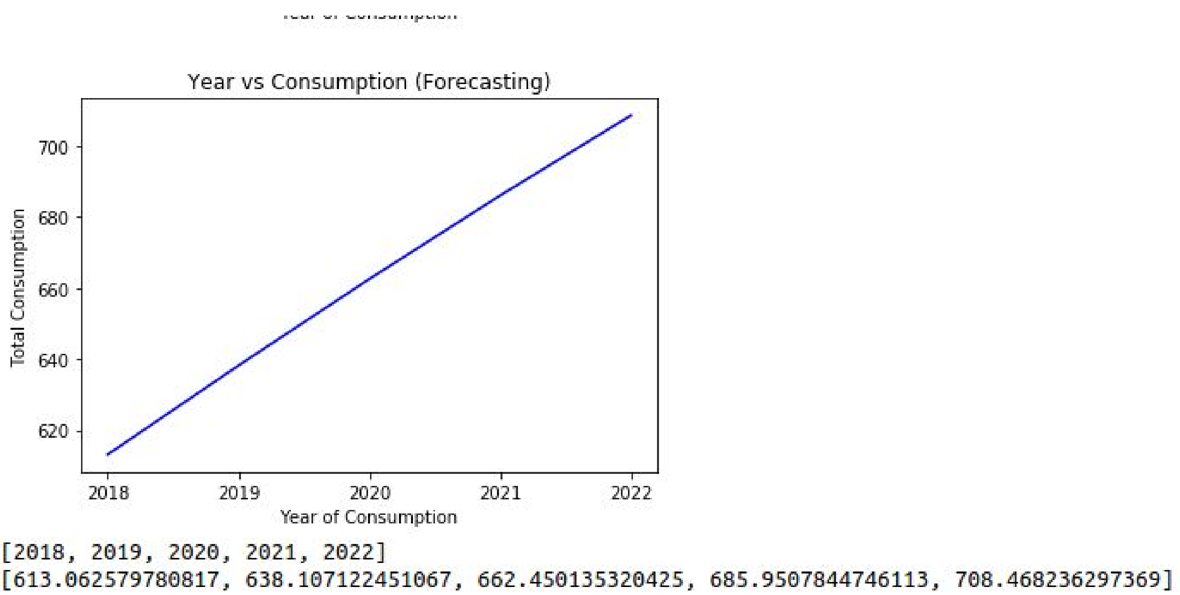
Crude oil consumption forecasting for next five years in India:



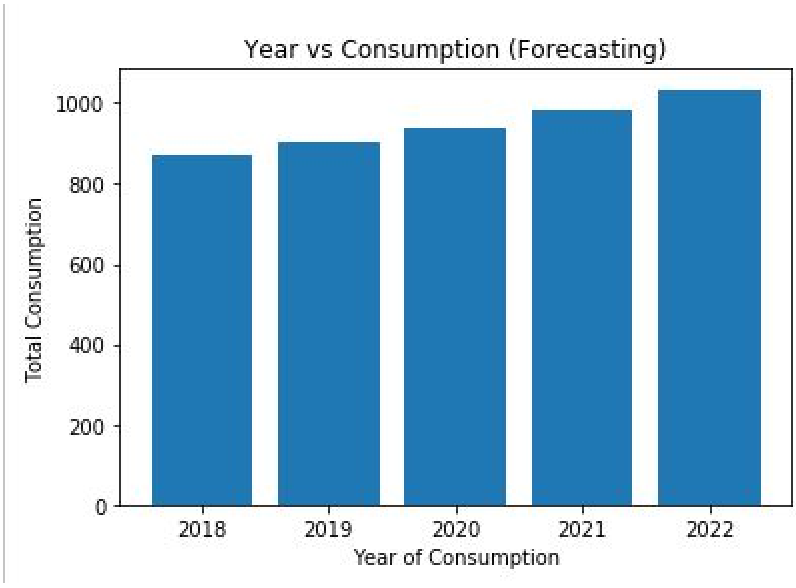
```
[2018, 2019, 2020, 2021, 2022]
[266.9999999979884, 267.9999999965812, 268.99999999480315, 269.99999999261786, 270.9999999899876]
```



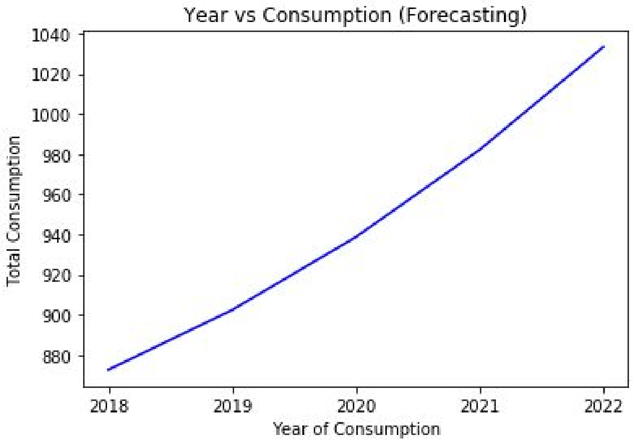
Crude oil consumption forecasting for next five years in China:



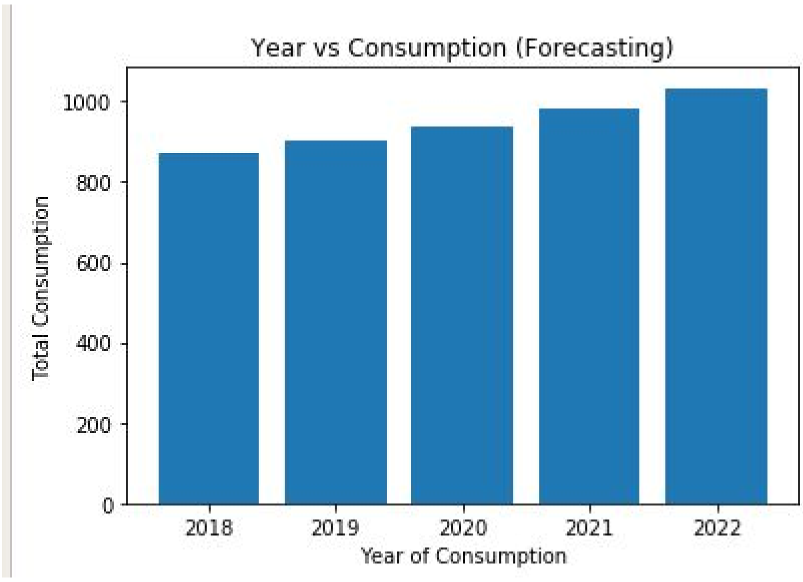
```
[2018, 2019, 2020, 2021, 2022]
[613.062579780817, 638.107122451067, 662.450135320425, 685.9507844746113, 708.468236297369]
```



Crude oil consumption forecasting for next five years in USA:



```
[2018, 2019, 2020, 2021, 2022]
[872.662883579731, 902.3851082921028, 938.6712898612022, 982.0944223999977, 1033.2283418774605]
```



7.2 FUTURE SCOPE

Every project whether large or small has some limitations no matter however diligently developed. In some cases limitations is small while in other cases they may be broad also. The new system has got some limitations. Major areas where modifications can be done are as follows:

1. An intuitive dashboard can be created where user can view see the results of prediction in graphical form
2. We can also make a centralized database where all the data will be stored
3. Apart from console based implementation we can extend our project so that it can be implemented both as web based and app based project.

REFERENCES

1. yearbook.enerdata.net/crude-oil/world-production-statistics.html
2. https://doi.org/10.1016/j.energy.2017.12.042 (Energy, Volume 144, 1 February 2018)
3. https://doi.org/10.1016/j.enpol.2005.08.023 (Energy Policy,Volume 34, Issue 18, December 2006)
4. http://www.statsoft.com/Textbook/Time-Series-Analysis

Industries

Renewable resources