OUTBREAK PREDICTION OF COVID-19 IN INDIA USING MACHINE LEARNING

An Online Summer Internship project *report submitted to the JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD in partial fulfillment of the requirements for the award of the degree of*

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

Vallamkondu Sai Vaishnavi (18071A05B7)

Paripally Preethi (18071A0598)

Moturi Rishitha (18071A0594)

Kolasani Aarti Chowdary (18071A0583)

Under the Guidance Of

Dr. N Sandhya

(Professor, VNRVJIET)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute, NAAC Accredited With 'A++' Grade, NBA Accredited, Approved by AICTE, New Delhi, Affiliated to JNTUH)

August, 2020.

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute**, NAAC Accredited With 'A++' Grade, NBA Accredited, Approved by AICTE, New Delhi, Affiliated to JNTUH)**

CERTIFICATE

This is to certify that Miss Vallamkondu Sai Vaishnavi (18071A05B7), Miss Paripally Preethi (18071A098), Miss Moturi Rishitha (18971A0594), Miss Kolasani Aarti Chowdary (18071A0583) have successfully completed their project work at CSE Department of VNR VJIET, Hyderabad entitled **"OUTBREAK PREDICTION OF COVID-19 IN INDIA USING MACHINE LEARNING"** in partial fulfilment of the requirements for the award of B. Tech degree during the academic year 2020-2021.

This work is carried out under my supervision and has not been submitted to any other University/Institute for award of any degree/diploma.

Dr. N. Sandhya

Project Guide
Professor
CSE Department
VNRVJIET.

Dr. B. V. Kiranmayee

Associate Professor and Head
CSE Department
VNRVJIET.

DECLARATION

This is to certify that the project work entitled **"OUTBREAK PREDICTION OF COVID-19 IN INDIA USING MACHINE LEARNING"** submitted in VNR Vignana Jyothi Institute of Engineering & Technology in partial fulfilment of requirement for the award of Bachelor of Technology in Computer Science and Engineering is a bonafide report of the work carried out by us under the guidance and supervision of Dr. N. Sandhya (Professor), Department of CSE, VNRVJIET. To the best of our knowledge, this report has not been submitted in any form to any university or institution for the award of any degree or diploma.

V. Sai Vaishnavi	**P. Preethi**	**M. Rishitha**	**K. Aarti Chowdary**
(18071A05B7)	(18071A0598)	(18071A0594)	(18071A0583)
III B.Tech-CSE	III B.Tech-CSE	III B.Tech-CSE	III B.Tech-CSE
VNR VJIET	VNR VJIET	VNR VJIET	VNRVJIET

ACKNOWLEDGEMENT

Behind every achievement lies an unfathomable sea of gratitude to those who activated it, without it would ever never have come into existence. To them we lay the words of gratitude imprinting within us.

We are indebted to our venerable principal **Dr. C. D. Naidu** for this unflinching devotion, which lead us to complete this project. The support, encouragement given by him and his motivation lead us to complete this project.

We express our thanks to internal guide **Prof. N. Sandhya** and also Head of the department **Dr. B. V. Kiranmayee** for having provided us a lot of facilities to undertake the project work and guide us to complete the project.

We express our sincere thanks to our faculty of the department of **Computer Science and Engineering** and the remaining members of our college **VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY** who extended their valuable support in helping us to complete the project in time.

Vallamkondu Sai Vaishnavi (18071A05B7)

Paripally Preethi (18071A0598)

Moturi Rishitha (18071A0594)

Kolasani Aarti Chowdary (18071A0583)

ABSTRACT

The unexpected pervading of severe acute respiratory syndrome COVID-19 has been leading into an eminent crisis. It has affected various fields in the economy, for say, public transportation, agricultural, IT industry, manufacturing. Due to the highly complex nature of the COVID-19 outbreak and variation in its behavior from nation-to-nation, Machine Learning (ML) is being suggested as an effective tool to model the outbreak. To know the severity of the pandemic in India, we used some ML algorithms and predicted the future scenario of the nation by enumerating the confirmed cases, deaths and recoveries. Various prediction models are made by using ML algorithms and their results are computed, compared and evaluated. We gathered case data from Jan 1st, 2020 to Jun 28th, 2020 for India to compare different models' proficiency in COVID-19 cases prediction. We assessed the performance of 6 ML models including Linear Regression, Logistic Regression, SVM, Decision Tree, Random Forest, Prophet. Prophet, SVM and Linear Regression had the comparatively negligible prediction error rates for tracking the dynamics of incidents cases in India. The major take-away of this work include accurate analysis of confirmed cases, recovered cases, deaths, prediction of pandemic outbreak for next 20 days. Needless to mention, there are many factors that lead to increase in number of cases in the forthcoming days. It is high time that people should take up responsibility and strive towards the decrease of the virus outbreak by following government norms and rules. It is people of the country and how responsibly they behave, will bring back the good old days. People in the country can reduce their chances of being infected or spreading COVID-19 by taking some simple precautions that are available on the WHO website and staying aware of the latest information on the COVID-19 outbreak.

Key Words:

COVID-19; Model; Prediction; Machine Learning; Logistic Regression; SVM; Prophet; Linear Regression; Random Forest; Decision Tree; Forecasting

INDEX

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION TO COVID-19

The catastrophic outbreak of Severe Acute Respiratory Syndrome - Coronavirus (SARS-CoV-2) also known as COVID-2019 has brought the worldwide threat to the living society. The pandemic has been spreading rapidly everywhere the planet leading to the heavy economic hardships and immense loss to mankind. the primary pandemic attack of SARSCoV-2 was reported in Wuhan (sprawling capital of south china) on 17th November 2019. The diagnosed opening case was on 8th December 2019 and specialists didn't openly admit there was human-to-human transmission until 21st January. World Health Organization (WHO) declared on January 30, 2020 the outbreak as an emergency and pandemic for public health. COVID-19's clinical symptoms are respiratory disease, fatigue, dry cough, tiredness, etc. while 80 percent of patients heal with none care. Older people, and other people with pre-existing medical conditions (such as cardiovascular disorder, obesity, asthma and diabetes) have more chances of becoming severely ill with the virus. The correct way to cease and drop transmission is maintaining social distancing. We have to guard our self and others from infection by washing our hands or using sanitizers and avoid touching face. Besides the numerous known and unknown variables involved in the spread of the virus, the complexity of population-wide behavior in various geopolitical areas and differences in containment strategies had increased model uncertainty to a strikingly large extent. Consequently, standard epidemiological models face new challenges to deliver more reliable results. To induce better of this challenge, many novel models have emerged which bring several assumptions to modeling (e.g., adding social distancing with in the sort of curfews, quarantines, etc.). Forecasting the infection spread can even make things easy with the estimation of hospital resources that maybe required. It can help countries so as to confront to face up the long run and allot their time and money.

While specializing in the Indian plight, the first case of the 2019–20 coronavirus epidemic in India was accounted on 30th January 2020. Specialists propose the count of the disease might be much higher as India's examination rates are among the most diminished on the planet. In India, as of 05:30 GMT+5:30, 27th August 2020, there have been 23,980,044 confirmed cases of COVID-19, including 820,763 deaths, reported to WHO. Collective and focused efforts for containment

and management of COVID-19 by the Government of India along with the States/UTs have led to the number of recovered cases among COVID-19 patients rise to 2,523,771 as of 08:00 IST (GMT+5:30) 27th Aug 2020, reported by MOHFW.

1.2 INTRODUCTION TO MACHINE LEARNING

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies, or analyze the impact of machine learning processes.

1.2.1 Some machine learning methods

ML algorithms are categorized as supervised or unsupervised.

i) Supervised learning: The computer is provided with example inputs that are labeled with their desired outputs. The purpose of this method is for the algorithm to be able to “learn” by comparing its actual output with the “taught” outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on

additional unlabeled data. A common use case of supervised learning is to use historical data to predict statistically likely future events.

ii) Unsupervised learning: The data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods that facilitate unsupervised learning are particularly valuable. The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

1.3 EXISTING SYSTEM

This is contrary to several reports questioning the testing strategies adopted by India quoting a low transmission rate and hence the smaller basic reproduction number. However, India is on high risk to enter into community transmission due to reported violation of quarantine norms by individuals as well as other social and demographic issues. The predictions made using the current epidemiological models in the current work will be invalid if such an event occurs.

1.4 PROPOSED SYSTEM

Six Machine Learning Algorithms are used to predict the outbreak for next 20 days. These models assume all the seed cases to be symptomatic, which may underestimate the actual numbers due to an uncertain number of asymptomatic individuals. Finally, the model is as good as the underlying data. Because of real time change in data daily, the predictions will accordingly change. Hence, the results from this paper should be used only for qualitative understanding and reasonable estimate of the nature of outbreak, but are not advisable for any decision making or policy change.

CHAPTER 2: LITERATURE SURVEY

2.1 PYTHON API's & LIBRARIES REQUIRED

2.1.1 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code.

2.1.2 Sklearn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy.

2.1.3 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation etc

2.1.4 Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labelled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language.

2.1.5 Plotly

Plotly is one of the powerful libraries for data science, machine learning and artificial intelligence-related operations where it helps to create multiple types of interactive visualizations by using Python, R and Java.

2.1.6 Random

Random module contains a number of functions that use random numbers. It can output random numbers, select a random item from a list, and reorder lists randomly. The randomly reordered lists can be output inline, or as various types of ordered and unordered lists.

2.1.7 Math

The Python Math Library provides us access to some common math functions and constants in Python, which we can use throughout our code for more complex mathematical computations. The library is a built-in Python module; therefore, you don't have to do any installation to use it.

2.1.8 Time

There is a popular time module available in Python which provides functions for working with times, and for converting between representations**.**

2.1.9 Warnings

Warning messages are displayed by warn() function defined in ‘warning’ module of Python's standard library. Warning is actually a subclass of Exception in built-in class hierarchy. There are a number of built-in Warning subclasses. User defined subclass can also be defined.

2.2 PYTHON IDE

2.2.1 Introduction to Python IDE

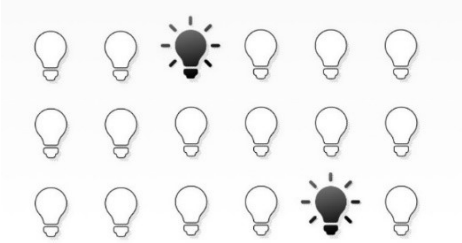
Integrated Development Environment (IDE) for Python has been bundled with the default implementation of the language. Its main features are – Multi window text editor with syntax highlighting, autocompletion, smart indent and others, python shell with syntax

highlighting, integrated debugger with stepping, persistent breakpoints, and call stack visibility.

2.3 MACHINE LEARNING APPLICATIONS IN COVID-19

9 ways machine learning is helping us fight the viral pandemic.

2.3.1 Identifying who is most at risk from COVID-19



Machine learning has proven to be invaluable in predicting risks in many spheres.

With medical risk specifically, machine learning is potentially interesting in three key ways.

- Infection risk: What is the risk of a specific individual or group getting COVID-19?
- Severity risk: What is the risk of a specific individual or group developing severe COVID-19 symptoms or complications that would require hospitalization or intensive care?
- Outcome risk: What is the risk that a specific treatment will be ineffective for a certain individual or group, and how likely are they to die?

Machine learning can potentially help predict all three risks. Although it's still too early for much COVID-19-specific machine learning research to have been conducted and published, early experiments are promising. Furthermore, we can look at how machine learning is used in related areas and imagine how it could help with risk prediction for COVID-19.

2.3.2 Screening patients and diagnosing COVID-19

When a new pandemic hits, diagnosing individuals is challenging. Testing on a large scale is difficult and tests are likely to be expensive, especially in the beginning. Anyone who has any

symptoms of COVID-19 is likely to be very concerned that they have contracted the disease, even if the same symptoms are indicative of many other, potentially milder diseases too.

Rather than taking medical samples from each patient and waiting for slow, expensive lab reports to come back, a simpler, faster, and cheaper test (even if it's less accurate) would be useful in gathering data on a larger scale. This data could be used for further research, as well as for screening and triaging patients.

When it comes to using machine learning to help diagnose COVID-19, promising research areas include:

- Using face scans to identify symptoms, such as whether or not the patient has a fever,
- Using wearable technology such as smart watches to look for tell-tale patterns in a patient's resting heart rate,
- Using machine learning-powered chatbots to screen patients based on self-reported symptoms.

2.3.3 Speeding up drug development

Regarding to a new pandemic, it's critical to come up with a vaccine, a reliable diagnostic method, and a drug for treatment – fast. Current methods involve a lot of trial and error, which takes time. It can take months to isolate even one viable vaccine candidate.

Machine learning can speed up this process significantly without sacrificing quality control. When researchers were trying to find small molecule inhibitors of the Ebola virus, they discovered that training Bayesian ML models with viral pseudo type entry assay and the Ebola virus replication assay data helped speed up the scoring process. (Scoring involves assigning each molecule a value based on how likely it is to help.) This accelerated process quickly identified three potential molecules for testing.

Similarly, researchers working on H7N9 discovered that ML-assisted virtual screening and scoring led to substantial improvements in the accuracy of the scores. Using the random forest algorithm (a classification algorithm made up of lots of decision trees) provided the best results with H7N9.

At times like the COVID-19 pandemic, where a virus is spreading rapidly, getting more accurate scores faster is critical to speeding up drug development.

2.3.4 Identifying effective existing drugs

Companies spend a lot of time and money getting new drugs approved. They need to be as sure as they possibly can that these drugs won't have unexpected, harmful side effects. This process protects us, but it also slows us down during a pandemic – just when we need a faster response. One alternative is to repurpose drugs that have already been tested and used to treat other diseases. But there are thousands of drug candidates, and we don't have time to test them all – so how do we find the right one?

Machine learning can help us prioritize drug candidates much faster by automatically:

1. Building knowledge graphs and
2. Predicting interactions between drugs and viral proteins.

2.3.5 Predicting the spread of infectious disease using social networks

Amid a pandemic, when we're trying to develop strategies to actively work against it, we first need to know where we are. We need to answer questions like "How many people are infected?" and "Where are these people?" Unfortunately, pandemics – especially those caused by viruses – are difficult and expensive to keep track of. Usually the government answers these questions, together with the health system. For example, every day (or week) the responsible agency counts and publicizes the number of new patients diagnosed with the disease. But one of the problems here is that there might be a big gap (in time and space) between contracting the disease, developing the first symptoms, and testing positive. Luckily, we live in a digital world. A farmer who is starting to develop symptoms might live in a small town with no nearby hospitals capable of performing the test. But this same farmer might still be able to access social networks and immediately leave hints about his health and the spread of the disease – hints that only a machine learning model can learn to process at scale.

By interpreting the content of public interactions on social media, a machine learning model assesses the likelihood of novel virus contamination. The model might not be able to classify people on an individual level, but it can use all of this data to estimate the spread of the

pandemic in real time and to forecast the spread in the upcoming weeks. The value of this information in decision-making processes in the midst of a rapidly evolving pandemic cannot be overstated.

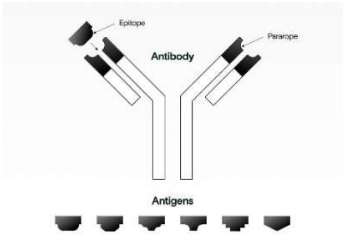
2.3.6 Understanding viruses through proteins

To understand a virus such as COVID-19 is to understand its proteins – whether and how we get sick depends entirely on how these proteins interact with our bodies. But interpreting them is no easy task. The following use cases provide examples of how machine learning can help improve our understanding of viruses by analyzing their proteins.

2.3.7 Figuring out how to attack the virus

Epitopes are clusters of amino acids found on the outside of a virus. Antibodies bind to epitopes, which is how our immune system recognizes and eliminates the virus. So, finding and classifying epitopes is essential in determining which part of a molecule to target when we develop vaccines. Compared to traditional vaccines, which contain inactivated pathogens, epitope-based vaccines are safer – they prevent disease without the risk of potentially deadly side effects. Locating the correct epitope can be a time-consuming, expensive process. With a new pandemic, such as COVID-19, locating epitopes faster speeds up the process of developing effective vaccines.

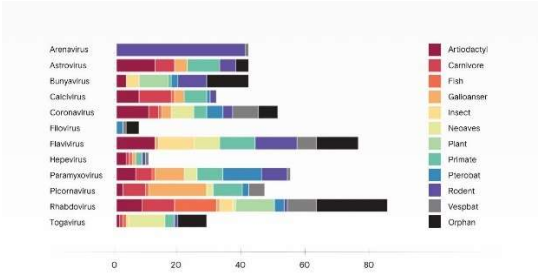
This is where machine learning can help. Support vector machines (SVM), hidden Markov Models, and artificial neural networks (specifically deep learning) have all proven to be faster and more accurate at identifying epitopes than human researchers are.



2.3.8 Identifying hosts in the natural world

A zoonotic pandemic – like the one we are experiencing with the novel coronavirus – is a pandemic caused by an infectious disease that originates in a different species (such as bats) and

spreads to humans. Viruses such as Ebola, HIV, or COVID-19 can survive unnoticed in the natural world for a long time, waiting for the next mutation and the next opportunity to infect us. They hide in animals – called reservoir hosts – that are unaffected by the illness. Knowing who these reservoir hosts are is vital in fighting a pandemic – once we've found them, we can develop strategies to control the spread of the disease and prevent more outbreaks from happening.



The classical approach to finding reservoir hosts can take years of research, and there are still many orphan viruses that haven't been matched to an animal host. Thanks to huge advances in technology, Whole-Genome Sequencing (WGS, the process of determining an organism's complete DNA sequence) has become cheap and fast. Research has shown that machine learning models can use genome sequencing data together with expert knowledge to pinpoint the species that most likely acted as hosts for the disease. By looking at a small subset of species, we can dramatically speed up the process of finding these pathogens in the wild.

2.3.9 Predicting the risk of new pandemics

Accurately predicting whether a strain of influenza is going to make a zoonotic leap (jumping from one species to another) can help doctors and medical professionals anticipate potential pandemics and prepare accordingly. As one example, Influenza A exists primarily in the avian population, but it has the potential to jump to human hosts. Researchers working on Influenza A isolated 67,940 protein sequences from a database. They filtered these sequences so that the dataset included only those influenza strains with complete sequences of 11 influenza proteins.

With machine learning the researchers were then able to identify potentially zoonotic strains of influenza with high levels of accuracy. More work needs to be done to establish prediction models for direct transmission, but knowing which strains of influenza are likely to make a leap is an important first step in preparing for the next pandemic.

CHAPTER 3: ALGORITHM DISCRIPTION

3.1 LINEAR REGRESSION

3.1.1 Introduction to Linear Regression

According to 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). 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.

$y = a + b1x + b2x^2 +....+ bnx^n$

3.1.2 Advantages of Linear Regression:

i) Linear Regression performs well when the dataset is linearly separable. We can use it to find the nature of the relationship among the variables.

ii) Linear Regression is easier to implement, interpret and very efficient to train.

iii) Linear Regression is prone to over-fitting but it can be easily avoided using some dimensionality reduction techniques, regularization (L1 and L2) techniques and cross-validation.

3.1.3 Disadvantages of Linear Regression:

i) Linear Regression Is Limited to Linear Relationships

ii) Linear Regression Only Looks at the Mean of the Dependent Variable

iii) Linear Regression Is Sensitive to Outliers

iv) Data Must Be Independent

3.1.4 Applications of Linear Regression:

i)Studying engine performance from test data in automobiles.

ii) Least squares regression is used to model causal relationships between parameters in biological systems.

iii) OLS regression can be used in weather data analysis

iv) Linear regression can be used in market research studies and customer survey results analysis.

v) Linear regression is used in observational astronomy commonly enough. A number of statistical tools and methods are used in astronomical data analysis, and there are entire libraries in languages like Python meant to do data analysis in astrophysics.

3.2 LOGISTIC REGRESSION

3.2.1 Introduction to Logistic Regression:

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values). A linear regression is not appropriate for predicting the value of a binary variable for two reasons: A linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1)

Since the dichotomous experiments can only have one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line.

On the other hand, a logistic regression produces a logistic curve, which is limited to values between 0 and 1. Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the "odds" of the target variable, rather than the probability. Moreover the predictors do not have to be normally distributed or have equal variance in each group.

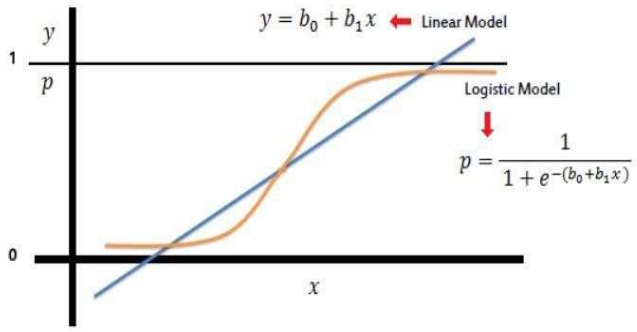


Fig 3.1: Logistic regression

Logistic regression has the constant (b_0) moving the curve left and right and the slope (b_1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.

$$\frac{p}{1-p} = \exp(b_0 + b_1 x)$$

Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b_1) is the amount the logit (log-odds) changes with a one unit change in x.

$$ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

As mentioned before, logistic regression can handle any number of numerical and/or categorical variables.

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}}$$

3.2.2 Advantages of Logistic Regression:

i) Logistic Regression performs well when the dataset is linearly separable.

ii) Logistic regression is less prone to over-fitting but it can overfit in high dimensional datasets. You should consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.

iii) Logistic Regression not only gives a measure of how relevant a predictor (coefficient size) is, but also its direction of association (positive or negative).

iv) Logistic regression is easier to implement, interpret and very efficient to train.

3.2.3 Disadvantages of Logistic Regression:

i) Main limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data is rarely linearly separable. Most of the time data would be a jumbled mess.

ii) If the number of observations are lesser than the number of features, Logistic Regression should not be used, otherwise it may lead to overfit.

iii) Logistic Regression can only be used to predict discrete functions. Therefore, the dependent variable of Logistic Regression is restricted to the discrete number set. This restriction itself is problematic, as it is prohibitive to the prediction of continuous data.

3.2.4 Applications of Logistic Regression:

i)Image Segmentation and Categorization.

ii) Geographic Image Processing.

iii) Handwriting recognition.

iv) It is one of the best tools used by statisticians, researchers and data scientists in predictive analytics.

3.3 SUPPORT VECTOR MACHINE (SVM)

3.3.1 Introduction to SVM:

Support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis in Machine Learning.

SVM regression being a non-parametric technique depends on a set of mathematical functions. The set of functions called kernel transforms the data inputs into the desired form. SVM solves the regression problems using a linear function, so while dealing with problems of non-linear regression, it maps the input vector (x) to n -dimensional space called a feature space (z). This mapping is done by non-linear mapping techniques after that linear regression is applied to space. Putting the concept in ML context with a multivariate training dataset (xn) with N number of observations with yn as a set of observed responses. The linear function can be depicted as:

$f(x)=x'\beta+b$

The objective is to make it as flat as possible thus to find the value of f(x) with ($\beta'\beta$) as minimal norm values. So, the problem fits in minimization function as:

$J(\beta)=(½)\beta'\beta$

with a special condition of the values of all residuals not more than ε, as in the following equation:

$\forall n:|yn-(x'n\beta+b)|\leq\varepsilon$

3.3.2 Advantages of SVM:

i) SVM works relatively well when there is clear margin of separation between classes.

ii) SVM is more effective in high dimensional spaces.

iii) SVM is effective in cases where number of dimensions is greater than the number of samples.

iv) SVM is relatively memory efficient.

3.3.3 Disadvantages of SVM:

i) SVM algorithm is not suitable for large data sets.

ii) SVM does not perform very well, when the data set has more noise i.e. target classes are overlapping.

iii) In cases where number of features for each data point exceeds the number of training data sample, the SVM will underperform.

iv) As the support vector classifier works by putting data points, above and below the classifying hyper plane there is no probabilistic explanation for the classification.

3.3.4 Applications of SVM:

i) Face Detection

ii) Image Classification

iii) Text and Hypertext Categorization

iv) Bioinformatics

v) Geo and Environmental Science

3.4 DECISION TREE

3.4.1 Introduction to Decision Tree:

Generally, Decision tree analysis is a predictive modelling tool that can be applied across many areas. Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. Decisions trees are the most powerful algorithms that falls under the category of supervised algorithms.

They can be used for both classification and regression tasks. The two main entities of a tree are decision nodes, where the data is split and leaves, where we got outcome. The example of a binary

tree for predicting whether a person is fit or unfit providing various information like age, eating habits and exercise habits, is given below –

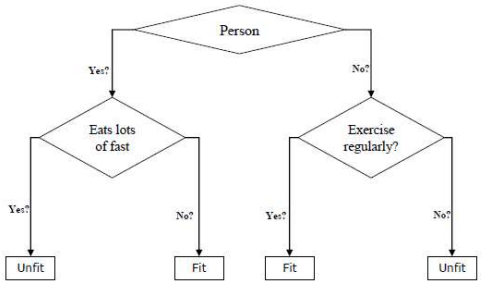


Fig 3.2 Decision Tree

In the above decision tree, the question are decision nodes and final outcomes are leaves. We have the following two types of decision trees.

- **Classification decision trees** − In this kind of decision trees, the decision variable is categorical. The above decision tree is an example of classification decision tree.
- **Regression decision trees** − In this kind of decision trees, the decision variable is continuous.

3.4.2 Advantages of Decision Tree:

i) Decision Tree can handle both continuous and categorical variables.

ii) Decision Tree can be used for both classification and regression problems.

iii) Decision Tree can automatically handle missing values.

iv) Simple and easy to understand

v) Decision Tree is usually robust to outliers and can handle them automatically.

3.4.3 Disadvantages of Decision Tree:

i) High variance

ii) Unstable

iii) Affected by noise

iv) Not suitable for large datasets

v) Overfitting

3.4.4 Applications of Decision Tree:

i)Assessing prospective growth opportunities.

ii) Using demographic data to find prospective clients.

iii) Serving as a support tool in several fields.

3.5 RANDOM FOREST

3.5.1 Introduction to Random Forest:

Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

Working of Random Forest Algorithm

We can understand the working of Random Forest algorithm with the help of following steps −

- **Step 1** − First, start with the selection of random samples from a given dataset.
- **Step 2** − Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
- **Step 3** − In this step, voting will be performed for every predicted result.
- **Step 4** − At last, select the most voted prediction result as the final prediction result.

The following diagram will illustrate its working –

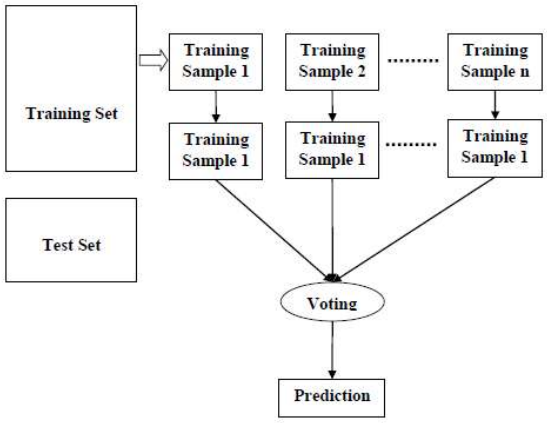


Fig 3.3 Random Forest

3.5.2 Advantages of Random Forest:

i) It is one of the most accurate learning algorithms available.

ii)It runs efficiently on large databases.

iii) It can handle thousands of input variables without variable deletion.

iv) It gives estimates of what variables are important in the classification.

3.5.3 Disadvantages of Random Forest:

i) Random forests have been shown to fit over certain noisy classification or regression problems.

ii) For data with different values, attributes with more values will have a greater impact on random forests, so the attribute weights generated by random forests on such data are not credible

3.5.4 Applications of Random Forest:

i) Classification od discrete values

ii) Regression of continuous values

iii) Unsupervised learning clustering

iv) Abnormal point detection

3.6 PROPHET

3.6.1 Introduction to Prophet:

Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

We use a decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

g(t): piecewise linear or logistic growth curve for modelling non-periodic changes in time series

s(t): periodic changes (e.g. weekly/yearly seasonality)

h(t): effects of holidays (user provided) with irregular schedules

ε_t: error term accounts for any unusual changes not accommodated by the model

Using time as a regressor, Prophet is trying to fit several linear and nonlinear functions of time as components. Modeling seasonality as an additive component is the same approach taken

by exponential smoothing in Holt-Winters technique . We are, in effect, framing the forecasting problem as a curve-fitting exercise rather than looking explicitly at the time-based dependence of each observation within a time series.

3.6.2 Advantages of Prophet:

i) Open Source.

ii) Accurate and fast.

iii) Allows for a large number of people to make forecasts, possibly without training in time series methods.

iv) Tunable parameters.

v) Available for both Python and R.

vi) Next, we will see if adding holiday indicators will help the accuracy of the model. Prophet comes with a Holiday Effects parameter that can be provided to the model prior to training.

3.6.3 Disadvantages of Prophet:

i) If you need to predict hundreds or thousands of targets simultaneously, Prophet will compute slowly.

ii) If you have more meaningful features than just seasonality or special events, Prophet won't help.

iii)Prophet was never meant to be an all-purpose predictive algorithm, and its downsides reflect its specialization.

3.6.4 Applications of Prophet:

i)Contain an extended time period (months or years) of detailed historical observations (hourly, daily, or weekly)

ii) Have multiple strong seasonality’s.

iii) Include previously known important, but irregular, events.

iv) Have missing data points or large outliers.

CHAPTER 4: SYSTEM-ANALYSIS

4.1 SYSTEM REQUIREMENTS

4.1.1 Software Requirements

Google Colab:

Colaboratory is a Google research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google

Fig 4.1 Google Colab

4.1.2 Hardware Requirements

RAM: 4GB and Higher

Processor: Intel i3 and above

Hard disk: 10GB Minimum

4.2 SYSTEM ARCHITECTURE

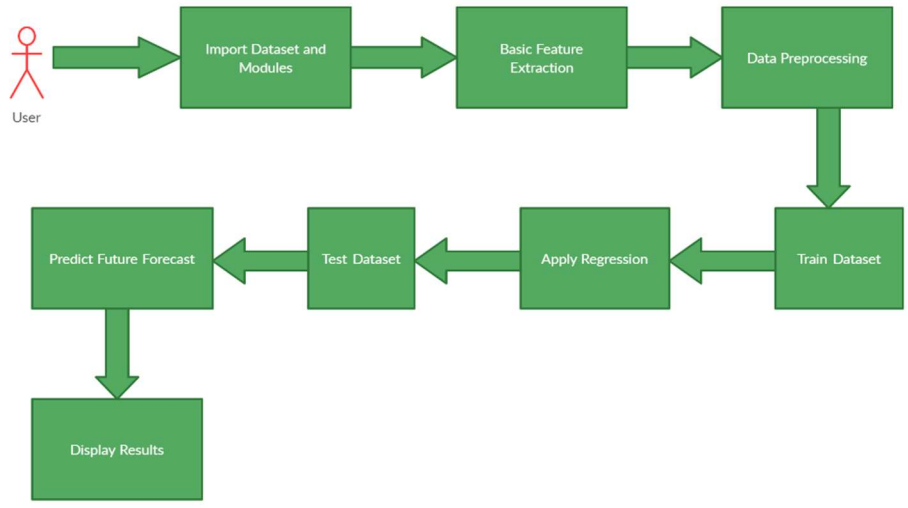


Fig 4.2 System Architecture

a) Importing Dataset:

User needs to Load the dataset (global data of covid-19 cases) which contains the required information.

b) Basic Feature Extraction:

The required 3 data frames of India are extracted to perform predictions from the global dataset into Confirmed cases, Death Cases and Recovered cases.

c)Data preprocessing:

Text preprocessing include the following

Filling the missing values,

Handling with NAN values,

Converting the date form into Numerical form etc..

d) Train the dataset:

Dataset is splitted into trained and test dataset and data is trained using regression technique.

e) Apply Regression:

Regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates', or 'features').

We have predicted the Outbreak using 6 models such as Linear Regression, Logistic Regression, SVM, Decision Tree, Random Forest and Prophet.

f) Test dataset:

The data is tested over trained model it predicts number of confirmed cases, deaths and recovered caes during the respective dates.

g) Predict Future Outbreak:

Number of confirmed cases, deaths, recovered cases for next 20 days are predicted.

h) Display results:

Plot Graphs and calculated the RMSE values. This shows the efficiency of algorithm

CHAPTER 5: SOFTWARE DESIGN

5.1 UML Diagrams

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation).

There are five different views that the UML aims to visualize through different modeling diagrams. These five views are:

a. User Model View:

This contains the diagrams in which the user's part of interaction with the software is defined. No internal working of the software is defined in this model. The diagrams contained in this view are:

- Use case Diagram

b. Structural Model View

Only the structure of the model is explained in the structural model view. This gives an estimate of what the software consists of. However, internal working is still not defined in this model. The diagram that this view includes are:

- Class Diagrams
- Object Diagrams

c. Behavioral Model View

The behavioral view contains the diagrams which explain the behavior of the software. These diagrams are:

- Sequence Diagram

- Collaboration Diagram
- State chart Diagram
- Activity Diagram

d. Implementation Model View

The implementation view consists of the diagrams which represent the implementation part of the software. This view is related to the developer's views. This is the only view in which the internal workflow of the software is defined. The diagram that this view contains is as follows:

- Component Diagram

e. Environmental Model View

The environmental view contains the diagram which explains the after deployment behavior of the software model. This diagram usually explains the user interactions and software effects on the system. The diagrams that the environmental model contain are:

- Deployment diagram

These are the physically replaceable components of the system. They are modelled using component diagrams.

The two most broad categories that encompass all other types are **Behavioral** UML diagram and **Structural** UML diagram. As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components. The different types are broken down as follows:

Behavioral UML Diagram

- Activity Diagram
- Use Case Diagram
- Interaction Overview Diagram
- Timing Diagram
- State Machine Diagram
- Communication Diagram

- Sequence Diagram

Structural UML Diagram

- Class Diagram
- Object Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram
- Profile Diagram

5.2 TYPES OF UML DIAGRAMS

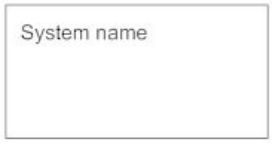
5.2.1 Use Case Diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

Basic Use Case Diagram Symbols and Notations

System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.



System

Use Case

Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.

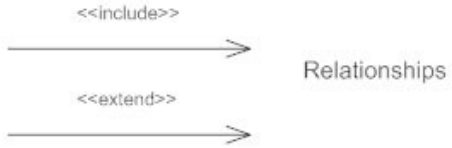
Use case

Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.

Relationships

Demonstrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



The following represents the use case diagram of the proposed system:

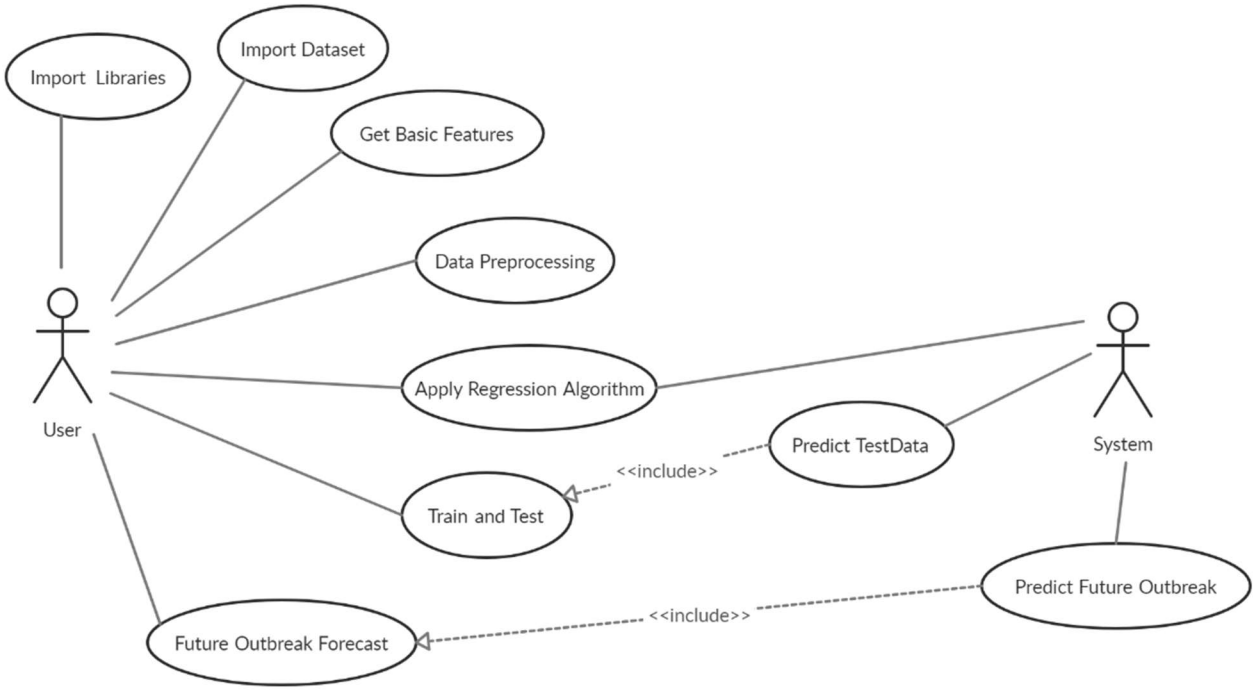


Fig 5.1: Use Case Diagram for Developed Model

5.2.2 Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top-level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram −

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

1.**Scopes:** The UML diagrams have two different types of scopes for class members:

i. instance members scope
ii. classifier members scope

2. **Classifier members** are "static" members of a class in many programming languages. The scope is the class itself.

i. Static attributes are common to all other objects that invoke the class.

ii. Static methods are not instantiated.

3.**Instance members** are nothing but the members that are local to an object.

i. The main purpose of instance members is to allow the objects to store their states.
ii. Declarations outside the methods are usually known as instance members.

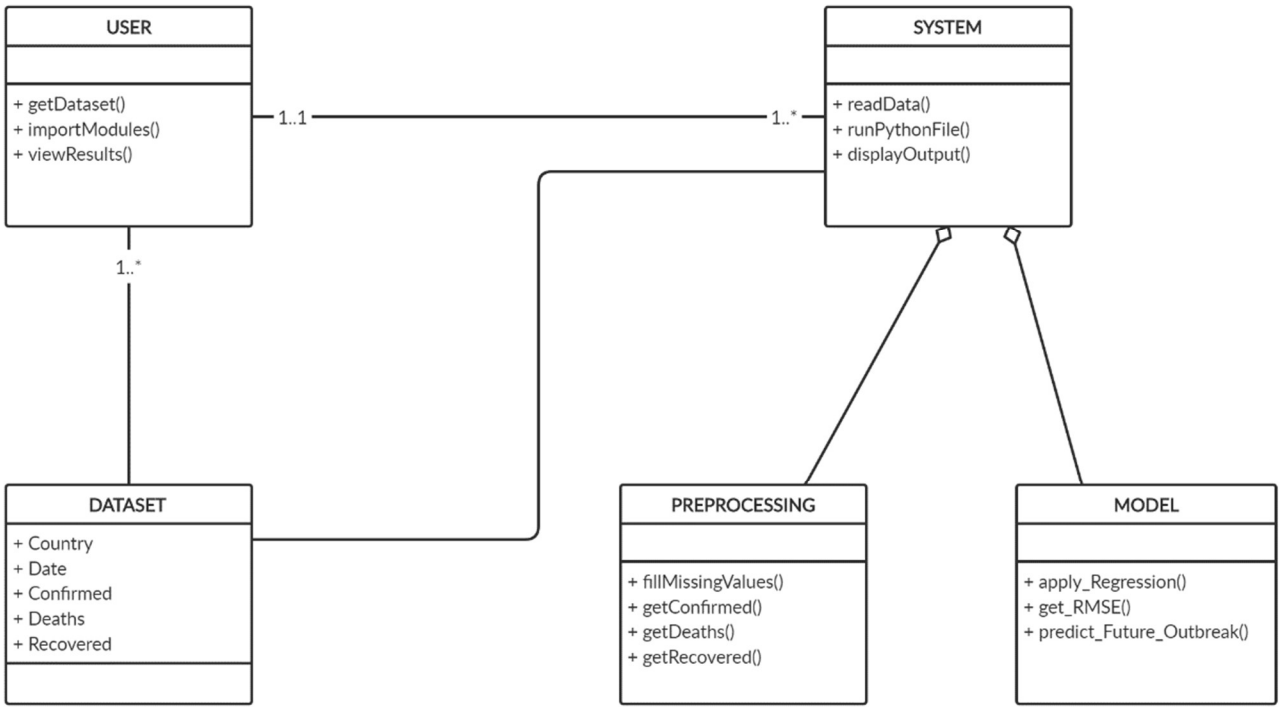


Fig 5.2: Class Diagram for Developed Model

5.2.3 Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

Basic symbols and components

To understand what a sequence diagram is, you should be familiar with its symbols and components. Sequence diagrams are made up of the following icons and elements:

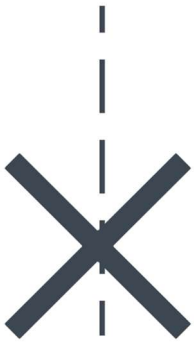
Symbol	Name	Description
	Object symbol	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape.
	Activation box	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
	Actor symbol	Shows entities that interact with or are external to the system.
	Package symbol	Used in UML 2.0 notation to contain interactive elements of the diagram. Also known as a frame, this rectangular shape has a small inner rectangle for labeling the diagram.

Symbol	Name	Description
	Lifeline symbol	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol.

Common message symbols

Use the following arrows and message symbols to show how information is transmitted between objects. These symbols may reflect the start and execution of an operation or the sending and reception of a signal.

Symbol	Name	Description
	Synchronous message symbol	Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues. The diagram should show both the call and the reply.
	Asynchronous message symbol	Represented by a solid line with a lined arrowhead. Asynchronous messages don't require a response before the sender continues. Only the call should be included in the diagram.
	Asynchronous return message symbol	Represented by a dashed line with a lined arrowhead.
	Asynchronous create message symbol	Represented by a dashed line with a lined arrowhead. This message creates a new object.

Symbol	Name	Description
	Reply message symbol	Represented by a dashed line with a lined arrowhead, these messages are replies to calls.
	Delete message symbol	Represented by a solid line with a solid arrowhead, followed by an X. This message destroys an object.

I). Common Properties:

An arrangement graph is much the same as unique sort of diagram and offers some indistinguishable properties from other diagrams. In any case, it varies from every single other diagram in its content.

II). **Contents**

Objects are normally named or unknown instances of class, however may likewise speak to occurrences of different things, for example components, collaboration and nodes. Graphically, object is represented as a rectangle by underlying its name.

III). Links

A link is a semantic association among objects i.e., an object of an affiliation is called as a connection. It is represented as a line.

IV). Messages

A message is a determination of a correspondence between objects that passes on the data with the desire that the action will follow.

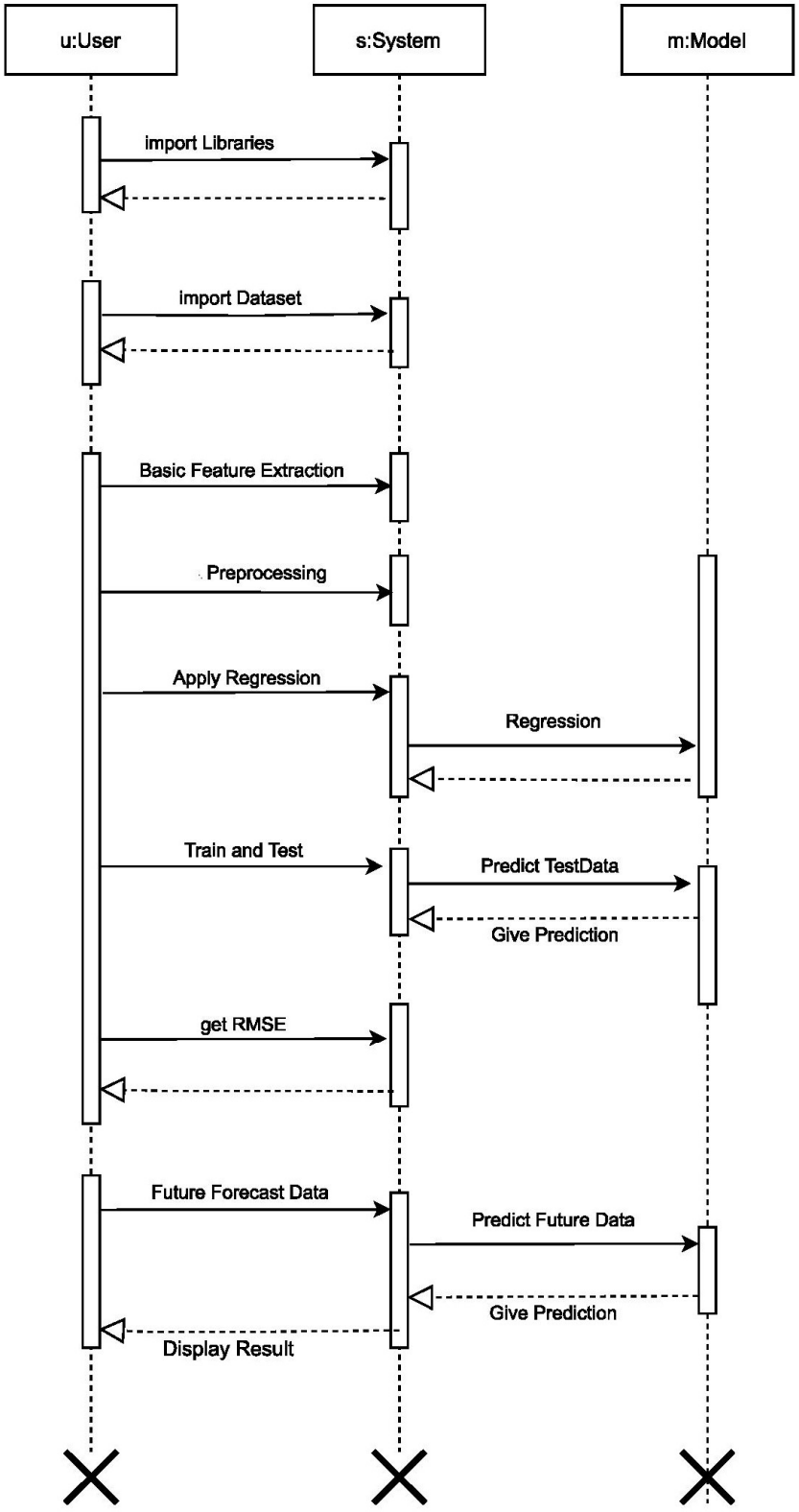


Fig 5.3 Sequence Diagram for Developed Model

5.2.4 Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

To Draw an Activity Diagram

Activity diagrams are mainly used as a flowchart that consists of activities performed by the system. Activity diagrams are not exactly flowcharts as they have some additional capabilities. These additional capabilities include branching, parallel flow, swim lane, etc.

Before drawing an activity diagram, we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities, we need to understand how they are associated with constraints and conditions.

Before drawing an activity diagram, we should identify the following elements −

- Activities
- Association
- Conditions
- Constraints

Once the above-mentioned parameters are identified, we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

Following is an example of an activity diagram for order management system. In the diagram, four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code.

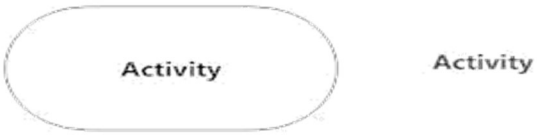
The activity diagram is made to understand the flow of activities and is mainly used by the business users

Following diagram is drawn with the four main activities −

- Send order by the customer
- Receipt of the order
- Confirm the order
- Dispatch the order

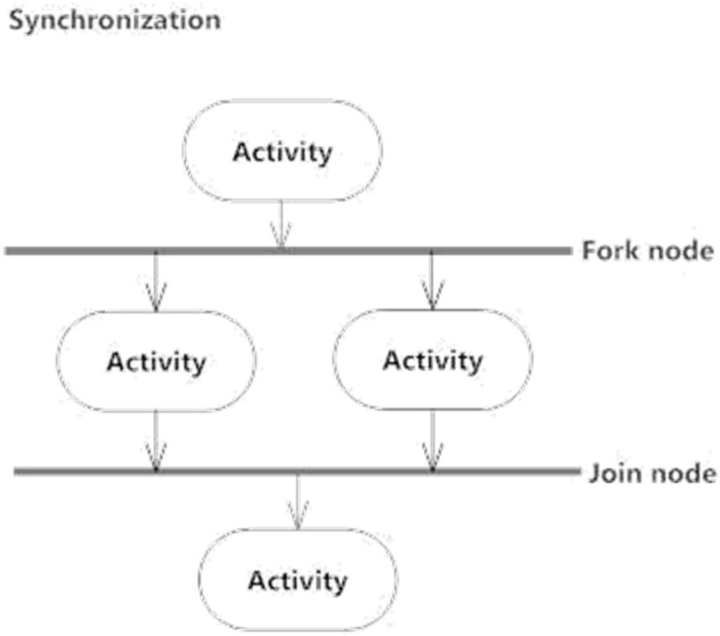
Activity diagrams are constructed using the following:

1. Actions are represented using rounded rectangles;



2. decisions are represented using diamonds;

3. concurrent activities bars are represented using the start (split) or end (join) ;



4. Time event is represented as

5. final state is represented using encircled black circle.

The basic purpose of an activity diagram is same as that of other UML diagrams. The dynamic behavior of the system is viewed by the activity diagram. They are used to construct a system using the backward and forward engineering mechanisms.

The purpose of an activity diagram is as follows:

1) For drawing the flow (i.e. activity) in a system.
2) For showing the flow of sequence from one activity to another activity.
3) For showing the concurrent and parallel flow of actions in the system.

Uses of Activity Diagrams?

The basic usage of activity diagram is similar to other four UML diagrams. The specific usage is to model the control flow from one activity to another. This control flow does not include messages. Activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes the flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues, or any other system.

Activity diagram can be used for −

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

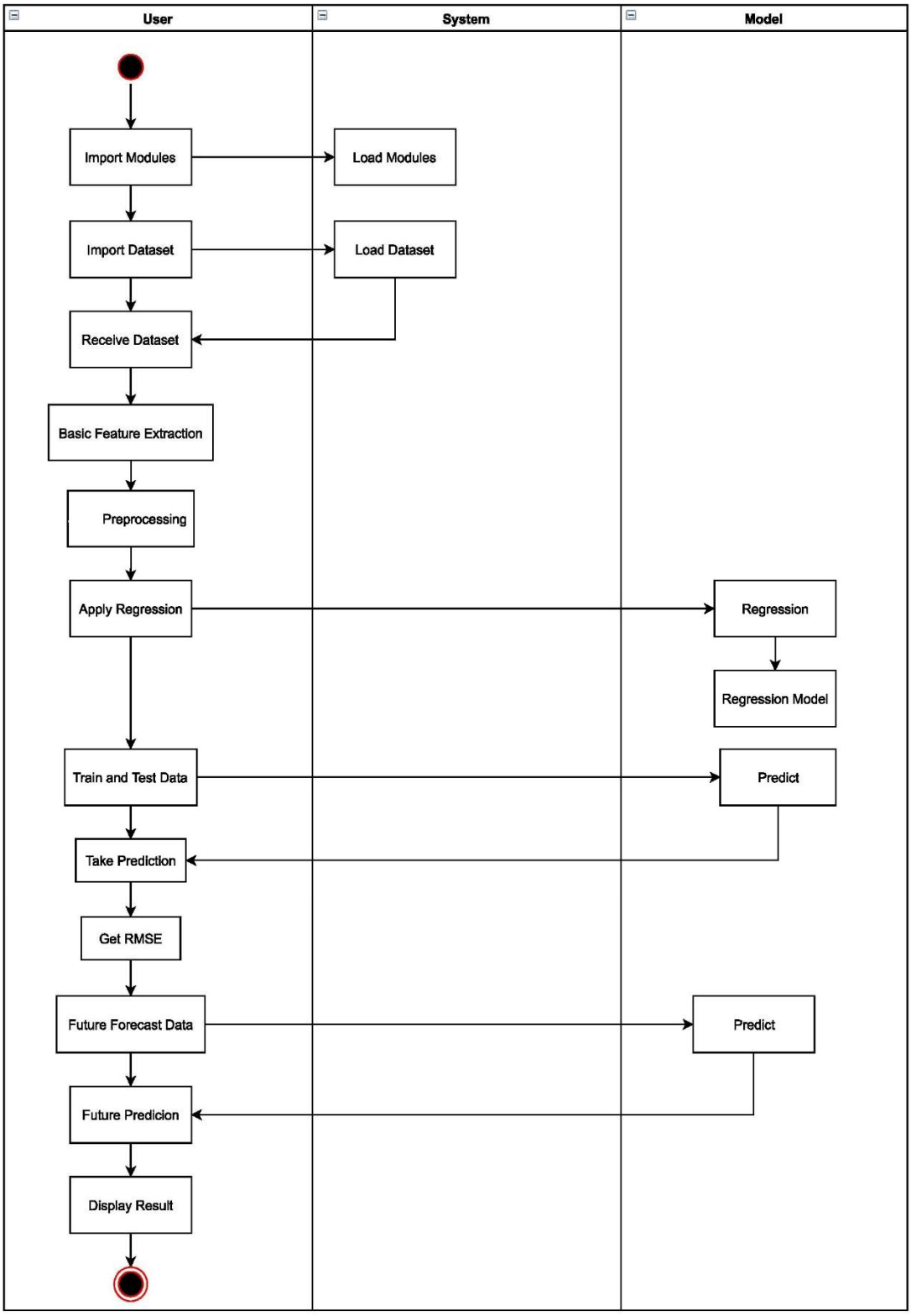


Fig 5.4 Activity Diagram for Developed Model

CHAPTER 6: IMPLEMENTATION

6.1 IMPORTING LIBRARIES:

Importing the required modules and libraries to split the dataset, to apply different classification techniques and displaying the result.

```
# importing the required libraries
import pandas as pd
import numpy as np
# Visualisation libraries
import matplotlib.pyplot as plt
import random
import math
import datetime
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from fbprophet import Prophet
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVR
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error
plt.style.use('fivethirtyeight')
%matplotlib inline
# Manipulating the default plot size
plt.rcParams['figure.figsize'] = 10, 12

# Disable warnings
import warnings
warnings.filterwarnings('ignore')
```

```
[ ] #This cell's code is required when you are working with plotly on colab
    import plotly
    plotly.io.renderers.default = 'colab'
```

6.2 DATA COLLECTION:

The dataset is collected from Kaggle regarding expansion of Covid-19 from the date of 1st January 2020 till 28th June 2020 globally (day-wise).

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
0	NaN	Afghanistan	33.939110	67.709953	1/22/2020	0	0	0	0	Eastern Mediterranean
1	NaN	Albania	41.153300	20.168300	1/22/2020	0	0	0	0	Europe
2	NaN	Algeria	28.033900	1.659600	1/22/2020	0	0	0	0	Africa
3	NaN	Andorra	42.506300	1.521800	1/22/2020	0	0	0	0	Europe
4	NaN	Angola	-11.202700	17.873900	1/22/2020	0	0	0	0	Africa
...	...	...	...	...	...	...	...	...	...	...
41494	NaN	Sao Tome and Principe	0.186400	6.613100	6/28/2020	713	13	219	481	Africa
41495	NaN	Yemen	15.552727	48.516388	6/28/2020	1118	302	430	386	Eastern Mediterranean
41496	NaN	Comoros	-11.645500	43.333300	6/28/2020	272	7	161	104	Africa
41497	NaN	Tajikistan	38.861000	71.276100	6/28/2020	5849	52	4448	1349	Europe
41498	NaN	Lesotho	-29.610000	28.233600	6/28/2020	27	0	4	23	Africa

41499 rows × 10 columns

6.3 DATA PREPROCESSING:

Data Cleansing: The dataset transformed in the following way

1.Firstly narrowed down the data from global scale to India as our subject interest is on pandemic outbreak in India.

2.To analyze the outbreak, date wise data is used and so date is changed to yyyy-mm-dd format.

3.Removed the other columns other than Country/Region, Date, Confirmed, Deaths, Active. Changed the columns Country/Region to Country.

	Country	Date	Confirmed	Deaths	Recovered	Active
0	India	2020-01-01	0	0	0	0
1	India	2020-01-02	0	0	0	0
2	India	2020-01-03	0	0	0	0
3	India	2020-01-04	0	0	0	0
4	India	2020-01-05	0	0	0	0
...	...	...	...	...	...	...
175	India	2020-06-24	473105	14894	271697	186514
176	India	2020-06-25	490401	15301	285637	189463
177	India	2020-06-26	508953	15685	295881	197387
178	India	2020-06-27	528859	16095	309713	203051
179	India	2020-06-28	548318	16475	321723	210120

180 rows × 6 columns

Dividing dataset into three parts:

```
confirmed = df.groupby('Date').sum()['Confirmed'].reset_index()
deaths = df.groupby('Date').sum()['Deaths'].reset_index()
recovered = df.groupby('Date').sum()['Recovered'].reset_index()
```

Confirmed:

	Date	Confirmed
0	2020-01-01	0
1	2020-01-02	0
2	2020-01-03	0
3	2020-01-04	0
4	2020-01-05	0
...	...	...
174	2020-06-24	473105
175	2020-06-25	490401
176	2020-06-26	508953
177	2020-06-27	528859
178	2020-06-28	548318

179 rows × 2 columns

Recovered:

	Date	Recovered
0	2020-01-01	0
1	2020-01-02	0
2	2020-01-03	0
3	2020-01-04	0
4	2020-01-05	0
...	...	...
174	2020-06-24	271697
175	2020-06-25	285637
176	2020-06-26	295881
177	2020-06-27	309713
178	2020-06-28	321723

179 rows × 2 columns

Deaths:

	Date	Deaths
0	2020-01-01	0
1	2020-01-02	0
2	2020-01-03	0
3	2020-01-04	0
4	2020-01-05	0
...	...	...
174	2020-06-24	14894
175	2020-06-25	15301
176	2020-06-26	15685
177	2020-06-27	16095
178	2020-06-28	16475

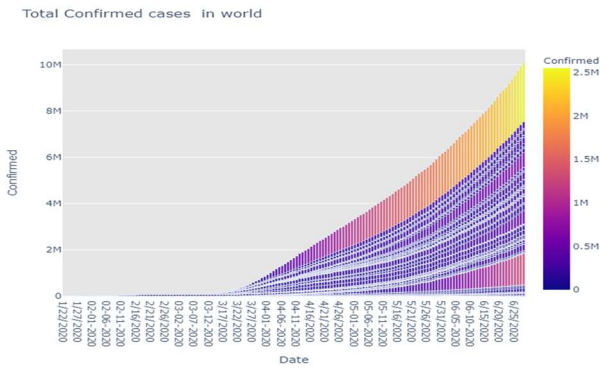
179 rows × 2 columns

6.4 DATA VISUALIZATION:

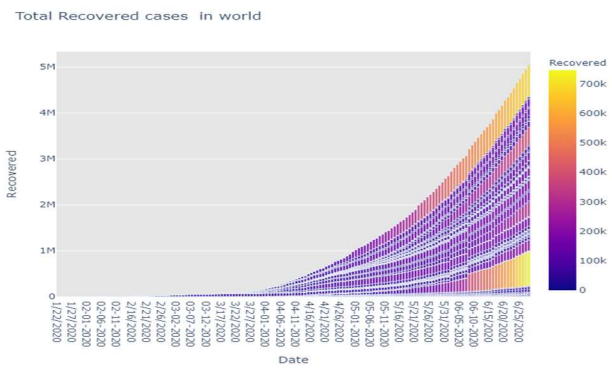
Global Dataset:

Visualized the global dataset: Death, Recovered and Confirmed cases per day

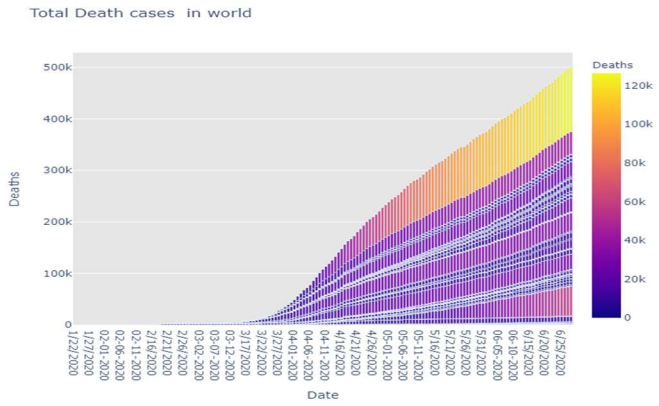
Confirmed:



Recovered:



Deaths:



Visualized the data by taking the values of the number of confirmed, recovered and deaths on Y-axis; count of days from 1st Jan to 28th June

```
Y_confirmed=list(confirmed.iloc[:,-1].values)
Y_deaths=list(deaths.iloc[:,-1].values)
Y_recovered=list(recovered.iloc[:,-1].values)
```

```
# Y_confirmed
X=np.array([i for i in range(len(dates))]).reshape(-1,1)
Y_confirmed=np.array(Y_confirmed).reshape(-1,1)
Y_deaths=np.array(Y_deaths).reshape(-1,1)
Y_recovered=np.array(Y_recovered).reshape(-1,1)
```

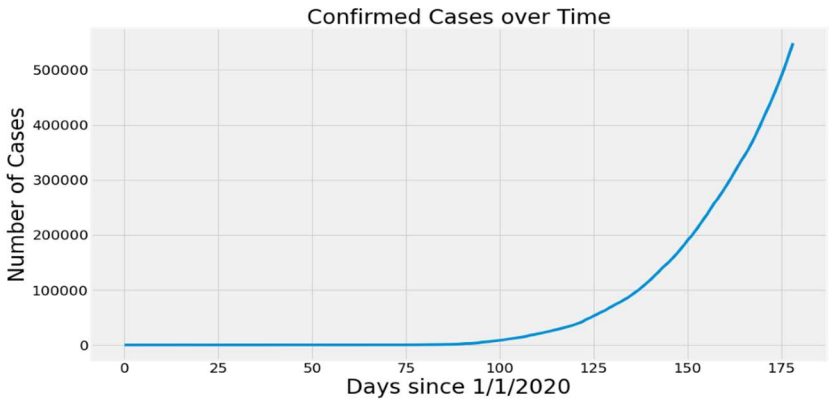
```
[ ] days_in_future=20
    future_forecast=np.array([i for i in range(len(dates)+days_in_future)]).reshape(-1,1)
    adjusted_dates=future_forecast[:-20]
    # future_forecast
```

```
[ ] # adjusted_dates
    adjusted_dates=adjusted_dates.reshape(1,-1)[0]
```

Confirmed:

```
[ ] plt.figure(figsize=(16,9))
    plt.plot(adjusted_dates,Y_confirmed)
    plt.title('Confirmed Cases over Time',size=30)
    plt.xlabel('Days since 1/1/2020',size=30)
    plt.ylabel('Number of Cases',size=30)
    plt.xticks(size=20)
    plt.yticks(size=20)
    plt.show()
```

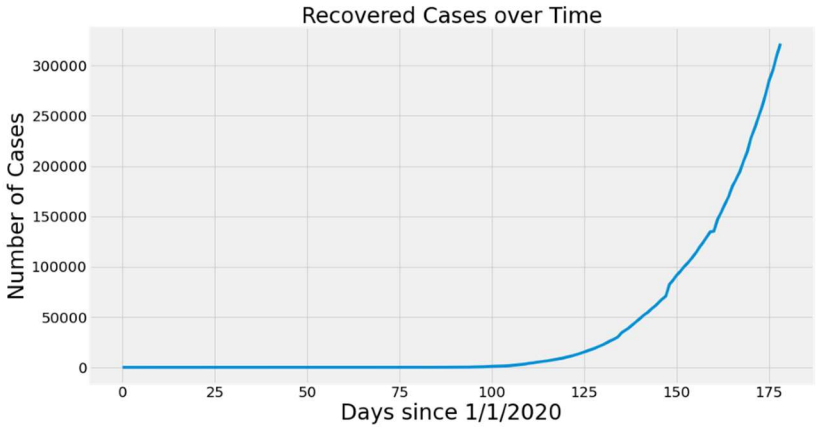
Output:



Recovered:

```
[ ]  plt.figure(figsize=(16,9))
     plt.plot(adjusted_dates,Y_recovered)
     plt.title('Recovered Cases over Time',size=30)
     plt.xlabel('Days since 1/1/2020',size=30)
     plt.ylabel('Number of Cases',size=30)
     plt.xticks(size=20)
     plt.yticks(size=20)
     plt.show()
```

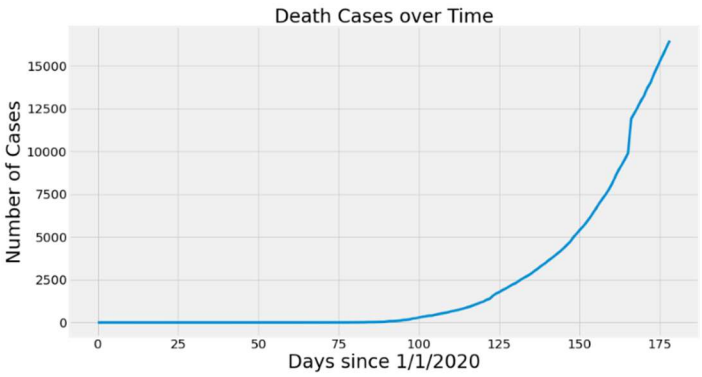
Output:



Deaths:

```
[ ]  plt.figure(figsize=(16,9))
     plt.plot(adjusted_dates,Y_recovered)
     plt.title('Recovered Cases over Time',size=30)
     plt.xlabel('Days since 1/1/2020',size=30)
     plt.ylabel('Number of Cases',size=30)
     plt.xticks(size=20)
     plt.yticks(size=20)
     plt.show()
```

Output:



6.5 DATA SPLITTING:

Split the dataset by taking 20% of data as test set and 80% as train set.

Confirmed

Confirmed :

```
[ ] X_train_confirmed,X_test_confirmed,Y_train_confirmed,Y_test_confirmed=train_test_split(X,Y_confirmed,test_size=0.2)
```

Recovered:

```
[ ] X_train_recovered,X_test_recovered,Y_train_recovered,Y_test_recovered=train_test_split(X,Y_recovered,test_size=0.2,shuffle=False)
```

Deaths:

```
X_train_deaths,X_test_deaths,Y_train_deaths,Y_test_deaths=train_test_split(X,Y_deaths,test_size=0.2,shuffle=False)
```

6.6 MODELLING

Prediction is a typical data science exercise that helps the administration with function planning, objective setting, and anomaly detection.

Machine Learning Model	Confirmed	Recovered	Deaths
SVM	327.764455033243	320.1904442973551	46.68268522744836
Random Forest	34.51638097419188	31.832068037681047	8.185522449897613
Decision Tree	61.154176744792615	39.40353904015335	8.237785570838264
Logistic Regression	149.5009290197816	78.12258885168161	10.92652226872251
Linear Regression	289.3116283104797	267.72517997069355	48.205087402502244

Table 6.1: Root mean absolute error of Machine Learning Algorithms

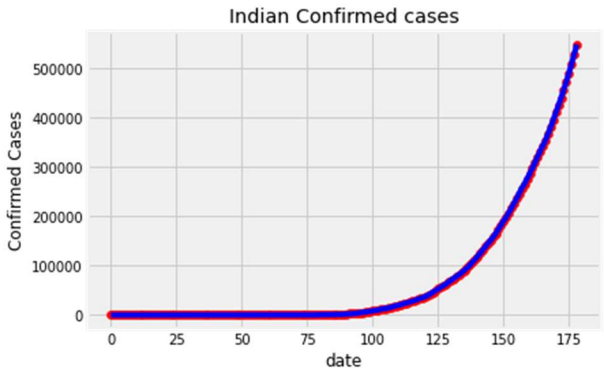
Computed the values of the Covid-19 cases for next 20 days by using different algorithms by using following algorithms

6.6.1 Linear Regression:

```
[ ] start='2020-01-01'
    start_date=datetime.datetime.strptime(start,'%Y-%m-%d')
    future_forecast_dates=[]
    for i in range(len(future_forecast)):
      future_forecast_dates.append((start_date+datetime.timedelta(days=i)).strftime('%Y-%m-%d'))
    # X[0] = pd.to_datetime(X[0], format='%Y-%m-%d')
```

```
plt.scatter(X,Y_confirmed,color='red')
plt.plot(X,Y_confirmed,color='blue')
plt.title('Indian Confirmed cases')
plt.xlabel('date')
plt.ylabel('Confirmed Cases')
plt.show()
```

Output:



Confirmed :

```
[ ] poly=PolynomialFeatures(degree=3)
    poly_X_train_confirmed=poly.fit_transform(X_train_confirmed)
    poly_X_test_confirmed=poly.fit_transform(X_test_confirmed)
    poly_future_forecast=poly.fit_transform(future_forecast)
```

```
[ ] poly.fit(poly_X_train_confirmed,Y_train_confirmed)
```

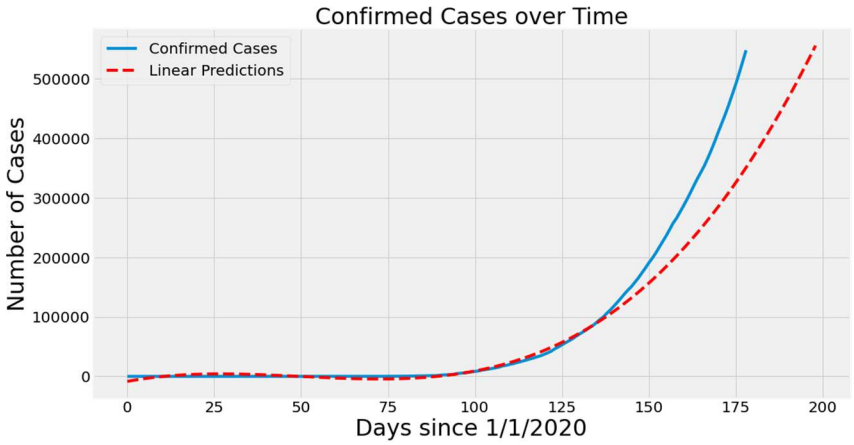
```
PolynomialFeatures(degree=3, include_bias=True, interaction_only=False,
                   order='C')
```

```
regressor=LinearRegression(normalize=True,fit_intercept=False)
regressor.fit(poly_X_train_confirmed,Y_train_confirmed)
```

```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=True)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_confirmed)
plt.plot(future_forecast,future_pred_confirmed,linestyle='dashed',color='red')
plt.title('Confirmed Cases over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Confirmed Cases','Linear Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of confirmed cases for next 20 days

```
[ ] tmp=future_pred_confirmed
    future_pred_confirmed=np.append(tmp[0],tmp[1:])
```

```
linear_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Linear Predicted # number of Confirmed cases in India':np.round(future_pred_confirmed[-20:])})
linear_df
```

Output:

	Date	Linear Predicted # number of Confirmed cases in India
0	2020-06-28	359849.0
1	2020-06-29	368832.0
2	2020-06-30	377959.0
3	2020-07-01	387229.0
4	2020-07-02	396645.0
5	2020-07-03	406208.0
6	2020-07-04	415918.0
7	2020-07-05	425777.0
8	2020-07-06	435785.0
9	2020-07-07	445945.0
10	2020-07-08	456257.0
11	2020-07-09	466721.0
12	2020-07-10	477340.0
13	2020-07-11	488114.0
14	2020-07-12	499045.0
15	2020-07-13	510133.0
16	2020-07-14	521380.0
17	2020-07-15	532786.0
18	2020-07-16	544354.0
19	2020-07-17	556083.0

Recovered:

```
[ ] poly=PolynomialFeatures(degree=3)
    poly_X_train_recovered=poly.fit_transform(X_train_recovered)
    poly_X_test_recovered=poly.fit_transform(X_test_recovered)
    poly_future_forecast=poly.fit_transform(future_forecast)
```

```
[ ] poly.fit(poly_X_train_recovered,Y_train_recovered)
```

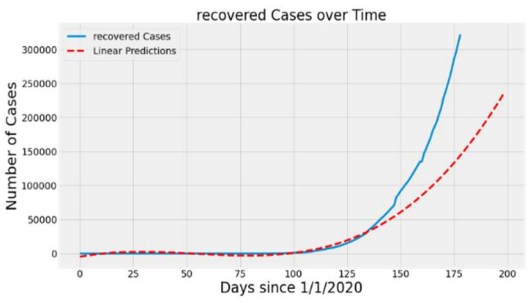
```
PolynomialFeatures(degree=3, include_bias=True, interaction_only=False,
                   order='C')
```

```
[ ] regressor=LinearRegression(normalize=True,fit_intercept=False)
    regressor.fit(poly_X_train_recovered,Y_train_recovered)
```

```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=True)
```

```
[ ] plt.figure(figsize=(16,9))
    plt.plot(adjusted_dates,Y_recovered)
    plt.plot(future_forecast,future_pred_recovered,linestyle='dashed',color='red')
    plt.title('recovered Cases over Time',size=30)
    plt.xlabel('Days since 1/1/2020',size=30)
    plt.ylabel('Number of Cases',size=30)
    plt.legend(['recovered Cases','Linear Predictions'],prop={'size':20})
    plt.xticks(size=20)
    plt.yticks(size=20)
    plt.show()
```

Output:



Predicted number of recovered cases for next 20 days

```
[ ] tmp=future_pred_recovered
    future_pred_recovered=np.append(tmp[0],tmp[1:])
```

```
linear_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Recovered Predicted # number of Confirmed cases in India':np.round(future_pred_recovered[-20:])})
linear_df
```

Output:

	Date	Recovered Predicted # number of Confirmed cases in India
0	2020-06-28	147991.0
1	2020-06-29	151888.0
2	2020-06-30	155851.0
3	2020-07-01	159878.0
4	2020-07-02	163971.0
5	2020-07-03	168131.0
6	2020-07-04	172357.0
7	2020-07-05	176650.0
8	2020-07-06	181012.0
9	2020-07-07	185441.0
10	2020-07-08	189939.0
11	2020-07-09	194507.0
12	2020-07-10	199144.0
13	2020-07-11	203852.0
14	2020-07-12	208631.0
15	2020-07-13	213481.0
16	2020-07-14	218402.0
17	2020-07-15	223397.0
18	2020-07-16	228464.0
19	2020-07-17	233604.0

Deaths:

```
[ ] poly=PolynomialFeatures(degree=3)
    poly_X_train_deaths=poly.fit_transform(X_train_deaths)
    poly_X_test_deaths=poly.fit_transform(X_test_deaths)
    poly_future_forecast=poly.fit_transform(future_forecast)
```

```
[ ] poly.fit(poly_X_train_deaths,Y_train_deaths)
```

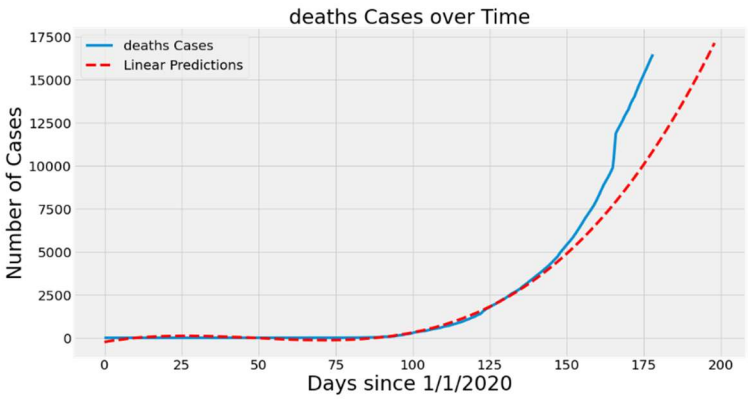
```
PolynomialFeatures(degree=3, include_bias=True, interaction_only=False,
                   order='C')
```

```
regressor=LinearRegression(normalize=True,fit_intercept=False)
regressor.fit(poly_X_train_deaths,Y_train_deaths)
```

```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=True)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_deaths)
plt.plot(future_forecast,future_pred_deaths,linestyle='dashed',color='red')
plt.title('deaths Cases over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['deaths Cases','Linear Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of deaths for next 20 days

```
[ ] tmp=future_pred_deaths
    future_pred_deaths=np.append(tmp[0],tmp[1:])
```

```
linear_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Linear Predicted # number of Deaths cases in India':np.round(future_pred_deaths[-20:])})
linear_df
```

Output:

	Date	Linear Predicted # number of Deaths cases in India
0	2020-06-28	11128.0
1	2020-06-29	11403.0
2	2020-06-30	11683.0
3	2020-07-01	11967.0
4	2020-07-02	12256.0
5	2020-07-03	12549.0
6	2020-07-04	12846.0
7	2020-07-05	13148.0
8	2020-07-06	13455.0
9	2020-07-07	13766.0
10	2020-07-08	14081.0
11	2020-07-09	14402.0
12	2020-07-10	14727.0
13	2020-07-11	15057.0
14	2020-07-12	15391.0
15	2020-07-13	15730.0
16	2020-07-14	16075.0
17	2020-07-15	16424.0
18	2020-07-16	16778.0
19	2020-07-17	17136.0

6.6.2 Logistic Regression:

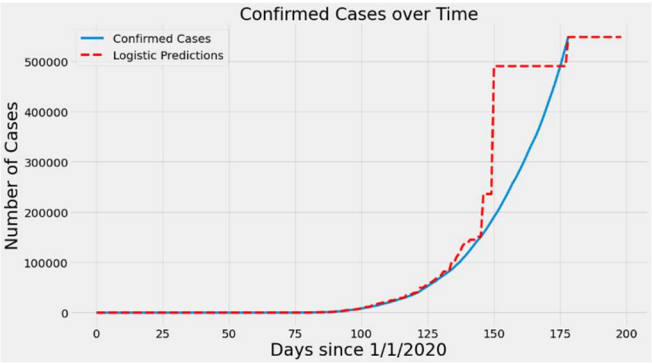
```
[ ] start='2020-01-01'
    start_date=datetime.datetime.strptime(start,'%Y-%m-%d')
    future_forecast_dates=[]
    for i in range(len(future_forecast)):
      future_forecast_dates.append((start_date+datetime.timedelta(days=i)).strftime('%Y-%m-%d'))
    # X[0] = pd.to_datetime(X[0], format='%Y-%m-%d')
```

Confirmed:

```
[ ]  logistic_confirmed=LogisticRegression()
     logistic_confirmed.fit(X_train_confirmed,Y_train_confirmed)
     logistic_pred=logistic_confirmed.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_confirmed)
plt.plot(future_forecast,logistic_pred,linestyle='dashed',color='red')
plt.title('Confirmed Cases over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Confirmed Cases','Logistic Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of confirmed cases for next 20 days

```
[ ]  len(future_forecast_dates),len(logistic_pred)
```

```
(199, 199)
```

```
svm_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Logistic Predicted # number of Confirmed cases in India':np.round(logistic_pred[-20:])})
svm_df
```

Output:

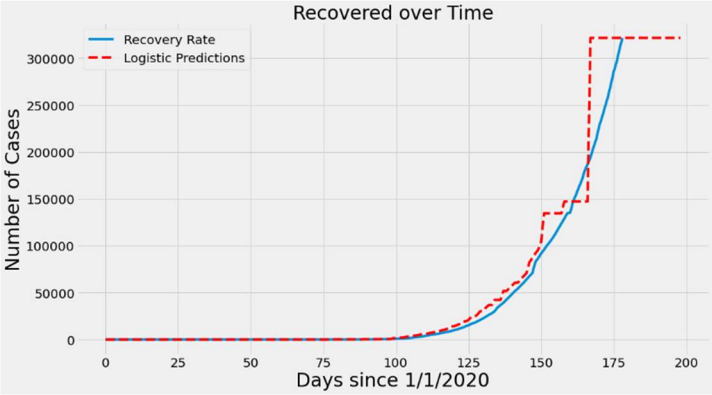
	Date	Logistic Predicted # number of Confirmed cases in India
0	2020-06-28	548318
1	2020-06-29	548318
2	2020-06-30	548318
3	2020-07-01	548318
4	2020-07-02	548318
5	2020-07-03	548318
6	2020-07-04	548318
7	2020-07-05	548318
8	2020-07-06	548318
9	2020-07-07	548318
10	2020-07-08	548318
11	2020-07-09	548318
12	2020-07-10	548318
13	2020-07-11	548318
14	2020-07-12	548318
15	2020-07-13	548318
16	2020-07-14	548318
17	2020-07-15	548318
18	2020-07-16	548318
19	2020-07-17	548318

Recovered:

```
[ ] logistic_recovered=LogisticRegression()
    logistic_recovered.fit(X_train_recovered,Y_train_recovered)
    logistic_pred=logistic_recovered.predict(future_forecast)
```

```
[ ] plt.figure(figsize=(16,9))
    plt.plot(adjusted_dates,Y_recovered)
    plt.plot(future_forecast,logistic_pred,linestyle='dashed',color='red')
    plt.title('Recovered over Time',size=30)
    plt.xlabel('Days since 1/1/2020',size=30)
    plt.ylabel('Number of Cases',size=30)
    plt.legend(['Recovery Rate','Logistic Predictions'],prop={'size':20})
    plt.xticks(size=20)
    plt.yticks(size=20)
    plt.show()
```

Output:



Predicted number of recovered cases for next 20 days

```
[ ] logistic_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Logistic Predicted # number of Recovered cases in India':np.round(logistic_pred[-20:])})
    logistic_df
```

Output:

[ ]

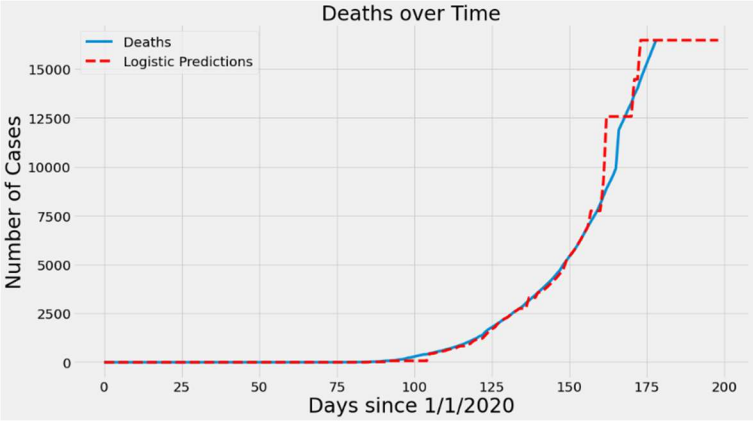
	Date	Logistic Predicted # number of Recovered cases in India
0	2020-06-28	321723
1	2020-06-29	321723
2	2020-06-30	321723
3	2020-07-01	321723
4	2020-07-02	321723
5	2020-07-03	321723
6	2020-07-04	321723
7	2020-07-05	321723
8	2020-07-06	321723
9	2020-07-07	321723
10	2020-07-08	321723
11	2020-07-09	321723
12	2020-07-10	321723
13	2020-07-11	321723
14	2020-07-12	321723
15	2020-07-13	321723
16	2020-07-14	321723
17	2020-07-15	321723
18	2020-07-16	321723
19	2020-07-17	321723

Deaths:

```
[ ] logistic_deaths=LogisticRegression()
    logistic_deaths.fit(X_train_deaths,Y_train_deaths)
    logistic_pred=logistic_deaths.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_deaths)
plt.plot(future_forecast,logistic_pred,linestyle='dashed',color='red')
plt.title('Deaths over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Deaths','Logistic Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of deaths for next 20 days

```
[ ] logistic_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Logistic Predicted # number of Deaths in India':np.round(logistic_pred[-20:])})
    logistic_df
```

Output:

[ ]

	Date	Logistic Predicted # number of Deaths in India
0	2020-06-28	16475
1	2020-06-29	16475
2	2020-06-30	16475
3	2020-07-01	16475
4	2020-07-02	16475
5	2020-07-03	16475
6	2020-07-04	16475
7	2020-07-05	16475
8	2020-07-06	16475
9	2020-07-07	16475
10	2020-07-08	16475
11	2020-07-09	16475
12	2020-07-10	16475
13	2020-07-11	16475
14	2020-07-12	16475
15	2020-07-13	16475
16	2020-07-14	16475
17	2020-07-15	16475
18	2020-07-16	16475
19	2020-07-17	16475

6.6.3 Support vector machine (SVM):

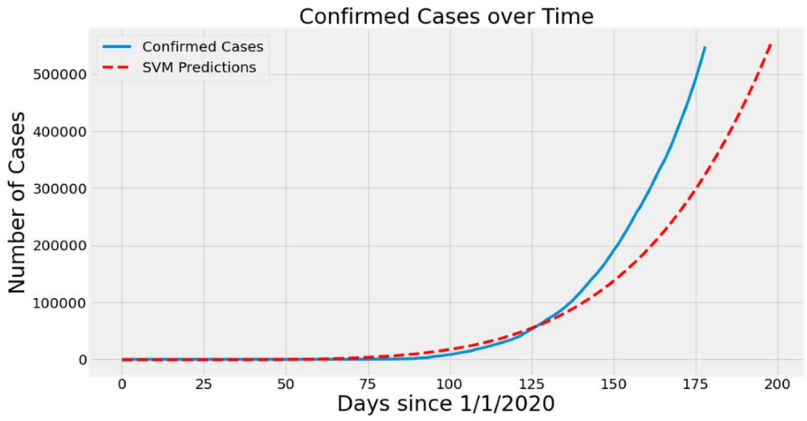
```
start='2020-01-01'
start_date=datetime.datetime.strptime(start,'%Y-%m-%d')
future_forecast_dates=[]
for i in range(len(future_forecast)):
  future_forecast_dates.append((start_date+datetime.timedelta(days=i)).strftime('%Y-%m-%d'))
# X[0] = pd.to_datetime(X[0], format='%Y-%m-%d')
```

Confirmed:

```
svm_confirmed=SVR(shrinking=True,kernel='poly',gamma=0.01,epsilon=1,degree=5,C=0.1)
svm_confirmed.fit(X_train_confirmed,Y_train_confirmed)
svm_pred=svm_confirmed.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_confirmed)
plt.plot(future_forecast,svm_pred,linestyle='dashed',color='red')
plt.title('Confirmed Cases over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Confirmed Cases','SVM Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

OUTPUT:



Prediction number of confirmed cases for next 20 days

```
svm_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'SVM Predicted # number of Confirmed cases in India':np.round(svm_pred[-20:])})
svm_df
```

Output:

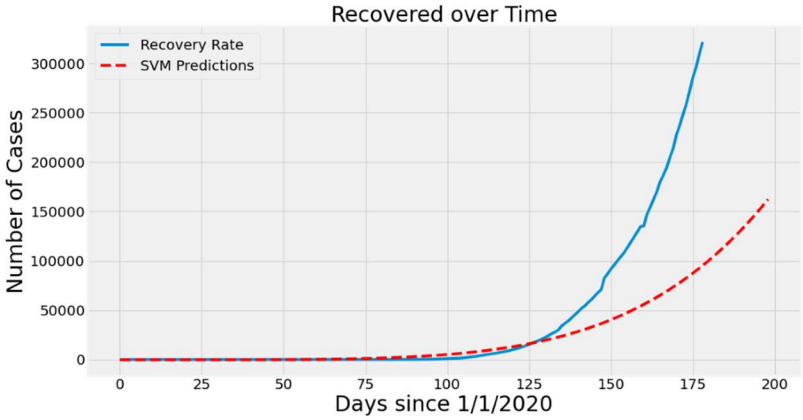
	Date	SVM Predicted # number of Confirmed cases in India
0	2020-06-28	333405.0
1	2020-06-29	342849.0
2	2020-06-30	352505.0
3	2020-07-01	362377.0
4	2020-07-02	372469.0
5	2020-07-03	382783.0
6	2020-07-04	393324.0
7	2020-07-05	404096.0
8	2020-07-06	415101.0
9	2020-07-07	426345.0
10	2020-07-08	437830.0
11	2020-07-09	449561.0
12	2020-07-10	461542.0
13	2020-07-11	473776.0
14	2020-07-12	486268.0
15	2020-07-13	499021.0
16	2020-07-14	512040.0
17	2020-07-15	525329.0
18	2020-07-16	538892.0
19	2020-07-17	552732.0

Recovered :

```
svm_recovered=SVR(shrinking=True,kernel='poly',gamma=0.01,epsilon=1,degree=5,C=0.1)
svm_recovered.fit(X_train_recovered,Y_train_recovered)
svm_pred=svm_recovered.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_recovered)
plt.plot(future_forecast,svm_pred,linestyle='dashed',color='red')
plt.title('Recovered over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Recovery Rate','SVM Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Prediction number of recovered cases for next 20 days

```
svm_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'SVM Predicted # number of Recovered cases in India':np.round(svm_pred[-20:])})
svm_df
```

Output:

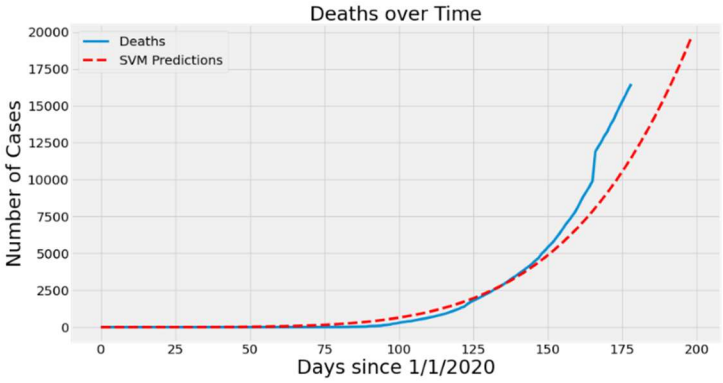
	Date	SVM Predicted # number of Recovered cases in India
0	2020-06-28	97932.0
1	2020-06-29	100706.0
2	2020-06-30	103543.0
3	2020-07-01	106443.0
4	2020-07-02	109407.0
5	2020-07-03	112437.0
6	2020-07-04	115534.0
7	2020-07-05	118698.0
8	2020-07-06	121931.0
9	2020-07-07	125233.0
10	2020-07-08	128607.0
11	2020-07-09	132053.0
12	2020-07-10	135573.0
13	2020-07-11	139166.0
14	2020-07-12	142836.0
15	2020-07-13	146582.0
16	2020-07-14	150407.0
17	2020-07-15	154310.0
18	2020-07-16	158294.0
19	2020-07-17	162360.0

Deaths:

```
svm_deaths=SVR(shrinking=True,kernel='poly',gamma=0.01,epsilon=1,degree=5,C=0.1)
svm_deaths.fit(X_train_deaths,Y_train_deaths)
svm_pred=svm_deaths.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_deaths)
plt.plot(future_forecast,svm_pred,linestyle='dashed',color='red')
plt.title('Deaths over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Deaths','SVM Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Prediction number of death cases for next 20 days

```
svm_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'SVM Predicted # number of Deaths in India':np.round(svm_pred[-20:])})
svm_df
```

Output:

	Date	SVM Predicted # number of Deaths in India
0	2020-06-28	11797.0
1	2020-06-29	12130.0
2	2020-06-30	12471.0
3	2020-07-01	12819.0
4	2020-07-02	13175.0
5	2020-07-03	13539.0
6	2020-07-04	13911.0
7	2020-07-05	14291.0
8	2020-07-06	14680.0
9	2020-07-07	15076.0
10	2020-07-08	15482.0
11	2020-07-09	15896.0
12	2020-07-10	16318.0
13	2020-07-11	16750.0
14	2020-07-12	17191.0
15	2020-07-13	17641.0
16	2020-07-14	18100.0
17	2020-07-15	18569.0
18	2020-07-16	19048.0
19	2020-07-17	19536.0

6.6.4 Decision Tree:

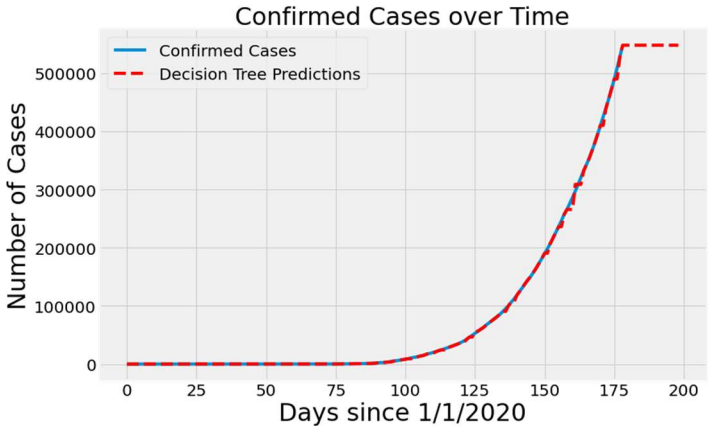
```
start='2020-01-01'
start_date=datetime.datetime.strptime(start,'%Y-%m-%d')
future_forecast_dates=[]
for i in range(len(future_forecast)):
  future_forecast_dates.append((start_date+datetime.timedelta(days=i)).strftime('%Y-%m-%d'))
# X[0] = pd.to_datetime(X[0], format='%Y-%m-%d')
```

Confirmed:

```
[ ] dt_confirmed=DecisionTreeClassifier( criterion='entropy', random_state=0)
    dt_confirmed.fit(X_train_confirmed,Y_train_confirmed)
    dt_pred=dt_confirmed.predict(future_forecast)
```

```
[ ] plt.figure(figsize=(12,8))
    plt.plot(adjusted_dates,Y_confirmed)
    plt.plot(future_forecast,dt_pred,linestyle='dashed',color='red')
    plt.title('Confirmed Cases over Time',size=30)
    plt.xlabel('Days since 1/1/2020',size=30)
    plt.ylabel('Number of Cases',size=30)
    plt.legend(['Confirmed Cases','Decision Tree Predictions'],prop={'size':20})
    plt.xticks(size=20)
    plt.yticks(size=20)
    plt.show()
```

Output:



Predicted number of confirmed cases for next 20 days:

```
[ ] dt_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Decision Tree Predicted # number of Confirmed cases in India':np.round(dt_pred[-20:])})
    dt_df
```

Output:

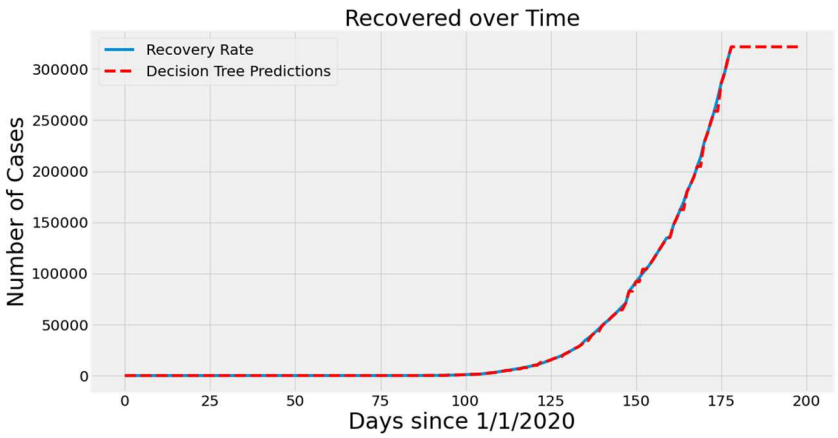
	Date	Decision Tree Predicted # number of Confirmed cases in India
0	2020-06-28	548318
1	2020-06-29	548318
2	2020-06-30	548318
3	2020-07-01	548318
4	2020-07-02	548318
5	2020-07-03	548318
6	2020-07-04	548318
7	2020-07-05	548318
8	2020-07-06	548318
9	2020-07-07	548318
10	2020-07-08	548318
11	2020-07-09	548318
12	2020-07-10	548318
13	2020-07-11	548318
14	2020-07-12	548318
15	2020-07-13	548318
16	2020-07-14	548318
17	2020-07-15	548318
18	2020-07-16	548318
19	2020-07-17	548318

Recovered:

```
[ ] dt_recovered=DecisionTreeClassifier( criterion='entropy', random_state=0)
    dt_recovered.fit(X_train_recovered,Y_train_recovered)
    dt_pred=dt_recovered.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_recovered)
plt.plot(future_forecast,dt_pred,linestyle='dashed',color='red')
plt.title('Recovered over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Recovery Rate','Decision Tree Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of recovered cases for next 20 days:

```
[ ] dt_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Decision Tree Predicted # number of Recovered cases in India':np.round(dt_pred[-20:])})
    dt_df
```

Output:

[ ]

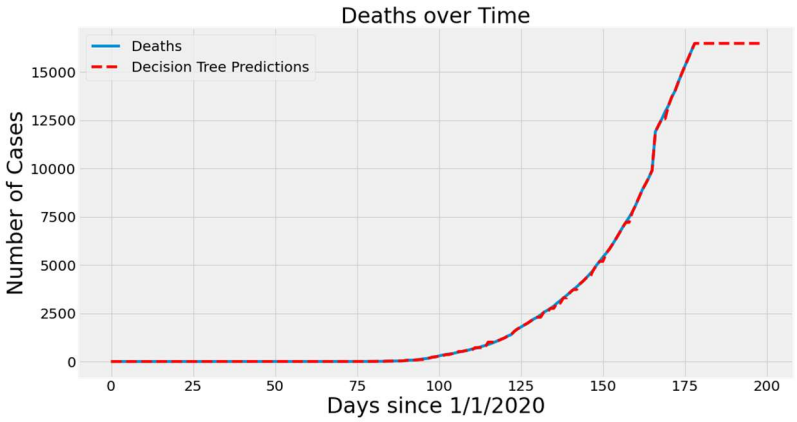
	Date	Decision Tree Predicted # number of Recovered cases in India
0	2020-06-28	321723
1	2020-06-29	321723
2	2020-06-30	321723
3	2020-07-01	321723
4	2020-07-02	321723
5	2020-07-03	321723
6	2020-07-04	321723
7	2020-07-05	321723
8	2020-07-06	321723
9	2020-07-07	321723
10	2020-07-08	321723
11	2020-07-09	321723
12	2020-07-10	321723
13	2020-07-11	321723
14	2020-07-12	321723
15	2020-07-13	321723
16	2020-07-14	321723
17	2020-07-15	321723
18	2020-07-16	321723
19	2020-07-17	321723

Deaths:

```
[ ] dt_deaths=DecisionTreeClassifier( criterion='entropy', random_state=0)
    dt_deaths.fit(X_train_deaths,Y_train_deaths)
    dt_pred=dt_deaths.predict(future_forecast)
```

```
[ ]  plt.figure(figsize=(16,9))
     plt.plot(adjusted_dates,Y_deaths)
     plt.plot(future_forecast,dt_pred,linestyle='dashed',color='red')
     plt.title('Deaths over Time',size=30)
     plt.xlabel('Days since 1/1/2020',size=30)
     plt.ylabel('Number of Cases',size=30)
     plt.legend(['Deaths','Decision Tree Predictions'],prop={'size':20})
     plt.xticks(size=20)
     plt.yticks(size=20)
     plt.show()
```

Output:



Predicted number of deaths for next 20 days:

```
[ ]  dt_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Decision Tree Predicted # number of Deaths in India':np.round(dt_pred[-20:])})
     dt_df
```

Output:

	Date	Decision Tree Predicted # number of Deaths in India
0	2020-06-28	16475
1	2020-06-29	16475
2	2020-06-30	16475
3	2020-07-01	16475
4	2020-07-02	16475
5	2020-07-03	16475
6	2020-07-04	16475
7	2020-07-05	16475
8	2020-07-06	16475
9	2020-07-07	16475
10	2020-07-08	16475
11	2020-07-09	16475
12	2020-07-10	16475
13	2020-07-11	16475
14	2020-07-12	16475
15	2020-07-13	16475
16	2020-07-14	16475
17	2020-07-15	16475
18	2020-07-16	16475
19	2020-07-17	16475

6.6.5 Random Forest:

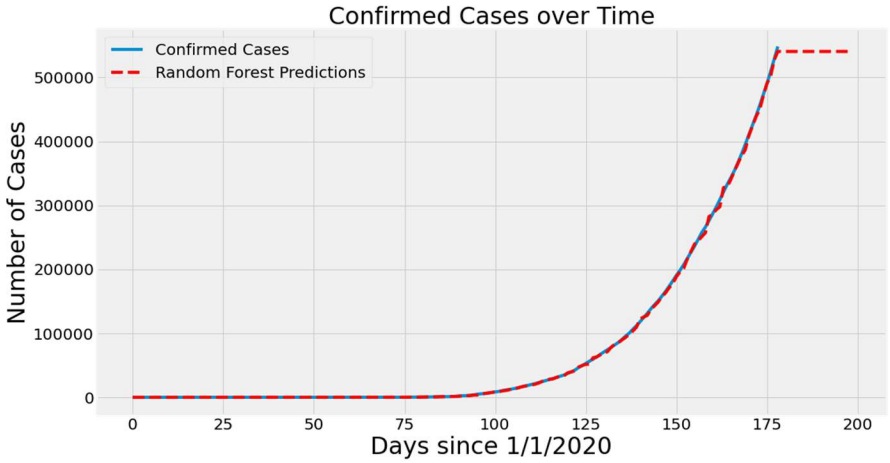
```
start='2020-01-01'
start_date=datetime.datetime.strptime(start,'%Y-%m-%d')
future_forecast_dates=[]
for i in range(len(future_forecast)):
  future_forecast_dates.append((start_date+datetime.timedelta(days=i)).strftime('%Y-%m-%d'))
# X[0] = pd.to_datetime(X[0], format='%Y-%m-%d')
```

Confirmed:

```
[ ] rf_confirmed=RandomForestRegressor(n_estimators=10,random_state=0)
    rf_confirmed.fit(X_train_confirmed,Y_train_confirmed)
    rf_pred=rf_confirmed.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_confirmed)
plt.plot(future_forecast,rf_pred,linestyle='dashed',color='red')
plt.title('Confirmed Cases over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Confirmed Cases','Random Forest Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of confirmed cases for next 20 days:

```
[ ] len(future_forecast_dates),len(rf_pred)
```

```
(199, 199)
```

```
[ ] rf_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Naive Bayes Predicted # number of Confirmed cases in India':np.round(rf_pred[-20:])})
    rf_df
```

Output:

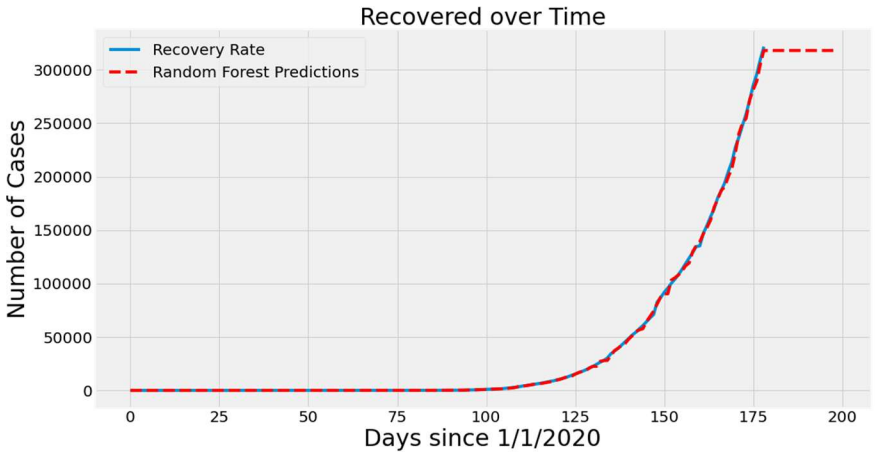
	Date	Naive Bayes Predicted # number of Confirmed cases in India
0	2020-06-28	540580.0
1	2020-06-29	540580.0
2	2020-06-30	540580.0
3	2020-07-01	540580.0
4	2020-07-02	540580.0
5	2020-07-03	540580.0
6	2020-07-04	540580.0
7	2020-07-05	540580.0
8	2020-07-06	540580.0
9	2020-07-07	540580.0
10	2020-07-08	540580.0
11	2020-07-09	540580.0
12	2020-07-10	540580.0
13	2020-07-11	540580.0
14	2020-07-12	540580.0
15	2020-07-13	540580.0
16	2020-07-14	540580.0
17	2020-07-15	540580.0
18	2020-07-16	540580.0
19	2020-07-17	540580.0

Recovered:

```
rf_recovered=RandomForestRegressor(n_estimators=10,random_state=0)
rf_recovered.fit(X_train_recovered,Y_train_recovered)
rf_pred=rf_recovered.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_recovered)
plt.plot(future_forecast,rf_pred,linestyle='dashed',color='red')
plt.title('Recovered over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Recovery Rate','Random Forest Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of recovered cases in next 20 days:

```
[ ] rf_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Random Forest Predicted # number of Recovered cases in India':np.round(rf_pred[-20:])})
    rf_df
```

Output:

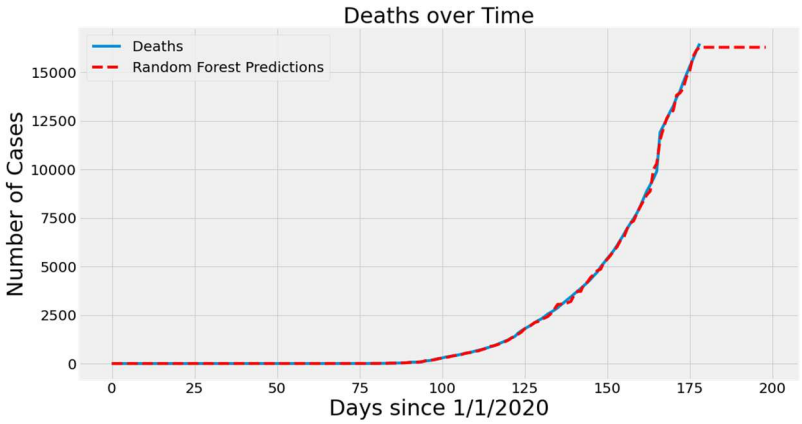
	Date	Random Forest Predicted # number of Recovered cases in India
0	2020-06-28	318114.0
1	2020-06-29	318114.0
2	2020-06-30	318114.0
3	2020-07-01	318114.0
4	2020-07-02	318114.0
5	2020-07-03	318114.0
6	2020-07-04	318114.0
7	2020-07-05	318114.0
8	2020-07-06	318114.0
9	2020-07-07	318114.0
10	2020-07-08	318114.0
11	2020-07-09	318114.0
12	2020-07-10	318114.0
13	2020-07-11	318114.0
14	2020-07-12	318114.0
15	2020-07-13	318114.0
16	2020-07-14	318114.0
17	2020-07-15	318114.0
18	2020-07-16	318114.0
19	2020-07-17	318114.0

Deaths:

```
[ ] rf_deaths=RandomForestRegressor(n_estimators=10,random_state=0)
    rf_deaths.fit(X_train_deaths,Y_train_deaths)
    rf_pred=rf_deaths.predict(future_forecast)
```

```
plt.figure(figsize=(16,9))
plt.plot(adjusted_dates,Y_deaths)
plt.plot(future_forecast,rf_pred,linestyle='dashed',color='red')
plt.title('Deaths over Time',size=30)
plt.xlabel('Days since 1/1/2020',size=30)
plt.ylabel('Number of Cases',size=30)
plt.legend(['Deaths','Random Forest Predictions'],prop={'size':20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

Output:



Predicted number of deaths in next 20 days:

```
[ ] rf_df=pd.DataFrame({'Date':future_forecast_dates[-20:],'Random Forest Predicted # number of Deaths in India':np.round(rf_pred[-20:])})
    rf_df
```

Output:

	Date	Random Forest Predicted # number of Deaths in India
0	2020-06-28	16282.0
1	2020-06-29	16282.0
2	2020-06-30	16282.0
3	2020-07-01	16282.0
4	2020-07-02	16282.0
5	2020-07-03	16282.0
6	2020-07-04	16282.0
7	2020-07-05	16282.0
8	2020-07-06	16282.0
9	2020-07-07	16282.0
10	2020-07-08	16282.0
11	2020-07-09	16282.0
12	2020-07-10	16282.0
13	2020-07-11	16282.0
14	2020-07-12	16282.0
15	2020-07-13	16282.0
16	2020-07-14	16282.0
17	2020-07-15	16282.0
18	2020-07-16	16282.0
19	2020-07-17	16282.0

6.6.6 Prophet:

Confirmed:

```
confirmed.columns = ['ds','y']
#confirmed['ds'] = confirmed['ds'].dt.date
confirmed['ds'] = pd.to_datetime(confirmed['ds'])
```

```
] m = Prophet(interval_width=0.95)
  m.fit(confirmed)
  future = m.make_future_dataframe(periods=20)
  future.tail()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

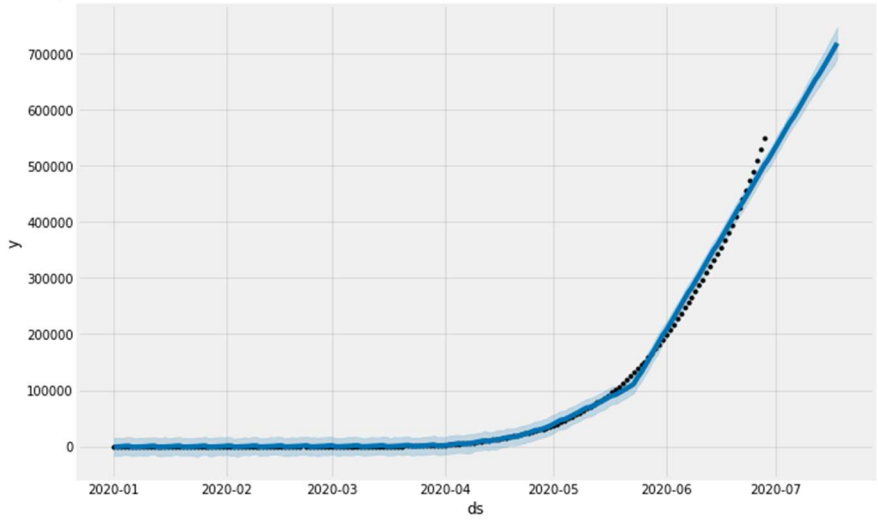
	ds
194	2020-07-14
195	2020-07-15
196	2020-07-16
197	2020-07-17
198	2020-07-18

```
#predicting the future with date, and upper and lower limit of y value
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
194	2020-07-14	673461.832708	652069.213263	698103.598540
195	2020-07-15	684573.993469	661131.042410	709588.921690
196	2020-07-16	695751.068073	670349.359841	723497.339096
197	2020-07-17	707002.012517	677988.591898	734710.751460
198	2020-07-18	718521.641870	691029.467285	747501.111860

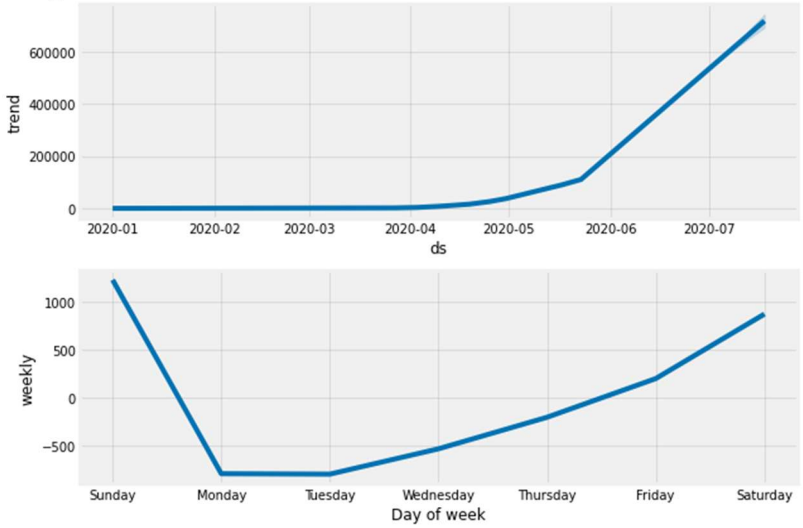
```
[ ]  confirmed_forecast_plot = m.plot(forecast)
```

Output:




```
confirmed_forecast_plot =m.plot_components(forecast)
```

Output:



Recovered:

```
[ ] recovered.columns = ['ds','y']
    recovered['ds'] = pd.to_datetime(recovered['ds'])
```

```
m = Prophet(interval_width=0.95)
m.fit(recovered)
future = m.make_future_dataframe(periods=7)
future.tail()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

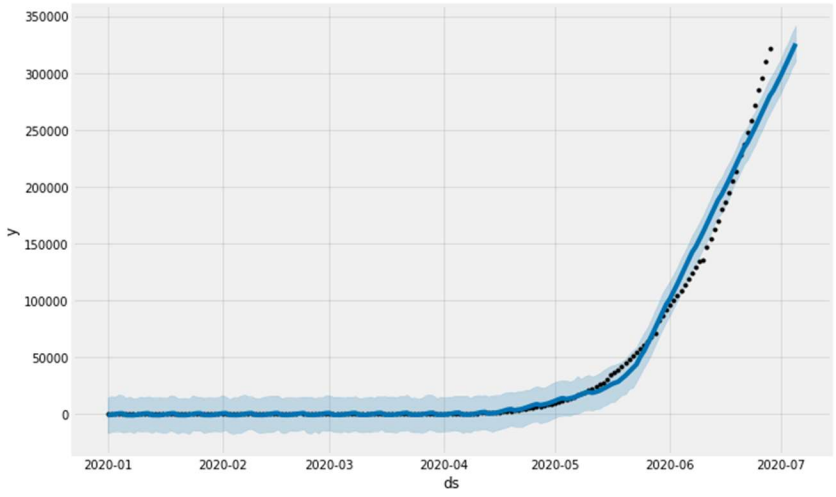
	ds
181	2020-07-01
182	2020-07-02
183	2020-07-03
184	2020-07-04
185	2020-07-05

```
[ ] forecast = m.predict(future)
    forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
181	2020-07-01	297862.781891	282642.777331	312448.441466
182	2020-07-02	304886.653168	290134.116983	321065.453923
183	2020-07-03	311899.997613	296387.113945	327052.498722
184	2020-07-04	319093.747107	304760.499680	335151.767951
185	2020-07-05	325810.365710	310516.145468	341669.063152

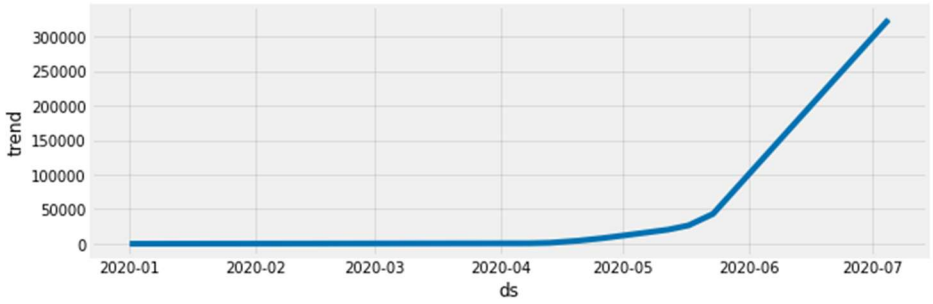
```
recovered_forecast_plot = m.plot(forecast)
```

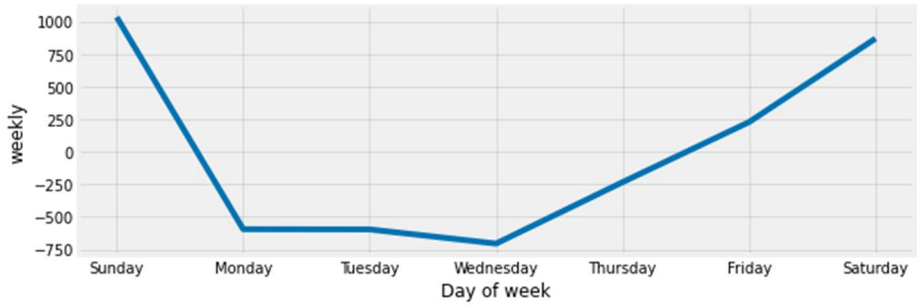
Output:




```
[ ] recovered_forecast_plot = m.plot_components(forecast)
```

Output:





Deaths:

```
deaths.columns = ['ds','y']
deaths['ds'] = pd.to_datetime(deaths['ds'])
```

```
m = Prophet(interval_width=0.95)
m.fit(deaths)
future = m.make_future_dataframe(periods=7)
future.tail()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

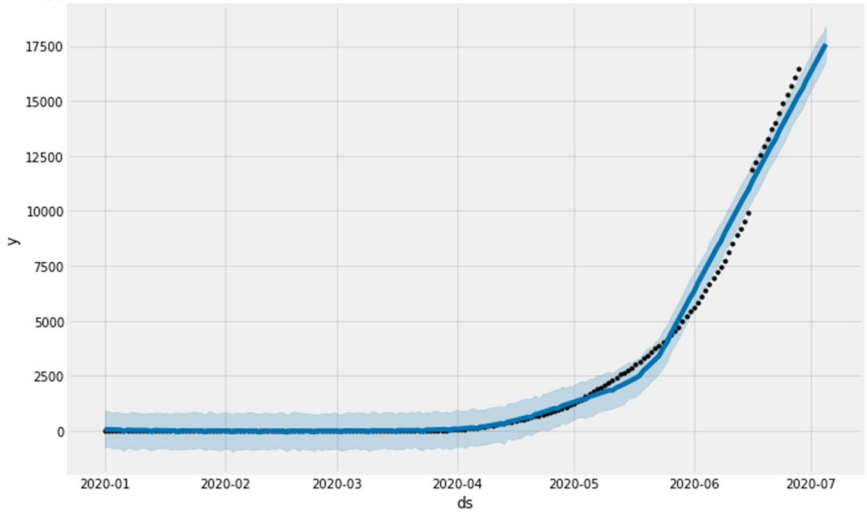
	ds
181	2020-07-01
182	2020-07-02
183	2020-07-03
184	2020-07-04
185	2020-07-05

```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
181	2020-07-01	16269.635781	15430.492112	17069.673059
182	2020-07-02	16600.628079	15790.778174	17474.098577
183	2020-07-03	16933.385971	16070.487452	17802.597846
184	2020-07-04	17264.791639	16433.338403	18055.174366
185	2020-07-05	17591.092015	16821.428042	18392.930518

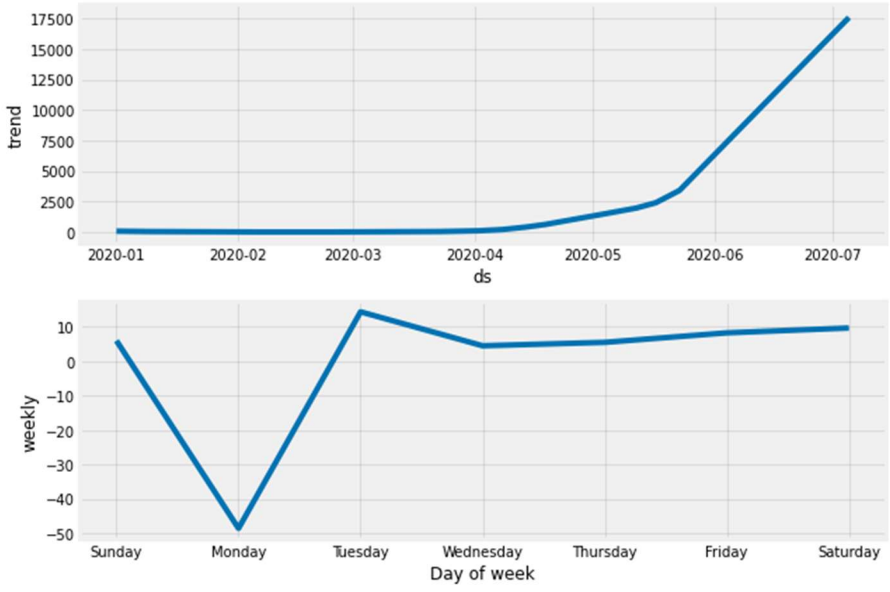
```
[ ] deaths_forecast_plot = m.plot(forecast)
```

Output:




```
deaths_forecast_plot = m.plot_components(forecast)
```

Output:



CHAPTER 7: TESTING

7.1 TESTING PLAN

Testing process starts with a test plan. This plan identifies all the testing related activities that must be performed and specifies the schedules, allocates the resources, and specified guidelines for testing. During the testing of the unit the specified test cases are executed and the actual result compared with expected output. The final output of the testing phase is the test report and the error report.

7.1.1 Test Data

Testing process begins with a test design. This arrangement recognizes all the testing related exercises that must be performed like the timetables, assigning the assets, and determining rules for testing. This testing of the unit of the predetermined experiments are executed and the genuine outcome is expected. The last part of the testing stage is the test report and the error report.

7.1.2 Unit testing

Every individual module has been tried against the necessity with some test information.

7.1.3 Test Report

The module is working appropriately given the client must enter data. All information section frames have tested with indicated test cases and all information passage shapes are working properly.

7.1.4 Error Report

On the off chance that the client does not enter information in determined request, at that point the client will be incited with error messages. Error reduction is done to deal with the normal and sudden mistakes.

CHAPTER 8: RESULT AND DISCUSSION

The following conclusions are drawn from the practiced algorithms that Prophet, SVM, Linear Regression provided better accuracy when compared to the remaining.

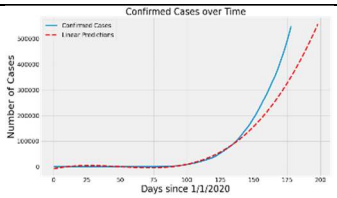
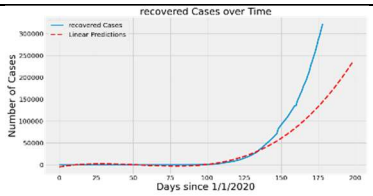
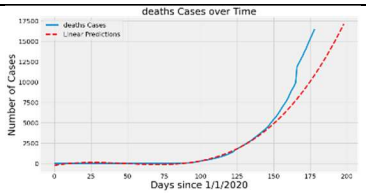
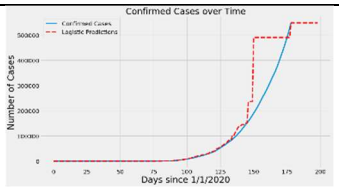
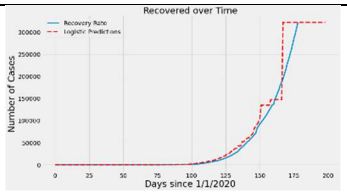
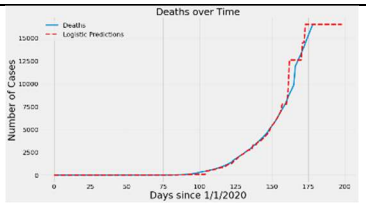
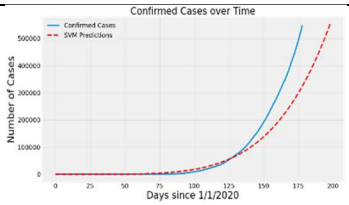
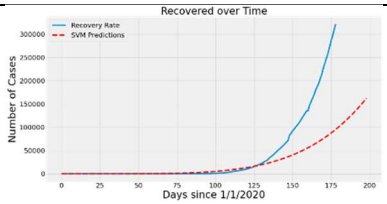
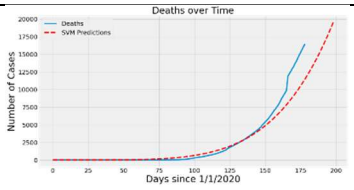
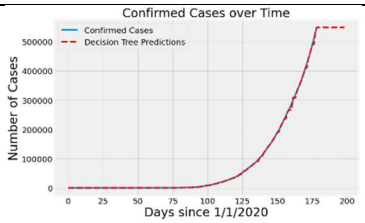
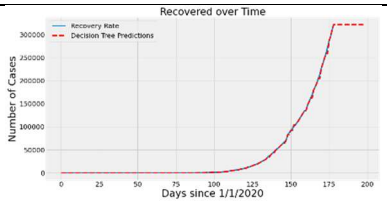
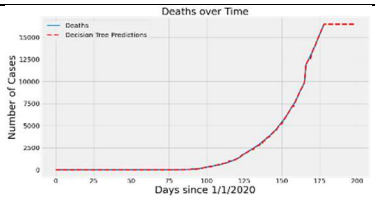
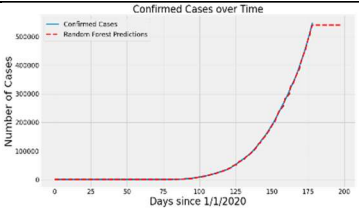
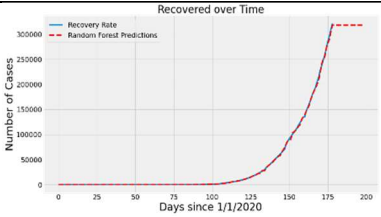
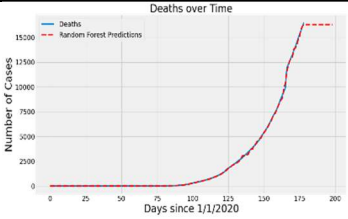
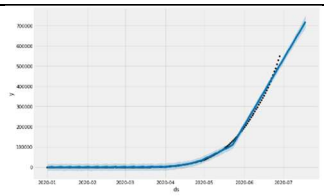
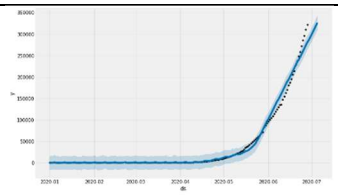
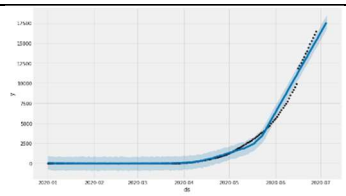
Name of Algorithm	Graph of Prediction for Confirmed Cases	Graph of Prediction for Recovered Cases	Graph of Prediction for Death Cases
Linear Regression			
Logistic Regression			
SVM			
Decision Tree			
Random Forest			
Prophet			

Table 8.1: Prediction Comparisons

Here, Prophet, SVM and Linear Regression has given us the more precise predicted values compared to the others as their curve is not flattened all of a sudden and It can also be seen that their graphs are mostly matched with actual values. Even though the RMSE values of Random Forest Algorithm is less than all other algorithms, the future prediction is given better by the Prophet and SVM Algorithms. The Random Forest algorithm could not give accurate prediction due to overfitting

CHAPTER 9: CONCLUSION AND FUTURE SCOPE

9.1 CONCUSION

Machine learning is an important tool in fighting the current pandemic. If we take this opportunity to collect data, pool our knowledge, and combine our skills, we can save many lives – both now and in the future. The study on the prediction of Covid-19 pandemic infection using machine learning reveals the comparative discussion on the confirmed cases, recovered cases, and deaths in India. While we go through the epidemic status, the lack of proper social distancing and personal hygiene playing an important role in leading to the community widespread. The effective management using symptomatic therapy and quarantine system can control the spread of the disease state up to a limit. Although, the condition is getting worse may question the human existence inside the earth. Hence, undoubtedly, we can fetch out in an assumption that SARS- Covid-19 has been tremendously defeating over larger developed country in the world.

9.2 FUTURE SCOPE

This can be further enhanced to increase the accuracy if tried to use different concepts like ARIMA Model, Autoregressive Model, LSTM and with the day to day new data, these predictions accuracy might change. With the very best accuracy, we can implement some policy changes for the better control of Covid19.

CHAPTER 10: BIBILOGRAPHY

References

[1] http://theprofessionalspoint.blogspot.com/2019/05/advantages-and-disadvantages-of-linear.htmlhttps://www.statisticssolutions.com/what-is-logistic-regression

[2] https://www.researchgate.net/publication/340314461_Predictions_for_COVID-19_outbreak_in_India_using_Epidemiological_models

[3] https://data-flair.training/blogs/applications-of-svm/www.tutorialspoint.com

[4] www.lucidchart.com

[5] www.smartdraw.com

[6] www.researchgate.net

[7] www.geeksforgeeks.org

[8] https://towardsdatascience.com/predicting-the-future-with-facebook-s-prophet-bdfe11af10ff

[9] https://easyai.tech/en/ai-definition/random-forest/

[10] https://www.datarevenue.com/en-blog/machine-learning-covid-19