DATA SCIENCE

(COVID-19 Data Analysis)

Summer Internship Report Submitted in partial fulfillment of the requirement for undergraduate

degree of

Bachelor of Technology

In

Computer Science Engineering

By

Thati Yasasvi

221710313059

Department of Computer Science and Engineering

GITAM School of Technology

GITAM(Deemed to be University)

Hyderabad-502329

June 2020

DECLARATION

I submit this industrial training work entitled "**COVID-19 Data Analysis**" to GITAM(Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology** " in " **Computer Science Engineering**". I declare that it was carried out independently by me under the guidance.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: **Thati Yasasvi**

Hyderabad   **221710313059**

//Certificate

ACKNOWLEDGEMENT

ABSTRACT

COVID-19 outbreak was first reported in Wuhan, China and has spread to 213 Countries and Territories. WHO declared COVID-19 as a Public Health Emergency of International Concern (PHEIC) on 30 January 2020. Naturally, a rising infectious disease involves fast spreading; endangering the health of large numbers of people, and thus requires immediate actions to prevent the disease at the community level. Being COVID-19 pandemic during a very short time span, it is very important to analyze the trend of these spread and infected cases.

In this project, I will attempt to predict the increase in the cases of COVID-19 from various countries by utilizing the previous data present. To achieve this I will be using a Deep learning technique known as Long Short-Term Memory algorithm.

Using a Keras Long Short-Term Memory (LSTM) Model to Covid-19 Data Analysis. LSTMs are very powerful in sequence prediction of confirmed cases because they're able to store past information. This is important in our case because the previous confirmed case is crucial in predicting its future confirmed cases.

Table of Contents:

CHAPTER 6. FEATURE SELECTION

CHAPTER 7. MODEL BUILDING AND EVALUATION

CHAPTER 1
INFORMATION ABOUT DATA SCIENCE

1.1.1 WHAT IS DATA SCIENCE?

Data Science is a blend of various tools, algorithms, and machine learning principles with the goal to discover hidden patterns from the raw data. How is this different from what statisticians have been doing for years?

The answer lies in the difference between explaining and predicting.
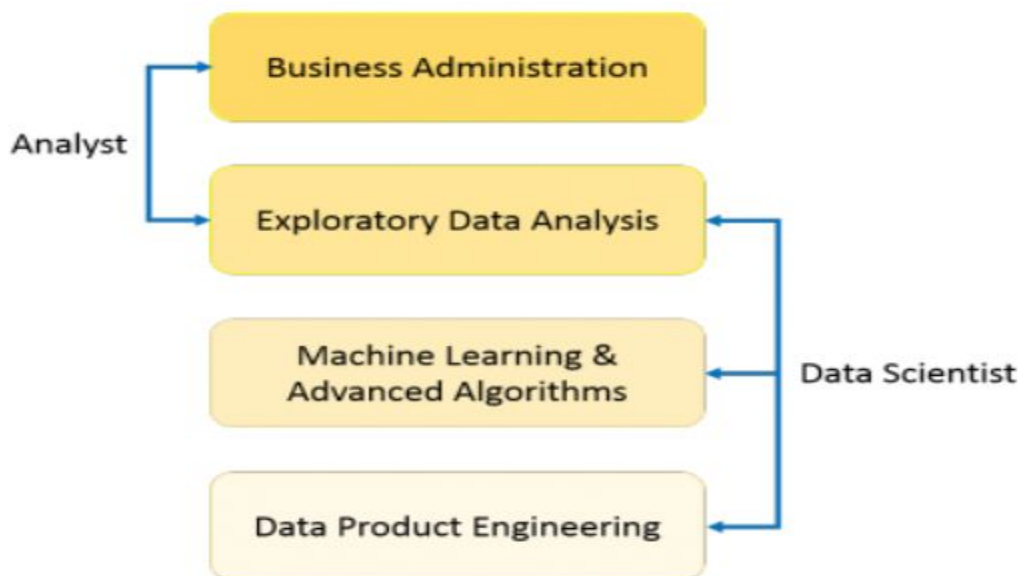
Fig.1.1: Difference between explaining and predicting

As you can see from the above image, a Data Analyst usually explains what is going on by processing the history of the data. On the other hand, Data Scientist not only does exploratory analysis to discover insights from it, but also uses various advanced machine learning algorithms to identify the occurrence of a particular event in the future. A Data Scientist will look at the data from many angles, sometimes angles not known earlier.

1.1.2 NEED OF DATA SCIENCE

Data Science is primarily used to make decisions and predictions making use of predictive causal analytics, prescriptive analytics (predictive plus decision science) and machine learning.

- Predictive causal analytics – If you want a model which can predict the possibilities of a particular event in the future, you need to apply predictive causal analytics. Say, if you are providing money on credit, then the probability of customers making future credit payments on time is a matter of concern for you. Here, you can build a model which can perform predictive analytics on the payment history of the customer to predict if the future payments will be on time or not.

- Prescriptive analytics: If you want a model which has the intelligence of taking its own decisions and the ability to modify it with dynamic parameters, you certainly need prescriptive analytics for it. This relatively new field is all about providing advice. In other terms, it not only predicts but suggests a range of prescribed actions and associated outcomes.
The best example for this is Google's self-driving car which I had discussed earlier too. The data gathered by vehicles can be used to train self-driving cars. You can run algorithms on this data to bring intelligence to it. This will enable your car to take decisions like when to turn, which path to take, when to slow down or speed up.

- Machine learning for making predictions — If you have transactional data of a finance company and need to build a model to determine the future trend, then machine learning algorithms are the best bet. This falls under the paradigm of supervised learning. It is

called supervised because you already have the data based on which you can train your machines. For example, a fraud detection model can be trained using a historical record of fraudulent purchases.

- Machine learning for pattern discovery — If you don't have the parameters based on which you can make predictions, then you need to find out the hidden patterns within the dataset to be able to make meaningful predictions. This is nothing but the unsupervised model as you don't have any predefined labels for grouping. The most common algorithm used for pattern discovery is Clustering.
  Let's say you are working in a telephone company and you need to establish a network by putting towers in a region. Then, you can use the clustering technique to find those tower locations which will ensure that all the users receive optimum signal strength.

Let's see how the proportion of above-described approaches differ for Data Analysis as well as Data Science. As you can see in the image below, Data Analysis includes descriptive analytics and prediction to a certain extent. On the other hand, Data Science is more about Predictive Causal Analytics and Machine Learning.

1.1.3 USES OF DATA SCIENCE

1. Medical Image Analysis

Procedures such as detecting tumors, artery stenosis, organ delineation employ various different methods and frameworks like MapReduce to find optimal parameters for tasks like lung texture classification. It applies machine learning methods, support vector machines (SVM), content-based medical image indexing, and wavelet analysis for solid texture classification.

2. Genetics & Genomics

Data Science applications also enable an advanced level of treatment personalization through research in genetics and genomics. The goal is to understand the impact of the DNA on our

health and find individual biological connections between genetics, diseases, and drug response. Data science techniques allow integration of different kinds of data with genomic data in the disease research, which provides a deeper understanding of genetic issues in reactions to particular drugs and diseases. As soon as we acquire reliable personal genome data, we will achieve a deeper understanding of the human DNA. The advanced genetic risk prediction will be a major step towards more individual care.

3. Drug Development

The drug discovery process is highly complicated and involves many disciplines. The greatest ideas are often bounded by billions of testing, huge financial and time expenditure. On average, it takes twelve years to make an official submission.

Data science applications and machine learning algorithms simplify and shorten this process, adding a perspective to each step from the initial screening of drug compounds to the prediction of the success rate based on the biological factors. Such algorithms can forecast how the compound will act in the body using advanced mathematical modeling and simulations instead of the "lab experiments". The idea behind the computational drug discovery is to create computer model simulations as a biologically relevant network simplifying the prediction of future outcomes with high accuracy.

4. Virtual assistance for patients and customer support

Optimization of the clinical process builds upon the concept that for many cases it is not actually necessary for patients to visit doctors in person. A mobile application can give a more effective solution by *bringing the doctor to the patient instead*.The AI-powered mobile apps can provide basic healthcare support, usually as chatbots. You simply describe your symptoms, or ask questions, and then receive key information about your medical condition derived from a wide network linking symptoms to causes. Apps can remind you to take your medicine on time, and if necessary, assign an appointment with a doctor.This approach promotes a healthy lifestyle by

encouraging patients to make healthy decisions, saves their time waiting in line for an appointment, and allows doctors to focus on more critical cases.

5.Internet Search

Now, this is probably the first thing that strikes your mind when you think Data Science Applications.When we speak of search, we think 'Google'. Right? But there are many other search engines like Yahoo, Bing, Ask, AOL, and so on. All these search engines (including Google) make use of data science algorithms to deliver the best result for our searched query in a fraction of seconds. Considering the fact that, Google processes more than 20 petabytes of data every day.
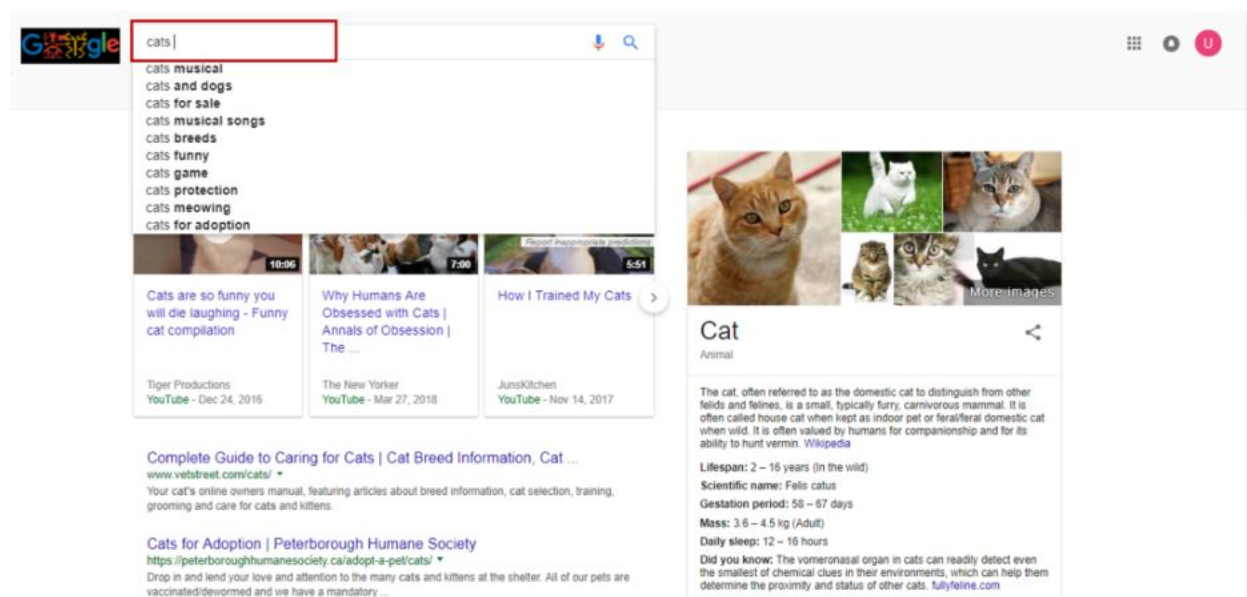
Fig.1.2 Internet search

6.Targeted Advertising

If you thought Search would have been the biggest of all data science applications, here is a challenger – the entire digital marketing spectrum. Starting from the display banners on various

websites to the digital billboards at the airports – almost all of them are decided by using data science algorithms.

This is the reason why digital ads have been able to get a lot higher CTR (Call-Through Rate) than traditional advertisements. They can be targeted based on a user's past behavior.

This is the reason why you might see ads of Data Science Training Programs while I see an ad of apparels in the same place at the same time.
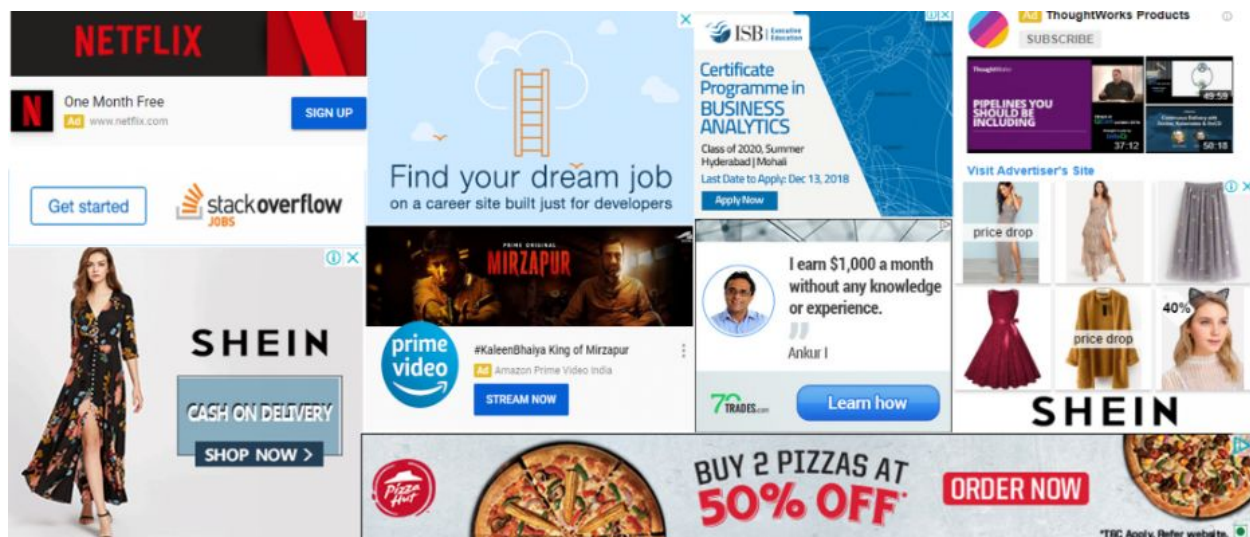
Fig 1.3 Targeted advertising

CHAPTER 2

INFORMATION ABOUT MACHINE LEARNING

2.1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

2.1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction,

and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works
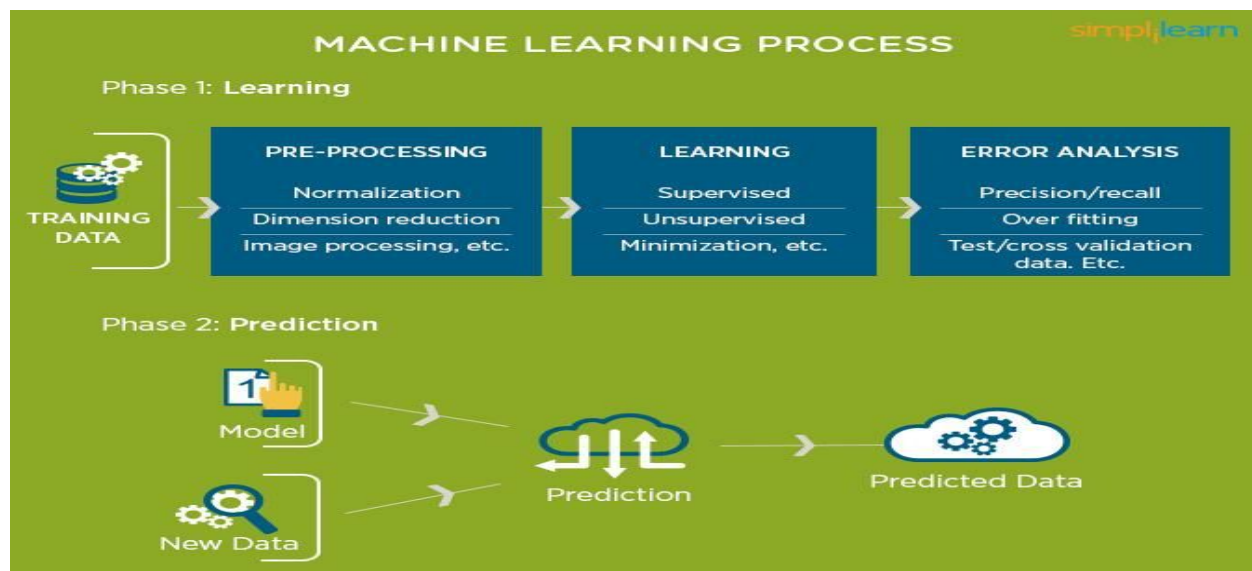
Fig 2.1 : The Process Flow

2.1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data Traditionally, data analysis was always characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

2.1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.
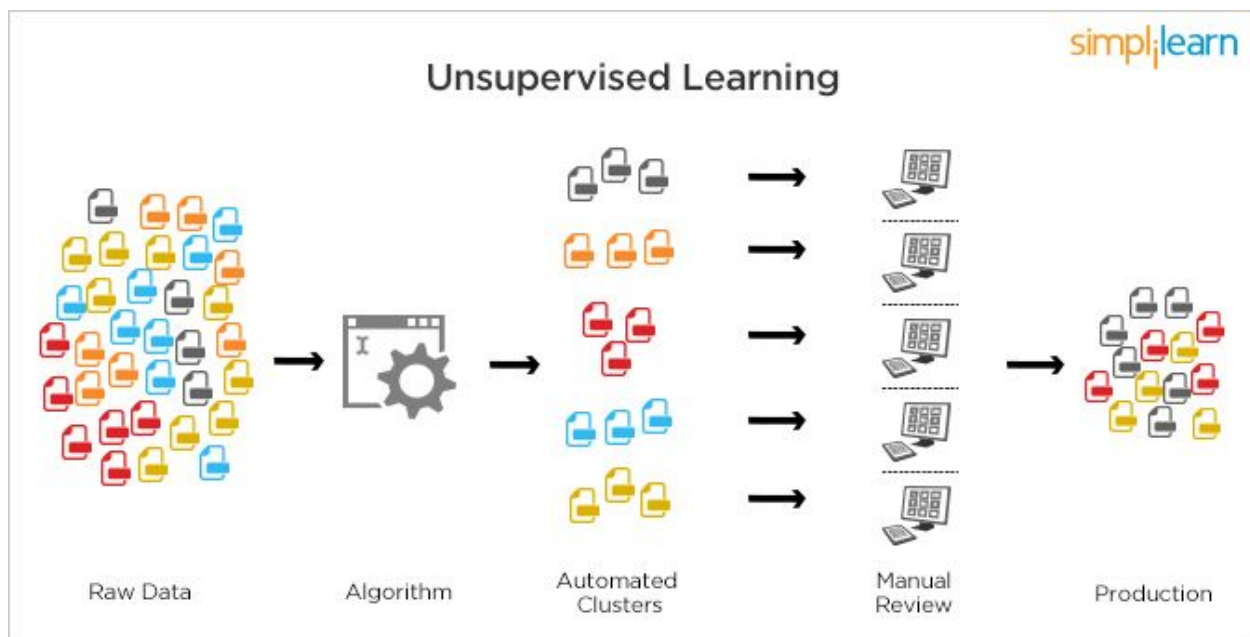
Fig 2.2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.
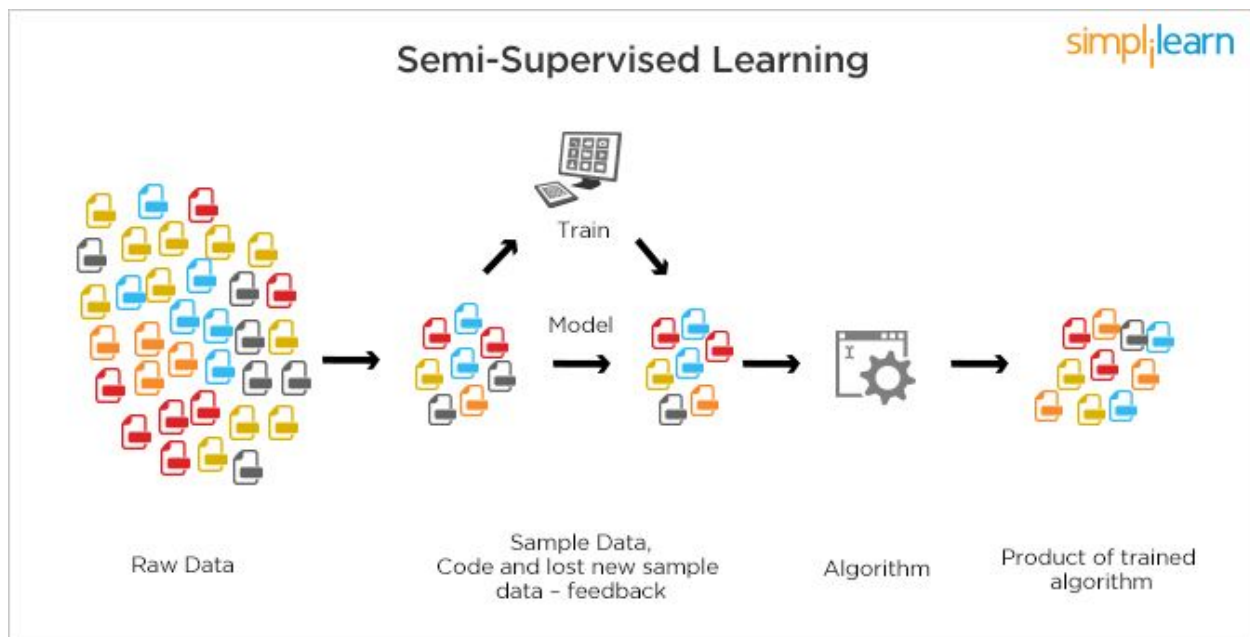
Fig 2.3 : Semi Supervised Learning

2.1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 3

INFORMATION ABOUT PYTHON

Basic programming language used for machine learning is : PYTHON

3.1 INTRODUCTION TO PYTHON:

● Python is a high-level, interpreted, interactive and object-oriented scripting language.

● Python is a general purpose programming language that is often applied in scripting roles

● Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.

● Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.

● Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

HISTORY OF PYTHON:

● Python was developed by GUIDO VAN ROSSUM in early 1990's

● Its latest version is 3.7 , it is generally called as python

3.2 FEATURES OF PYTHON:

● Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.

● Easy-to-read: Python code is more clearly defined and visible to the eyes.

● Easy-to-maintain: Python's source code is fairly easy-to-maintaining.

● A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

● Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

● Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

● Databases: Python provides interfaces to all major commercial databases.

● GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

3.3 HOW TO SETUP PYTHON:

● Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

● The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

Installation(using python IDLE):

● Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.

● Download python from www.python.org

● When the download is completed, double click the file and follow the instructions to install it.

● When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.
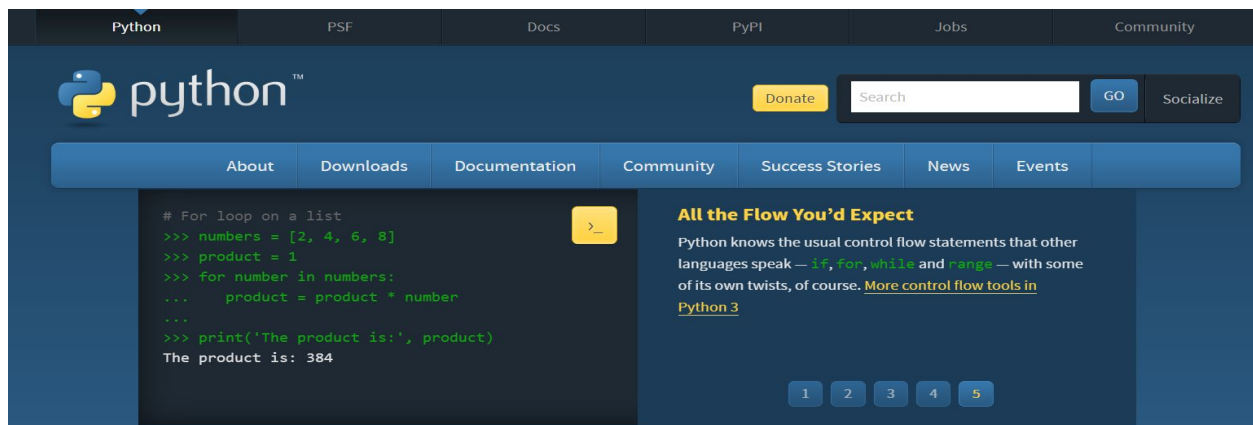
Fig 3.1 : Python download

Installation(using Anaconda):

● Python programs are also executed using Anaconda.

● Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.

● Conda is a package manager that quickly installs and manages packages.

In WINDOWS:

In windows,

● Step 1: Open Anaconda.com/downloads in a web browser.

● Step 2: Download python 3.4 version for (32-bits graphic installer/64 -bit graphic installer)

● Step 3: select installation type(all users)

● Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish

● Step 5: Open jupyter notebook (it opens in default browser).
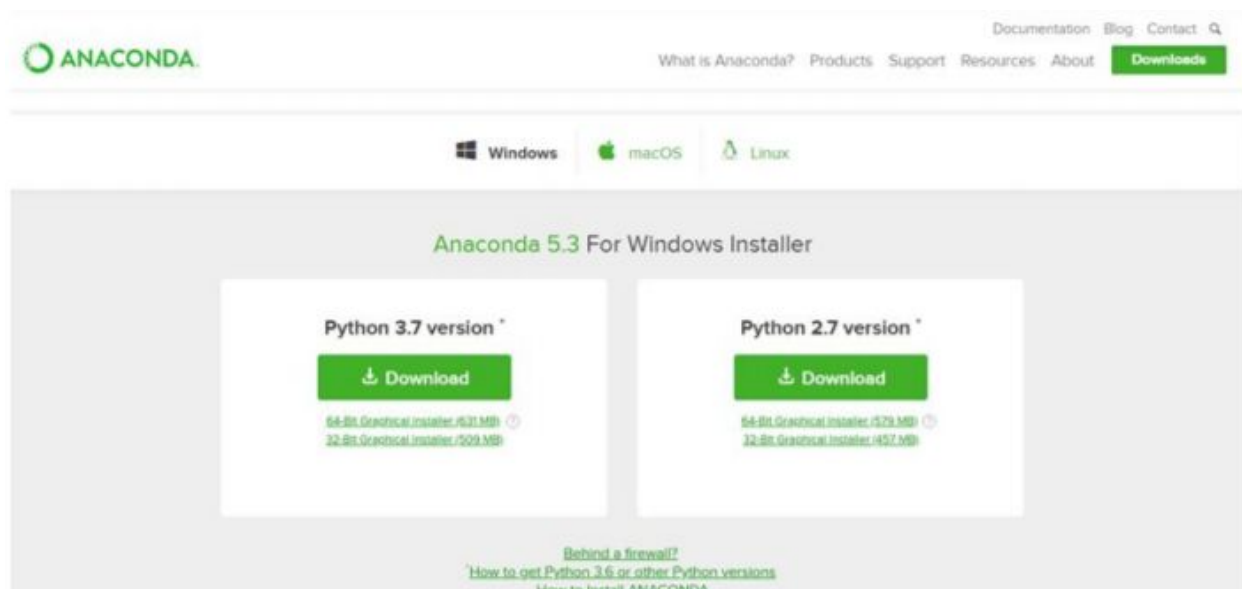
Fig 3.2 : Anaconda download

Fig 3.3 : Jupyter notebook

3.4 PYTHON VARIABLE TYPES:

● Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

● Variables are nothing but reserved memory locations to store values.

● Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

● Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

● Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

● Python has five standard data types –

        o Numbers

        o Strings

        o Lists

        o Tuples

        o Dictionary

Python Numbers:

● Number data types store numeric values. Number objects are created when you assign a value to them.

● Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

Python Strings:

● Strings in Python are identified as a contiguous set of characters represented in the quotation marks.

● Python allows for either pairs of single or double quotes.

● Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

● The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Python Lists:

● Lists are the most versatile of Python's compound data types
. ● A list contains items separated by commas and enclosed within square brackets.

● To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data types.

● The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.

● The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Python Tuples:

● A tuple is another sequence data type that is similar to the list.

● A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

● The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.

● Tuples can be thought of as read-only lists.

● For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

Python Dictionary:

● Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

● Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

● You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.

● What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

3.5 PYTHON FUNCTION:

Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Calling a Function:
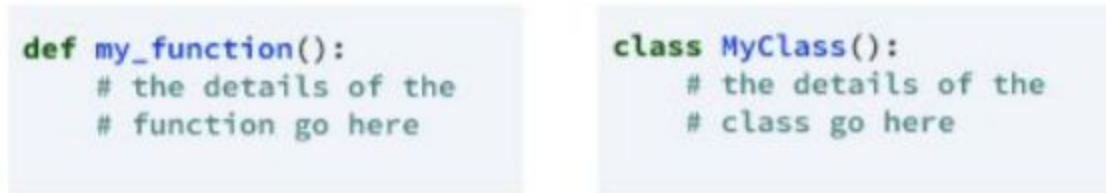
Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

3.6 PYTHON USING OOP's CONCEPTS:

Class:

● Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

● Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

● Data member: A class variable or instance variable that holds data associated with a class and its objects.

● Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

● Defining a Class:
    o We define a class in a very similar way how we define a function.

o Just like a function ,we use parentheses and a colon after the class name(i.e. ():)
when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Fig 3.4 : Defining a Class

__init__ method in Class:

● The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.

● The init method has a special name that starts and ends with two underscores:__init__().

CHAPTER 4

PROJECT NAME(INFORMATION ABOUT THE PROJECT)

4.1 PROJECT REQUIREMENTS

4.1.1 Packages Used

The packages used are:

- pandas

- numpy

- Seaborn

- matplotlib


```
#Importing the required packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Fig 4.1.1 Importing packages

4.1.2 Versions of the Packages

- pandas version 1.0.5

- numpy version 1.18.5

- seaborn version 0.10.1

```
print(pd.__version__)
print(np.__version__)
print(sns.__version__)

1.0.5
1.18.5
0.10.1
```

Fig 4.1.2 Versions

4.1.3 ALGORITHM USED

Long Short-Term Memory (LSTM)

4.2 PROBLEM STATEMENT

The goal of this project is to predict the increase in the confirmed cases using the previous days data. To achieve this, I will be using Long Short-Term memory (LSTM).

4.3 DATASET DESCRIPTION

The dataset used for this project consists of 104 columns, which are described below.

1. Province/State: This tells about the Country/Region belongs to which Province/State.
2. Country/Region: This is the list of countries where the confirmed cases are.
3. Lat : Lat is a shortcut for Latitude, it gives us the latitude of the countries present.
4. Long : Long is a shortcut for Longitude, it provides us the longitude of the countries given.
5. 1/22/20 to 4/30/20 : This tells about the COVID-19 confirmed cases on their respective days.

4.4 OBJECTIVE OF CASE STUDY

To get a better understanding on what is the predicted increase of the COVID-19 confirmed cases by using the previous data present in the dataset.

CHAPTER 5
DATA PREPROCESSING

5.1 READING THE DATASET

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be accessed using the dataframe.

```
#Reading the CSV file from the G-drive
data = pd.read_csv("/content/drive/My Drive/Colab Notebooks/covid19_Confirmed_dataset.csv")
data
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/20	1/
0	NaN	Afghanistan	33.000000	65.000000	0	0	0	0	0	0	0	0	0	
1	NaN	Albania	41.153300	20.168300	0	0	0	0	0	0	0	0	0	
2	NaN	Algeria	28.033900	1.659600	0	0	0	0	0	0	0	0	0	
3	NaN	Andorra	42.506300	1.521800	0	0	0	0	0	0	0	0	0	
4	NaN	Angola	-11.202700	17.873900	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
261	NaN	Western Sahara	24.215500	-12.885800	0	0	0	0	0	0	0	0	0	
262	NaN	Sao Tome and Principe	0.186360	6.613081	0	0	0	0	0	0	0	0	0	
263	NaN	Yemen	15.552727	48.516388	0	0	0	0	0	0	0	0	0	
264	NaN	Comoros	-11.645500	43.333300	0	0	0	0	0	0	0	0	0	
265	NaN	Tajikistan	38.861034	71.276093	0	0	0	0	0	0	0	0	0	

266 rows × 104 columns

Fig 5.1 Reading the dataset

5.2 HANDLING MISSING VALUES AND DUPLICATE VALUES

We can find the number of missing values and duplicate values in each column using isnull() and duplicated() functions respectively. For our dataset no missing values or duplicated values were found.

```
data.shape

(266, 102)

data.isnull().sum()

Province/State      184
Country/Region        0
1/22/20               0
1/23/20               0
1/24/20               0
                    ...
4/26/20               0
4/27/20               0
4/28/20               0
4/29/20               0
4/30/20               0
Length: 102, dtype: int64
```

Fig 5.2.1 : Total number of missing values in each column.

From the above observations we can tell that the dataset for COVID19 doesn't have any missing values.

```
sns.heatmap(data.isnull())
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f160e18ff28>
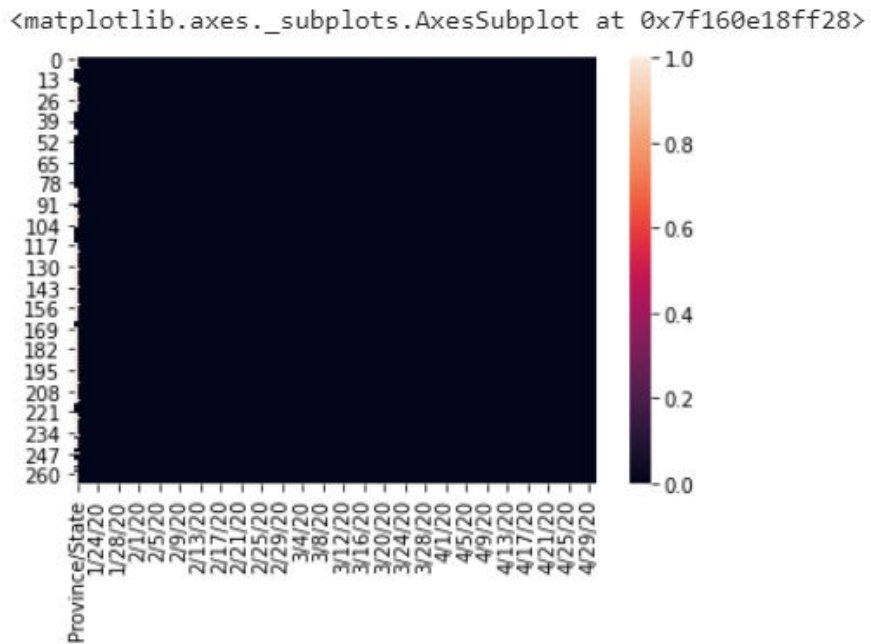
Fig 5.2.2: Visualization through heatmap

5.3 Generating Plots

5.3.1 Visualization of the confirmed cases of one country

```
#Visualizing the increase in the confirmed cases in India
data1.loc['India'].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f160be329b0>
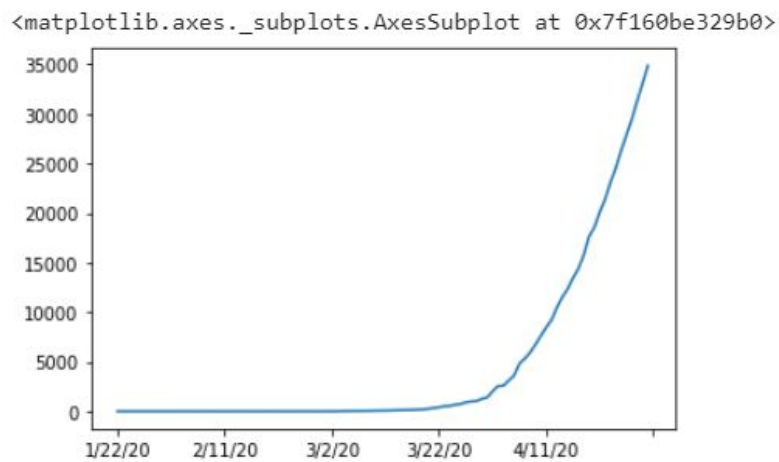
Fig 5.3.1: Visualization Plot of India

From the above plot we can clearly see that there is a constant in the COVID19 confirmed cases upto 3/22/20 , but there is a drastic increase in the cases after 4/11/20.

5.3.2 Visualization of the confirmed cases of two country

```
#Comparing two Country cases
data1.loc['India'].plot()
data1.loc['China'].plot()
plt.xlabel('Dates')
plt.ylabel('No.of cases')
plt.legend()
```

<matplotlib.legend.Legend at 0x7f160bd68898>
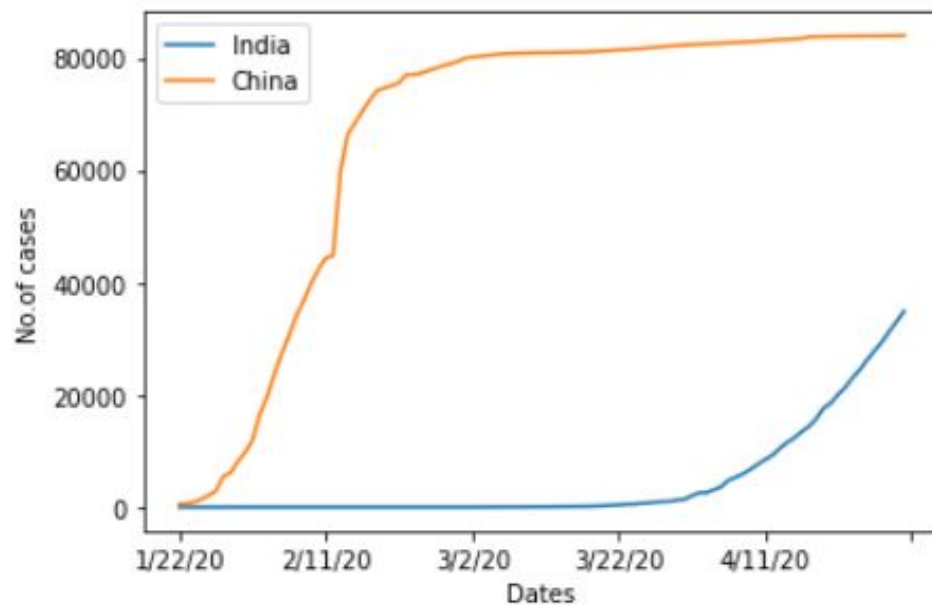
Fig 5.3.1: Visualization Plot between two countries

The plot was drawn between India and China of their respective confirmed COVID19 cases. Through the above plot we can see that since the beginning China has had a rapid increase in cases compared to India . But then China has a constant from 3/22/20 whereas India increases from 4/11/20.

5.3.2 Visualization of the confirmed cases of three country

```python
#Comparing three Country cases
data1.loc['India'].plot()
data1.loc['US'].plot()
data1.loc['Canada'].plot()
plt.xlabel('Dates')
plt.ylabel('No.of cases')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f160bdf5da0>
```
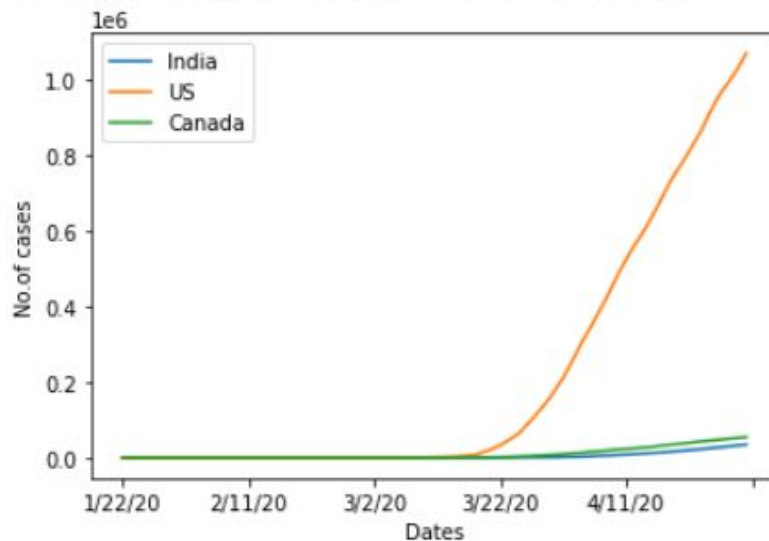
Fig 5.3.1: Visualization Plot between three countries

From the above plot we can see that the US has the highest confirmed COVID-19 cases compared to India and Canada. While comparing between India and Canada , Canada has a slightly higher number of cases than India.

CHAPTER 6
FEATURE SELECTION

6.1 DROP IRRELEVANT FEATURES

For the predictions of the future confirmed cases of COVID-19 , we can drop the columns which are not useful for the predictions. As per the dataset taken we can Lat (latitude), Long (longitude) columns as there is no use in predicting the future cases.

```
#Dropping the irrevalent columns
data = data.drop(['Lat','Long'],axis = 1)
data
```

	Province/State	Country/Region	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/20	1
0	NaN	Afghanistan	0	0	0	0	0	0	0	0	0	
1	NaN	Albania	0	0	0	0	0	0	0	0	0	
2	NaN	Algeria	0	0	0	0	0	0	0	0	0	
3	NaN	Andorra	0	0	0	0	0	0	0	0	0	
4	NaN	Angola	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	
261	NaN	Western Sahara	0	0	0	0	0	0	0	0	0	
262	NaN	Sao Tome and Principe	0	0	0	0	0	0	0	0	0	
263	NaN	Yemen	0	0	0	0	0	0	0	0	0	
264	NaN	Comoros	0	0	0	0	0	0	0	0	0	
265	NaN	Tajikistan	0	0	0	0	0	0	0	0	0	

266 rows × 102 columns

Fig 6.1: Dropping the irrelevant columns

6.2 SELECTING THE DATA(COUNTRY) TO PREDICT THE CASES

To predict the confirmed cases, I selected a specific country which is China. As the figure below I first viewed that there are 33 Provinces/States that belong to China in the dataset.

```
df = data[data['Country/Region']=='China']
df
```

	Province/State	Country/Region	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/3
49	Anhui	China	1	9	15	39	60	70	106	152	
50	Beijing	China	14	22	36	41	68	80	91	111	
51	Chongqing	China	6	9	27	57	75	110	132	147	
52	Fujian	China	1	5	10	18	35	59	80	84	
53	Gansu	China	0	2	2	4	7	14	19	24	
54	Guangdong	China	26	32	53	78	111	151	207	277	
55	Guangxi	China	2	5	23	23	36	46	51	58	
56	Guizhou	China	1	3	3	4	5	7	9	9	
57	Hainan	China	4	5	8	19	22	33	40	43	
58	Hebei	China	1	1	2	8	13	18	33	48	
59	Heilongjiang	China	0	2	4	9	15	21	33	38	
60	Henan	China	5	5	9	32	83	128	168	206	
61	Hong Kong	China	0	2	2	5	8	8	8	10	

Fig 6.2: Selecting China for predicting the confirmed cases.

6.3 Sum the total Provinces/States of China along with cumulating them.

```
df = df.groupby('Country/Region').sum()
df = pd.DataFrame(df)
df
```

	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30
Country/Region									
China	548	643	920	1406	2075	2877	5509	6087	8

1 rows × 100 columns

Fig 6.3.1: Sum the total Provinces/States of China

```
#Using cumsum() --> cumulative sum
df = df.T.cumsum()
df
```

Country/Region	China
1/22/20	548
1/23/20	1191
1/24/20	2111
1/25/20	3517
1/26/20	5592
...	...
4/26/20	6351180
4/27/20	6435098
4/28/20	6519038
4/29/20	6602982
4/30/20	6686938

100 rows × 1 columns

Fig 6.3.2: Cumulative sum

6.4 TRAIN-TEST-SPLIT

```
train_data = df[:len(df)-43]
test_data = df[len(df)-43:]

print(train_data.shape)
print(test_data.shape)
```

```
(43, 1)
(43, 1)
```

Fig 6.4 Train-Test data with their shapes

6.5 FEATURE SCALING

Feature Scaling--> when applied, this units and scaling will be removed To make the data unitless and scaleless, we have to apply Feature Scaling. Here i am using MinMaxScaler from sklearn library

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(train_data)
scaled_train_data = scaler.transform(train_data)
scaled_test_data = scaler.transform(test_data)
scaled_train_data
```

```
array([[0.        ],
       [0.01023925],
       [0.02165786],
       [0.03398165],
       [0.04731474],
       [0.06149309],
       [0.07635167],
       [0.09133512],
       [0.11138546],
       [0.13359935],
       [0.15650117],
       [0.18010598],
       [0.20435387],
       [0.22919661],
       [0.25417594],
       [0.27930859],
       [0.30459958],
       [0.3303763 ],
       [0.35616005],
       [0.38201712],
```

Fig 6.5: Feature scaling the data

CHAPTER 7
MODEL BUILDING AND EVALUATION

7.1 LONG SHORT-TERM MEMORY

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.
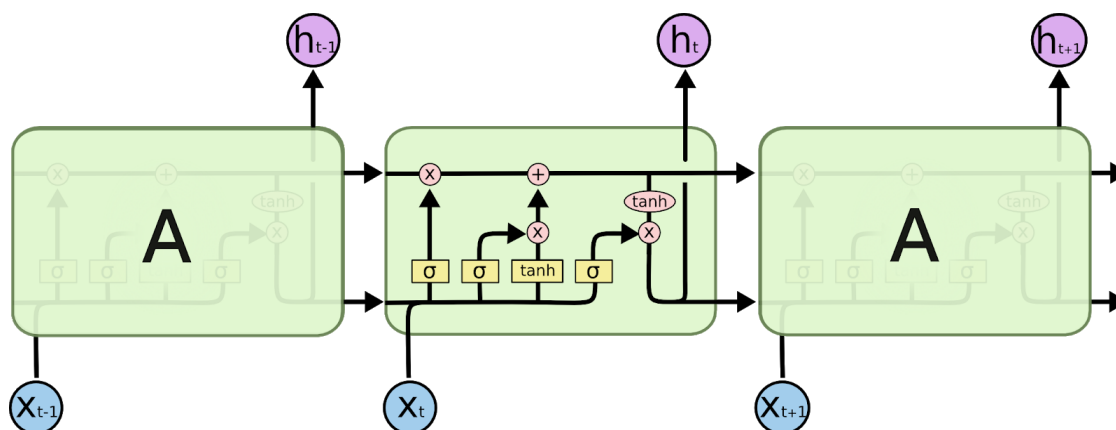
Fig 7.1.1: The repeating module in an LSTM contains four interacting layers.

There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates.

```
df.columns = ['China']
print("len of the dataset::"+str(len(df)))
data = np.array(df).reshape(-1,1)
train_data = df[:len(df)-43]
test_data = df[len(df)-43:]

print(train_data.shape)
print(test_data.shape)

len of the dataset::86
```

Fig 7.1: Dividing the training and testing data for predicting

As the length of the dataset is 86 , I have decided to divide the train and test data as shown in the above figure.

7.2 TRAIN THE MODELS

➔ **TensorFlow :** The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

➔ **Sequential Model** : A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

➔ **Dense Layers :** A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

➔ **LSTM Layers** : An LSTM network is a recurrent neural network that has LSTM cell blocks in place of our standard neural network layers.

➔ **Dropout :** Dropout is a technique used to prevent a model from overfitting.

➔ **TimeseriesGenerator :** The Keras deep learning library provides the TimeseriesGenerator to automatically transform both univariate and multivariate time series data into samples, ready to train deep learning models.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout

from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
n_input =3
n_features =1

generator = TimeseriesGenerator(scaled_train_data,scaled_train_data, length=n_input, batch_size=1)

lstm_model = Sequential()
lstm_model.add(LSTM(43, activation='linear', input_shape = (n_input, n_features), recurrent_dropout = 0.2))
lstm_model.add(Dropout(0.3))
lstm_model.add(Dense(1))
lstm_model.summary()
```

```
Model: "sequential"

_____
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 43)                7740

_____
dropout (Dropout)            (None, 43)                0

_____
dense (Dense)                (None, 1)                 44
=================================================================
Total params: 7,784
Trainable params: 7,784
Non-trainable params: 0
```

Fig 7.1.2: Applying the LSTM algorithm to the scaled model

➔ **Compile :** Compile defines the loss function, the optimizer and the metrics.

➔ **fit_generator** : The .fit_generator function accepts the batch of data, performs backpropagation, and updates the weights in our model.

```
lstm_model.compile(optimizer='adam', loss='mse')
```

```
lstm_model.fit_generator(generator, epochs=100)
```

```
WARNING:tensorflow:From <ipython-input-25-42c90f30c883>:1: Model.fit_generator (fro
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/100
40/40 [==============================] - 0s 3ms/step - loss: 0.2192
Epoch 2/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0463
Epoch 3/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0192
Epoch 4/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0121
Epoch 5/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0142
Epoch 6/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0132
Epoch 7/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0067
Epoch 8/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0068
Epoch 9/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0075
Epoch 10/100
```

Fig 7.2.1: Generating the epoch

7.3 VALIDATE THE MODEL

→ **History. history :** The History. history attribute is a dictionary recording training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values .

```
losses_lstm = lstm_model.history.history['loss']
plt.figure(figsize = (12,4))
plt.xticks(np.arange(0,21,1))
plt.plot(range(len(losses_lstm)), losses_lstm)
```
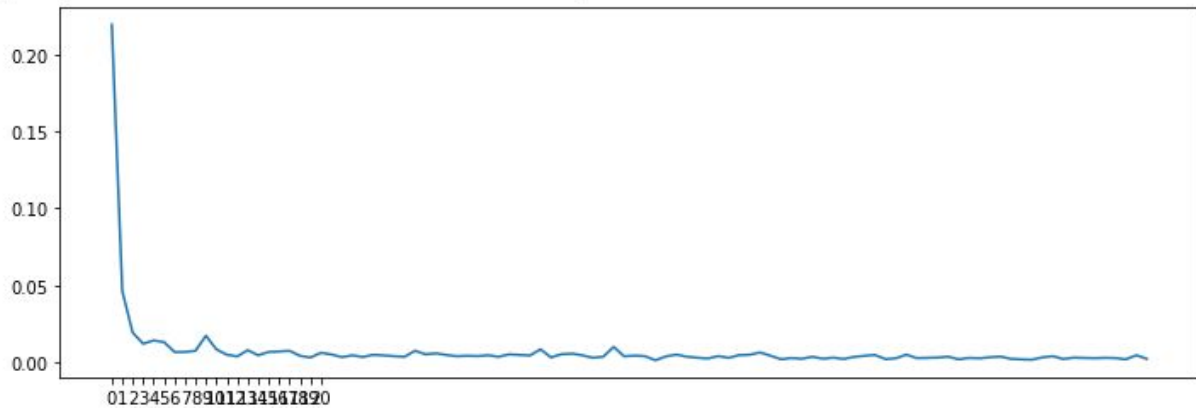
[<matplotlib.lines.Line2D at 0x7f16402f29e8>]

Fig 7.2.2 Plotting the losses_lstm

7.4 MAKE PREDICTIONS

```
lstm_predictions_scaled = []

batch = scaled_train_data[-n_input:]
current_batch = batch.reshape((1, n_input, n_features))

for i in range(len(test_data)):
    lstm_pred = lstm_model.predict(current_batch)[0]
    lstm_predictions_scaled.append(lstm_pred)
    current_batch = np.append(current_batch[:,1:,:],[[lstm_pred]],axis=1)
lstm_predictions_scaled
```

```
[array([1.0310249], dtype=float32),
 array([1.0590856], dtype=float32),
 array([1.0878091], dtype=float32),
 array([1.1170188], dtype=float32),
 array([1.145461], dtype=float32),
 array([1.1741046], dtype=float32),
 array([1.2026663], dtype=float32),
 array([1.2309678], dtype=float32),
 array([1.2592185], dtype=float32),
 array([1.2873015], dtype=float32),
 array([1.3151997], dtype=float32),
 array([1.3429428], dtype=float32),
 array([1.37049], dtype=float32),
 array([1.3978366], dtype=float32),
 array([1.4249777], dtype=float32),
```

Fig 7.4.1: Predicting the values

```
lstm_predictions = scaler.inverse_transform(lstm_predictions_scaled)
lstm_predictions
```

```
array([[3217300.61019135],
       [3301124.29937792],
       [3386927.93551159],
       [3474184.12624907],
       [3559147.35324693],
       [3644712.39865136],
       [3730032.79953194],
       [3814575.82196188],
       [3898967.1434207 ],
       [3982857.78045368],
       [4066196.09775376],
       [4149071.12171245],
       [4231360.70812058],
       [4313051.32496667],
       [4394128.01581693],
       [4474540.56978655].
```

Fig 7.4.2: Inversing the predictions

7.5 PREDICTIONS COMPARED TO THE RAW DATA

Here, we are adding the predicted LSTM predictions to the test data for comparing it with the raw data.

```
#Adding the LSTM_Predictions for test_data
test_data['LSTM_Predictions'] = lstm_predictions
test_data
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

	China	LSTM_Predictions
3/19/20	3205778	3.217301e+06
3/20/20	3287028	3.301124e+06
3/21/20	3368333	3.386928e+06
3/22/20	3449768	3.474184e+06
3/23/20	3531266	3.559147e+06
3/24/20	3612857	3.644712e+06
3/25/20	3694518	3.730033e+06
3/26/20	3776300	3.814576e+06

Fig 7.5.1: Adding the predicted values to test data

```
test_data.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f160ec8a390>
```
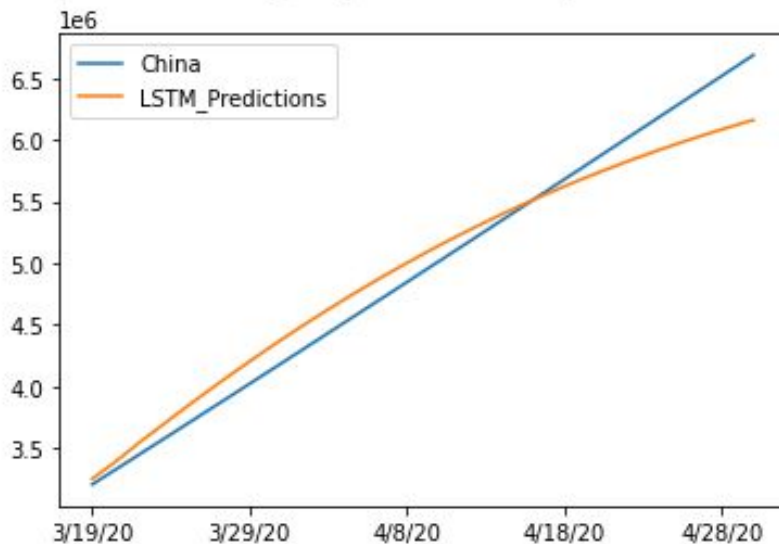

Fig 7.5.2: Plot between LSTM_Predictions and China

7.6 CALCULATING THE ERROR RATE

➔ **Mean Squared Error (MSE) :** MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.

➔ **Mean Absolute Error (MAE) :** MAE is a measure of errors between paired observations expressing the same phenomenon.The mean absolute error is one of a number of ways of comparing forecasts with their eventual outcomes.

```
#Calculating the mean absolute error and mean squared error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

print('MAE of LSTM Model ',mean_absolute_error(test_data['China'], test_data['LSTM_Predictions']))

print('MSE of LSTM Model ',mean_squared_error(test_data['China'], test_data['LSTM_Predictions']))
```

```
MAE of LSTM Model  89060.94445947159
MSE of LSTM Model  17598228703.872044
```

Fig 7.6: Calculating MAE AND MSE