

**Data Analysis Project On Covid-19**

**Presented by -**

* + - **Sejal Kankariya**
* **TABLE OF CONTENT :**

|  |  |
| --- | --- |
| **1.** | Table of figures |
| **2.** | Abstract |
| **3.** | Chapters |
| **4.** | Complete Code  (.ipynb file is attached) |
| **5.** | Conclusion |

* **TABLE OF FIGURES / GRAPHS:**

1. **STATEWISE COVID-19 DATA IN INDIA**

**Figure 1…………………………………………………………**

**Figure 2…………………………………………………………**

**Figure 3…………………………………………………………**

1. **OVERALL COVID-19 STATUS IN INDIA**

**Figure 1…………………………………………………………**

1. **VISUALIZING THE SPREADS GEOGRAPHICALLY**

**Map 1…………………………………………………………..**

**Map 2…………………………………………………………..**

**Map 3…………………………………………………………..**

1. **TREND OF VIRUS**

**Graph 1………………………………………………………...**

**Graph 2………………………………………………………...**

**Graph 3………………………………………………………...**

**Graph 4………………………………………………………...**

1. **FORECAST USING ‘ fbprophet ’**

**Graph 1………………………………………………………...**

**Graph 2………………………………………………………...**

**Graph 3………………………………………………………...**

* **ABSTRACT**

We are creating a Data Analysis (Data Science) project on spread of corona virus India. The raw data of this project are .csv and .xls files and transform it into a Data Analysis. This project is actually an attempt of analyzing the coronavirus spread in India with the help of data science and data analytics with python language .This analysis helps us to find the major Covid-19 affected states of India and all that.

The data used for this project is split across two files –

1. Covid\_19\_india.csv
2. Indian\_Coordinates.xls

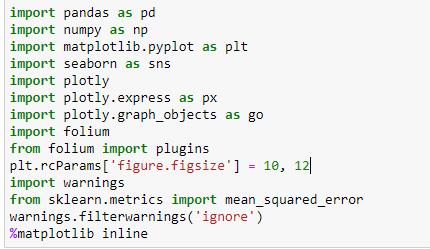
In that analysis we use some python libraries such as numpy , pandas, plotly, seaborn, matplotlib, folium, warning, fbprophet and sklearn.

In this project we import the files and libraries for explore the parameters in both files .We filter off the unwanted data entries,thus making our data perfect for analysis. Then we merge the data from datasets into dataframes for further analysis. We also do a state-wise and over all corona analysis for the states of India. Pie-charts and sub plots are used for that.

We use folium library for visualizing the spreads of Corona virus geographically and by using Indian\_Coordinates.xls file we map them on the world map. We use fbprophet and predict the future prospectus by analyzing the existing data. Then we compare the result of analysis with the real data.

**CHAPTERS**

**Importing data visualization libraries in python**

****

* import pandas as pd -

pandas (all lowercase) is a popular Python-based data analysis toolkit which can be imported using import pandas as pd. It presents a diverse range of utilities, ranging from parsing multiple file formats to converting an entire data table into a NumPy matrix array.

* import numpy as np -

NumPy is a Python package that stands for ‘Numerical Python’. It is the core library for scientific computing, which contains a powerful n-dimensional array object. It is imported using import numpy as np. numpy is one of the most powerful Python libraries .

* import matplotlib.pyplot as plt –

matplotlib is the library in python for data visualization. Pyplot is the module within the matplotlib.plt is use as an object. It is imported using import matplotlib.pyplot as plt .

* import seaborn as sns –

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. It is imported using import seaborn as sns .

* import plotly

import plotly.express as px

import plotly.graph\_object as go -

plotly enables Python users to create beautiful interactive web-based visualizations that can be displayed in Jupyter notebooks.

Plotly Express is a built-in part of the plotly library, and is the recommended starting point for creating most common figures. Every Plotly Express function uses graph objects internally and returns a plotly.graph\_objects.

* import folium

from folium import plugins –

Folium is a powerful Python library that helps you create several types of Leaflet maps.It creates an inline map containing an interactive map with markers based on the resulting dataset. The bigger the zoom number, the closer in you get from folium import plugins.

* import warnings –

The warning module is actually a subclass of Exception which is a built-in class in Python. Warnings are provided to warn the developer of situations that aren’t necessarily exceptions. Usually, a warning occurs when there is some obsolete of certain programming elements.

* from sklearn.metrics import mean\_squared\_error -

The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

* warnings.filterwarnings('ignore')-

The determination whether to issue a warning message is controlled by the warning filter, which is a sequence of matching rules and actions. Rules can be added to the filter by calling filterwarnings().

* %matplotlib inline-

%matplotlib inline sets the backend of matplotlib to the 'inline' backend:

* df\_India= pd.read\_csv(r'covid\_19\_india.csv') –

we will import the different data tables from where I sourced my information.

* India\_coord = pd.read\_excel(r'Indian Coordinates.xlsx')-

We use pandas read\_excel() to read an Excel file into a pandas DataFrame.

* print(df\_India.info())-

It print full information.

Output – <class 'pandas.core.frame.DataFrame'>

RangeIndex: 2450 entries, 0 to 2449

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Sno 2450 non-null int64

1 Date 2450 non-null object

2 Time 2450 non-null object

3 State/UnionTerritory 2450 non-null object

4 ConfirmedIndianNational 2450 non-null object

5 ConfirmedForeignNational 2450 non-null object

6 Cured 2450 non-null int64

7 Deaths 2450 non-null int64

8 Confirmed 2450 non-null int64

dtypes: int64(4), object(5)

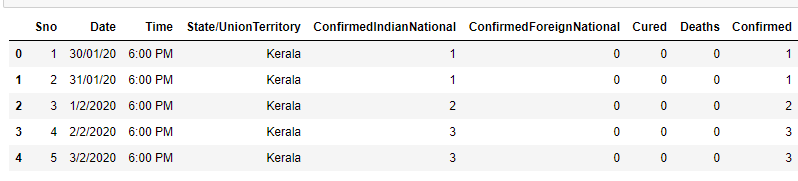
memory usage: 172.4+ KB

None

* df\_India.head() -

The head() returns the first n rows for the object based on position.

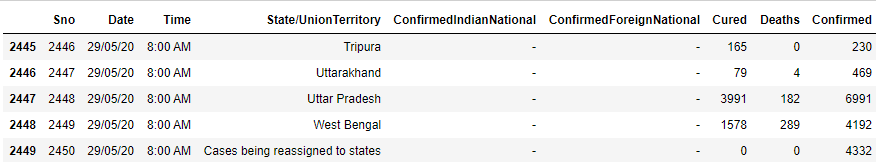
Output –



* df\_India.tail() –

The tail() returns the last n rows for the object based on position.

Output-



* df\_India.dtypes -

Pandas DataFrame.dtypes attribute return the dtypes in the DataFrame. It returns a Series with the data type of each column.

Output –

Sno int64

Date object

Time object

State/UnionTerritory object

ConfirmedIndianNational object

ConfirmedForeignNational object

Cured int64

Deaths int64

Confirmed int64

dtype: object

* print(India\_coord.info())-

Print the information of India\_coord file.

Output-

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 35 entries, 0 to 34

Data columns (total 3 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Name of State / UT 35 non-null object

1 Latitude 35 non-null float64

2 Longitude 35 non-null float64

dtypes: float64(2), object(1)

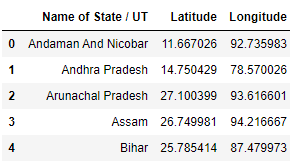
memory usage: 968.0+ bytes

None

* India\_coord.head()-

The head() returns the first n rows for the object based on position.

Output-



**Data Cleaning –**

* def replace\_dash\_with\_zeros(inp):

return int(inp.replace("-","0"))

# This code replaces all the dashes present in our dataset with the zeros.

df\_India.drop(['Sno'],axis=1,inplace=True)

#It drops the ‘Sno’ column present in dataset.

df\_India['Date'] =pd.to\_datetime(df\_India['Date'],format = "%d/%m/%y")

# convert object to date in pandas dataframe · python pandas format row datetime to column.The date()function display a formatted local time/date

df\_India['ConfirmedIndianNational'] = df\_India['ConfirmedIndianNational'].apply(replace\_dash\_with\_zeros)

df\_India['ConfirmedForeignNational'] = df\_India['ConfirmedForeignNational'].apply(replace\_dash\_with\_zeros)

df\_India.sort\_values("Confirmed", ascending = False, inplace = True)

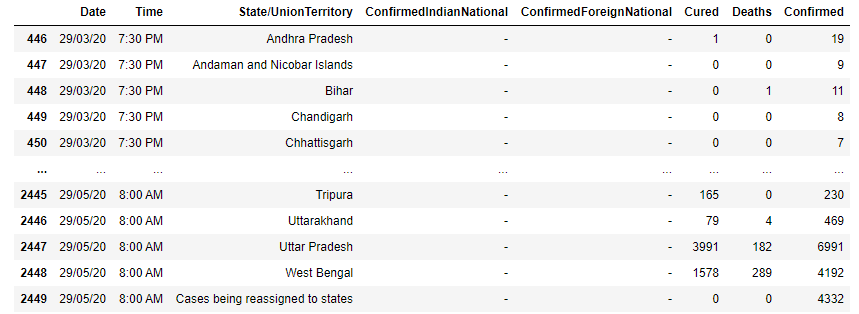
df\_India

#we replace all the dashes with zeros from the 'ConfirmedIndianNational'

and 'ConfirmedForeignNational' Column and then store the values in that column then we sort the values in ascending order. Then gives the value.

* df\_India.loc[df\_India["ConfirmedForeignNational"] == "-",:]

#This checks whether the ConfirmedForeignNationalcolumn is having

similar values or not. loc is used to accsess a group of rows or columns.

* list(zip(df\_India.columns,df\_India.dtypes,df\_India.isna().sum()))

#Creating zip file to combine it and put all data in a list and checks if

there is any null value is present or not.

Output-

[('Date', dtype('O'), 0),

('Time', dtype('O'), 0),

('State/UnionTerritory', dtype('O'), 0),

('ConfirmedIndianNational', dtype('O'), 0),

('ConfirmedForeignNational', dtype('O'), 0),

('Cured', dtype('int64'), 0),

('Deaths', dtype('int64'), 0),

('Confirmed', dtype('int64'), 0)]

* print(f'We have data available from : {df\_India.Date.min()} to {df\_India.Date.max()}')

# To know when the dataset is latest updated.

Output-

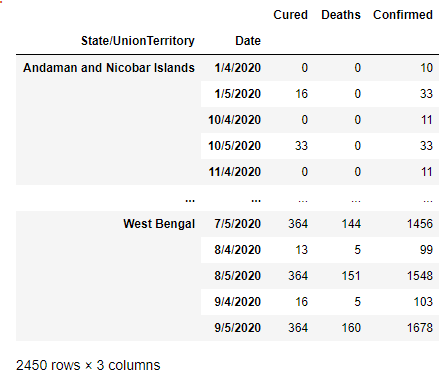
We have data available from : 1/2/2020 to 9/5/2020

* df\_India.groupby(["State/UnionTerritory", "Date"]).sum()

# groupby() function is used to split the data into groups based on some

Criteria(State/UnionTerritory).

Output-



* States = df\_India['State/UnionTerritory'].unique().tolist()

States

# unique functions gives unique values of column.tolist() method convert

Given data into list datatype.

Output-

['Kerala',

'Telengana',

'Delhi',

'Rajasthan',

'Uttar Pradesh',

'Haryana',

'Ladakh',

'Tamil Nadu',

'Karnataka',

'Maharashtra',

'Punjab',

'Jammu and Kashmir',

'Andhra Pradesh',

'Uttarakhand',

'Odisha',

'Puducherry',

'West Bengal',

'Chhattisgarh',

'Chandigarh',

'Gujarat',

'Himachal Pradesh',

'Madhya Pradesh',

'Bihar',

'Manipur',

'Mizoram',

'Andaman and Nicobar Islands',

'Goa',

'Unassigned',

'Assam',

'Jharkhand',

'Arunachal Pradesh',

'Tripura',

'Nagaland',

'Meghalaya',

'Dadar Nagar Haveli',

'Cases being reassigned to states',

'Sikkim']

* States.remove("Cases being reassigned to states")

#Removing the ‘Cases being reassigned to states’ field from state column

States.remove("Unassigned")

#Removing the ‘Unassigned’ field from state column.

State

#printing the all states.

Output-

['Maharashtra',

'Tamil Nadu',

'Delhi',

'Gujarat',

'Rajasthan',

'Madhya Pradesh',

'Uttar Pradesh',

'West Bengal',

'Andhra Pradesh',

'Bihar',

'Karnataka',

'Punjab',

'Telengana',

'Jammu and Kashmir',

'Odisha',

'Haryana',

'Kerala',

'Assam',

'Uttarakhand',

'Jharkhand',

'Chhattisgarh',

'Chandigarh',

'Himachal Pradesh',

'Tripura',

'Goa',

'Ladakh',

'Puducherry',

'Manipur',

'Andaman and Nicobar Islands',

'Meghalaya',

'Nagaland',

'Arunachal Pradesh',

'Dadar Nagar Haveli',

'Sikkim',

'Mizoram']

* len(States)

#len() function gives the total number of states.

Output- 35

**Merging Data Frames –**

* df\_final\_India = pd.DataFrame()

#here we use dataframe method of pandas A Data frame is a two-

dimensional data structure.

* dates = pd.DataFrame({"Date": pd.date\_range(df\_India.Date.min(),df\_India.Date.max())})

#pandas.date\_range() is one of the general functions in Pandas which is

used to return a fixed frequency DatetimeIndex.

* all\_dates\_df=pd.merge(dates,df\_India.loc[df\_India['State/UnionTerritory'] == state,:], on = "Date",how = "left")

all\_dates\_df['State/UnionTerritory'] = state

#If for a state at any date no values are available to replace it with 0.

* all\_dates\_df = all\_dates\_df.fillna(0)

all\_dates\_df['New Cases'] = all\_dates\_df['Confirmed'] - all\_dates\_df

['Confirmed'].shift(1)

#Adding new column in the dataframe.gives old cases – new cases.

* df\_final\_India = pd.concat([df\_final\_India, all\_dates\_df],axis = 0)

#Concatenate df\_final\_India, all\_dates\_df

* print("Finally we have a data of Size: ",df\_final\_India.shape)

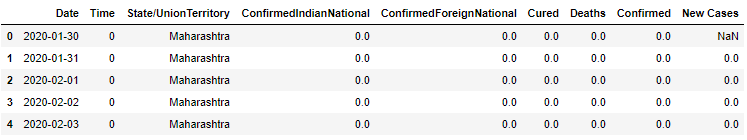
# Gives the shape of new updated data.

Output - Finally we have a data of Size: (4235, 9)

* df\_final\_India.head()

# gives the first n entries of updated dataset.

Output-



* df\_final\_India.dropna(inplace = True)

# dropna() method removes missing values.

* df\_final\_India.shape

# It returns the shape of dataset.

After dropping missing values.

Output – (4200, 9)

* del df\_final\_India['Time']

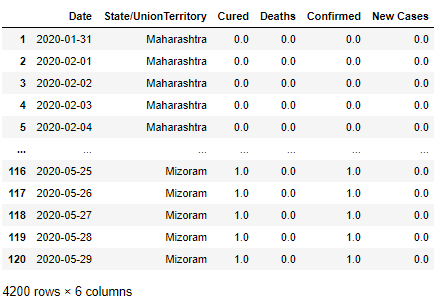
del df\_final\_India['ConfirmedIndianNational']

del df\_final\_India['ConfirmedForeignNational']

df\_final\_India

# Delete 'Time' , 'ConfirmedIndianNational' , 'ConfirmedForeignNational' this three columns from our dataset.

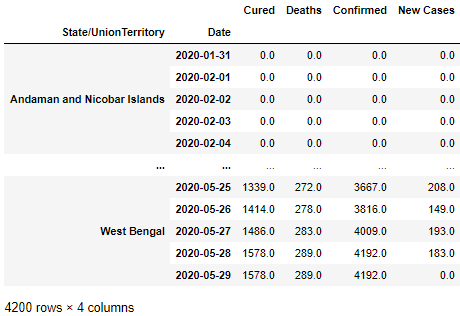
Output –



* df\_final\_India.groupby(["State/UnionTerritory", "Date"]).sum()

# It computes the sum of all group (“State/UnionTerritory", "Date”)

values. Output –



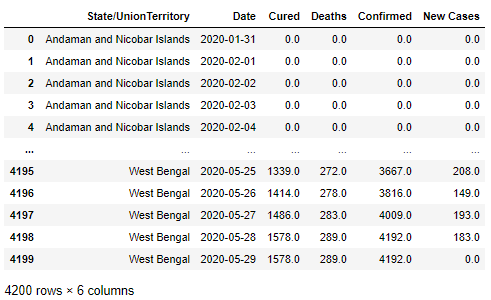
* df\_final\_India =df\_final\_India.groupby(["State/UnionTerritory", "Date"]).sum().reset\_index()

df\_final\_India

# Compute the sum of (["State/UnionTerritory", "Date"]) this group

values and reset the index of it. prints the dataset.

Output-



**Statewise Covid19 Status in India-**

* def plot\_pie(active,cured,death,title):

labels = ['Active','Recovered','Died']

sizes = [active,cured,death]

color= ['#66b3ff','green','red']

explode = []

# pie() function of plotly is used for drawing the pie charts.

**explode** : array-like, optional, default: None

If not None, is a len(x) array which specifies the fraction of the radius with which to offset each wedge.

**labels** : list, optional, default: None

A sequence of strings providing the labels for each wedge

**colors** : array-like, optional, default: None

A sequence of matplotlib color args through which the pie chart will cycle. If None, will use the colors in the currently active cycle.

for i in labels:

explode.append(0.05)

plt.figure(figsize= (15,6))

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=9,explode = explode,colors = color)

centre\_circle = plt.Circle((0,0),0.70,fc='white')

#**autopct** : None (default), string, or function, optional

If not None, is a string or function used to label the wedges with their numeric value. The label will be placed inside the wedge. If it is a format string, the label will be fmt%pct. If it is a function, it will be called.

**startangle** : float, optional, default: None

If not None, rotates the start of the pie chart by angle degrees counterclockwise from the x-axis.

**radius** : float, optional, default: None

The radius of the pie, if radius is None it will be set to 1.

The append() method inserts specified content at the end of the selected elements.

fig = plt.gcf()

fig.gca().add\_artist(centre\_circle)

plt.title(title + 'COVID-19 Cases',fontsize = 20)

plt.axis('equal')

plt.tight\_layout()

#**ply.gcf()** is primarily used to **get the current figure**. If no current figure is available then one is created with the help of the figure() function.

Because **fig.gca()** holds the last created axis in the figure, so the circle is placed there three times independently of the loop variables.

**axis('equal')** sets both axes to be equal. **tight\_layout()** is for getting equal and perfect output.

* total\_cases\_india = 0

cured\_cases\_india = 0

death\_cases\_india = 0

active\_cases\_india = 0

state\_df = pd.DataFrame()

for state in States:

one\_state\_df = df\_final\_India.loc[df\_final\_India['State/UnionTerritory'= = state,:]

m=one\_state\_df.iloc[-1,:]

z=pd.DataFrame(m).T

state\_df = pd.concat([state\_df,z],axis = 0)

total\_cases = one\_state\_df['Confirmed'].values[-1]

cured = one\_state\_df['Cured'].values[-1]

deaths = one\_state\_df['Deaths'].values[-1]

active = total\_cases - cured - deaths

plot\_pie(active, cured, deaths,state)

total\_cases\_india += total\_cases

cured\_cases\_india += cured

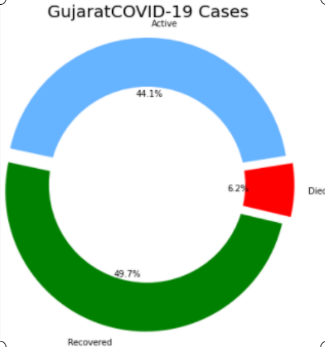
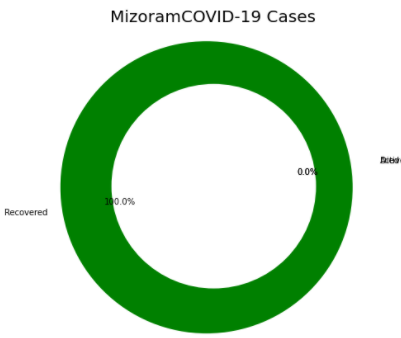
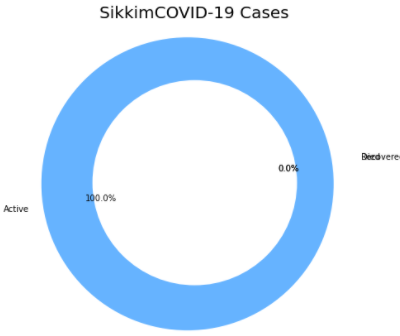
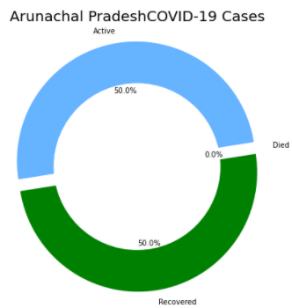
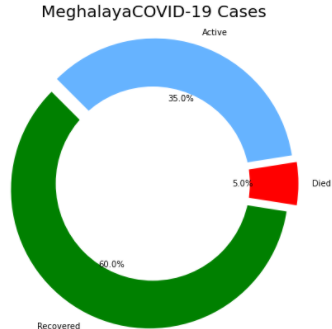
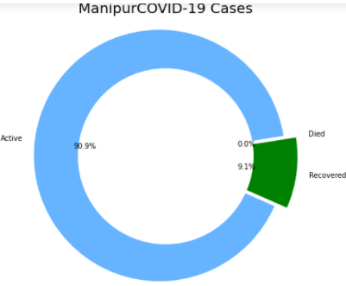
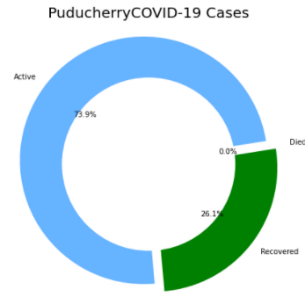
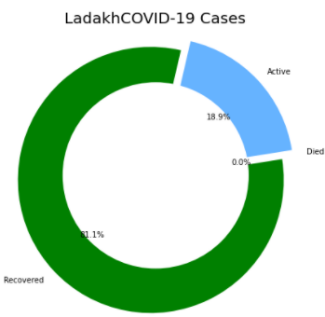
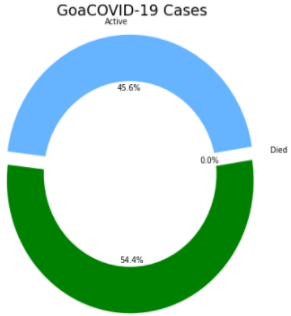
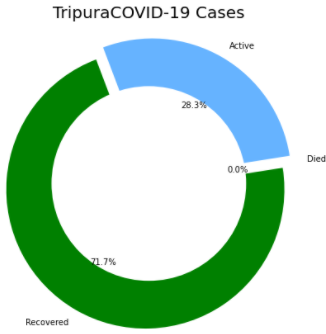
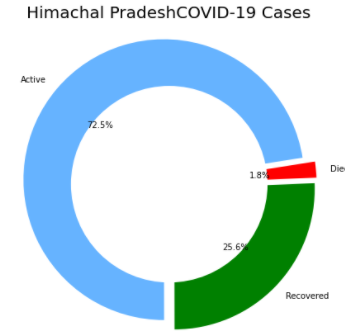
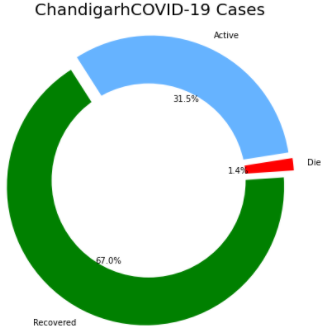
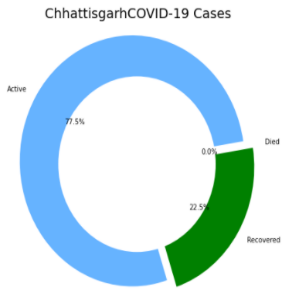
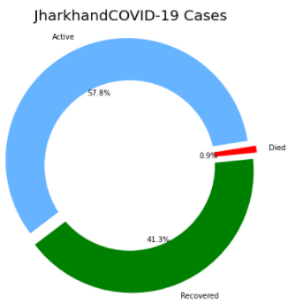
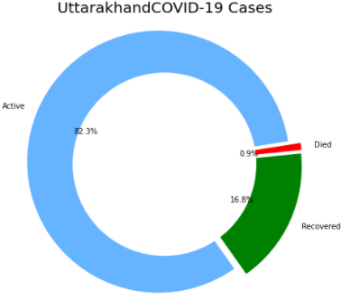
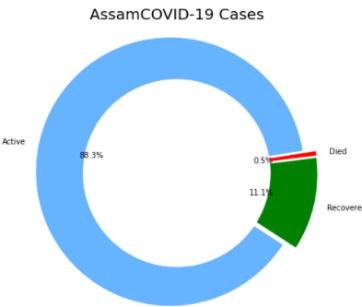
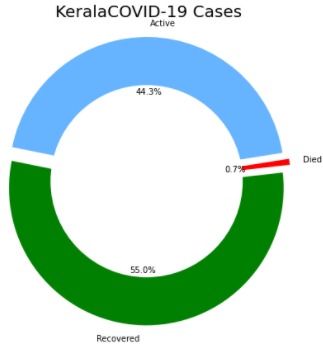
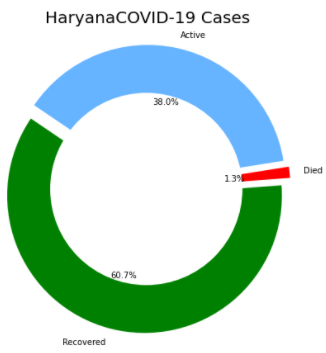
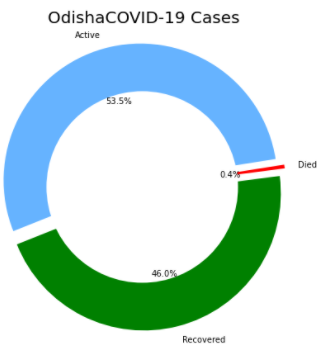
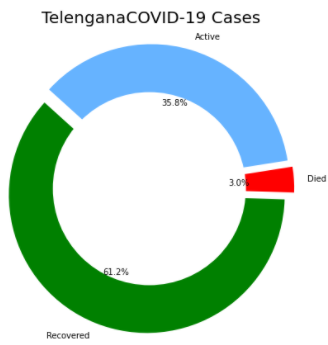
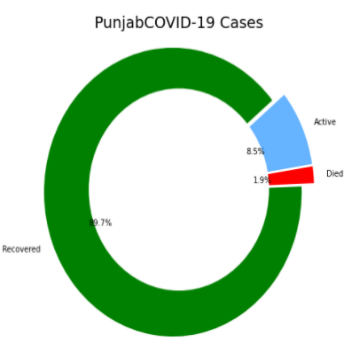
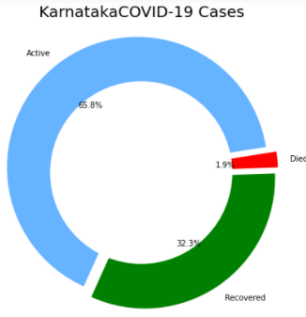
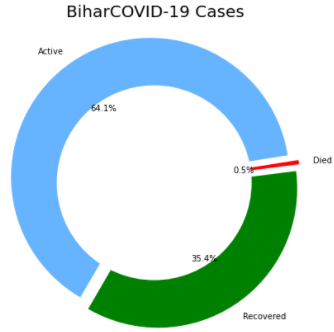
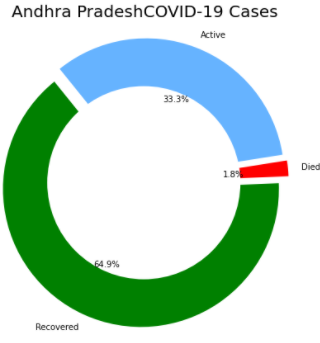
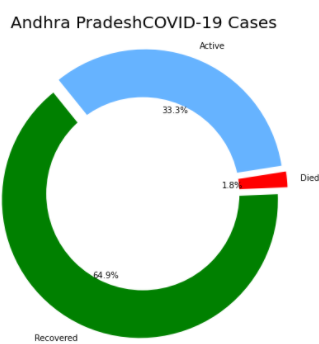
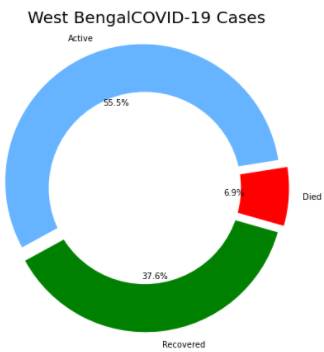
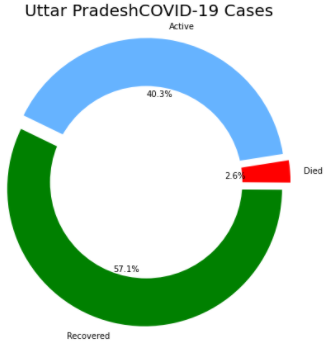
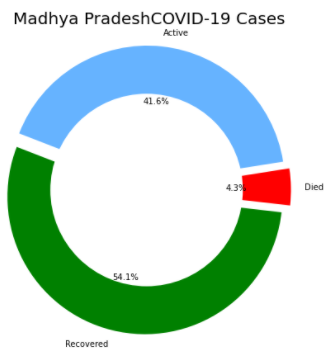
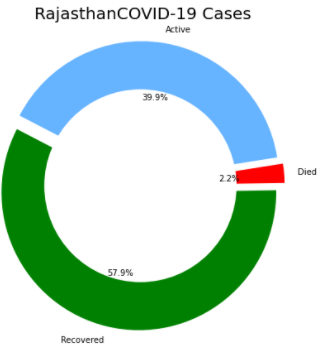
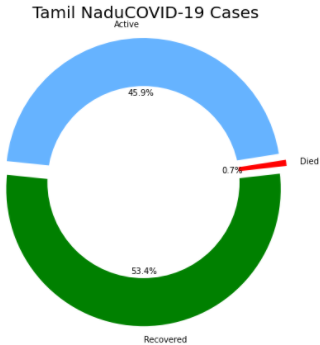
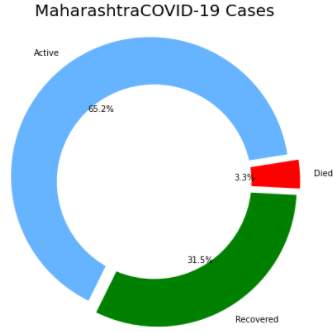
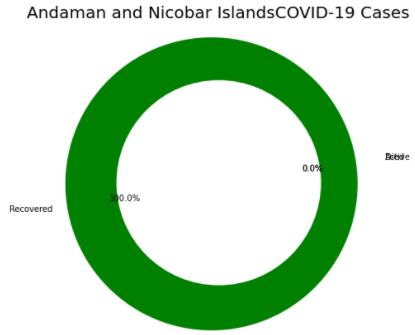
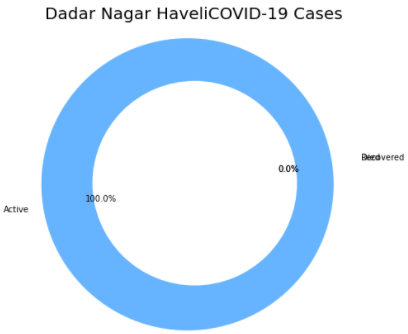
death\_cases\_india += deaths

active\_cases\_india += active

#Initially we take the value of total\_cases\_india=0, cured\_cases\_india 0 , death\_cases\_india = 0, active\_cases\_india = 0 . Then we take for loop for states . We access groupof columns i.e 'State/UnionTerritory' by loc()

Function. Also we concatenatesome values. Here we use the the logic for for loop is that if person is cured then total\_cases-1 And At last we increments count of all three cases.plot the pieplot .

Output-

* state\_df.reset\_index(inplace = True,drop = True)

state\_df

#.reset\_index() to reset the index to the default integer index beginning at 0.we can simply use the reset index() fuction.

Output-

| **State/UnionTerritory** | **Date** |  | **Cured** | **Deaths** | **Confirmed** | **New Cases** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | Maharashtra |  | 2020-05-29 | 17918 | 1897 | 56948 | 0 |
| **1** | Tamil Nadu |  | 2020-05-29 | 9909 | 133 | 18545 | 0 |
| **2** | Delhi |  | 2020-05-29 | 7264 | 303 | 15257 | 0 |
| **3** | Gujarat |  | 2020-05-29 | 7549 | 938 | 15195 | 0 |
| **4** | Rajasthan |  | 2020-05-29 | 4457 | 173 | 7703 | 0 |
| **5** | Madhya Pradesh |  | 2020-05-29 | 3927 | 313 | 7261 | 0 |
| **6** | Uttar Pradesh |  | 2020-05-29 | 3991 | 182 | 6991 | 0 |
| **7** | West Bengal |  | 2020-05-29 | 1578 | 289 | 4192 | 0 |
| **8** | Andhra Pradesh |  | 2020-05-29 | 2057 | 58 | 3171 | 0 |
| **9** | Bihar |  | 2020-05-29 | 1083 | 15 | 3061 | 0 |
| **10** | Karnataka |  | 2020-05-29 | 781 | 47 | 2418 | 0 |
| **11** | Punjab |  | 2020-05-29 | 1918 | 40 | 2139 | 0 |
| **12** | Telengana |  | 2020-05-29 | 1284 | 63 | 2098 | 0 |
| **13** | Jammu and Kashmir |  | 2020-05-29 | 854 | 26 | 1921 | 0 |
| **14** | Odisha |  | 2020-05-29 | 733 | 7 | 1593 | 0 |
| **15** | Haryana |  | 2020-05-29 | 838 | 18 | 1381 | 0 |
| **16** | Kerala |  | 2020-05-29 | 552 | 7 | 1004 | 0 |
| **17** | Assam |  | 2020-05-29 | 87 | 4 | 781 | 0 |
| **18** | Uttarakhand |  | 2020-05-29 | 79 | 4 | 469 | 0 |
| **19** | Jharkhand |  | 2020-05-29 | 185 | 4 | 448 | 0 |
| **20** | Chhattisgarh |  | 2020-05-29 | 83 | 0 | 369 | 0 |
| **21** | Chandigarh |  | 2020-05-29 | 187 | 4 | 279 | 0 |
| **22** | Himachal Pradesh |  | 2020-05-29 | 70 | 5 | 273 | 0 |
| **23** | Tripura |  | 2020-05-29 | 165 | 0 | 230 | 0 |
| **24** | Goa |  | 2020-05-29 | 37 | 0 | 68 | 0 |
| **25** | Ladakh |  | 2020-05-29 | 43 | 0 | 53 | 0 |
| **26** | Puducherry |  | 2020-05-29 | 12 | 0 | 46 | 0 |
| **27** | Manipur |  | 2020-05-29 | 4 | 0 | 44 | 0 |
| **28** | Andaman and Nicobar Islands |  | 2020-05-29 | 33 | 0 | 33 | 0 |
| **29** | Meghalaya |  | 2020-05-29 | 12 | 1 | 20 | 0 |
| **30** | Nagaland |  | 2020-05-29 | 0 | 0 | 4 | 0 |
| **31** | Arunachal Pradesh |  | 2020-05-29 | 1 | 0 | 2 | 0 |
| **32** | Dadar Nagar Haveli |  | 2020-05-29 | 0 | 0 | 2 | 0 |
| **33** | Sikkim |  | 2020-05-29 | 0 | 0 | 1 | 0 |
| **34** | Mizoram |  | 2020-05-29 | 1 | 0 | 1 | 0 |

* f, ax = plt.subplots(figsize=(12, 28))

data = state\_df[['State/UnionTerritory','Confirmed','Cured','Deaths']]

data.sort\_values('Confirmed',ascending=False,inplace=True)

sns.set\_color\_codes("pastel")

sns.barplot(x="Confirmed", y="State/UnionTerritory", data=data,label="Total", color="red")

sns.set\_color\_codes("muted")

sns.barplot(x="Cured", y="State/UnionTerritory", data=data, label="Cured", color="green")

ax.legend(ncol=5, loc="lower right", frameon=True)

ax.set(ylabel="",xlabel="Cases")

i = 0

for p in ax.patches:

x = p.get\_x() + p.get\_width() + 3

y = p.get\_y() + p.get\_height()/2

if i <= len(States):

ax.annotate(" "\*10 + str(int(p.get\_width())), (x, y))

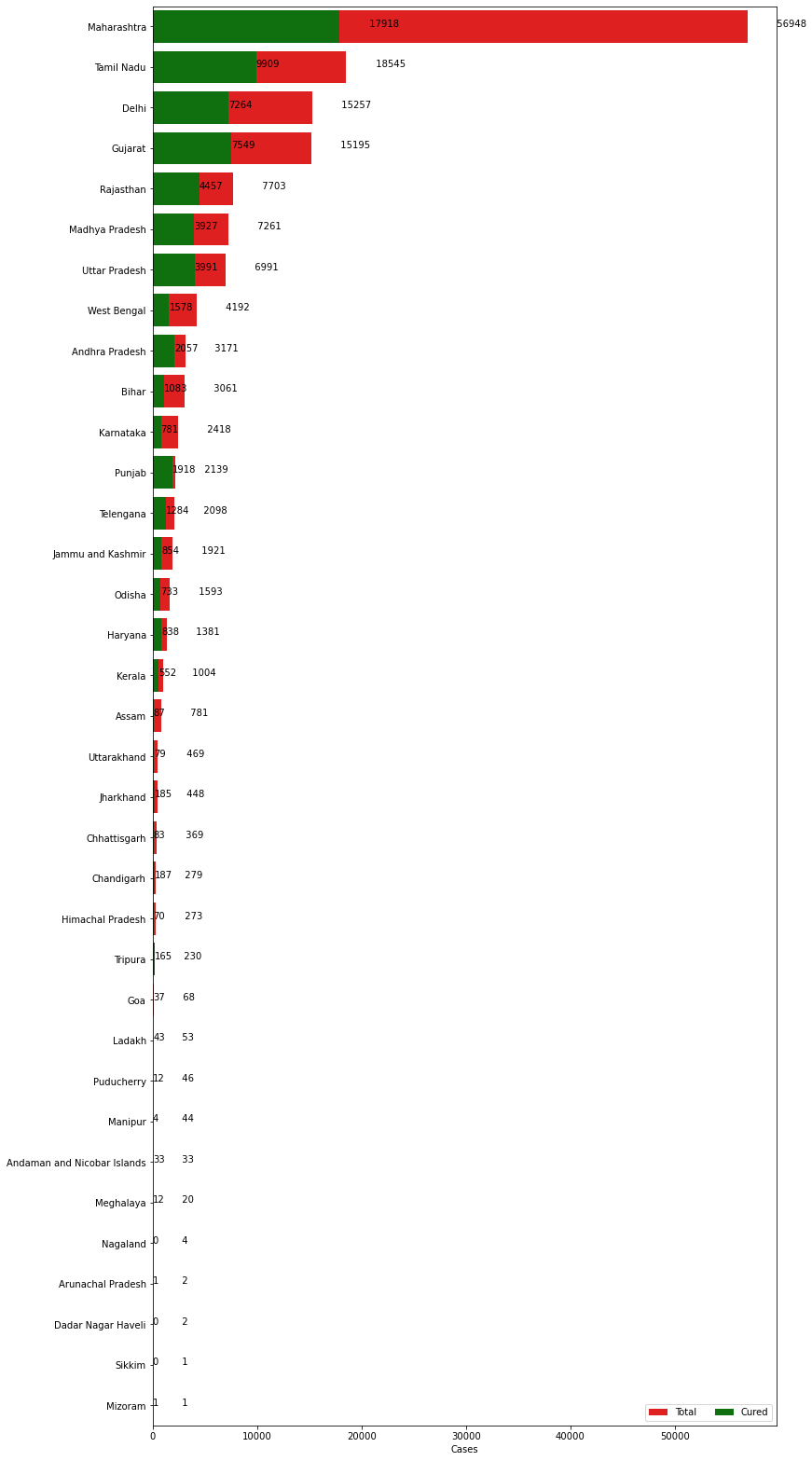
else:

ax.annotate(int(p.get\_width()), (x, y))

i += 1

# We use subplot function to draw the different types of plots like barplot

Output-



* f, ax = plt.subplots(figsize=(12, 28))

data = state\_df[['State/UnionTerritory','Confirmed','Cured','Deaths']]

data.sort\_values('Confirmed',ascending=False,inplace=True)

sns.set\_color\_codes("pastel")

sns.barplot(x="Confirmed", y="State/UnionTerritory", data=data,label="Total", color="red")

sns.set\_color\_codes("muted")

sns.barplot(x="Cured", y="State/UnionTerritory", data=data, label="Cured", color="green")

ax.legend(ncol=5, loc="lower right", frameon=True)

ax.set(ylabel="",xlabel="Cases")

total = total\_cases\_india

i = 0

for p in ax.patches:

percentage = '{:.1f}%'.format(100 \* p.get\_width()/total)

x = p.get\_x() + p.get\_width() + 3

y = p.get\_y() + p.get\_height()/2

if i <= len(States):

ax.annotate(" "\*10 + str(percentage), (x, y))

else:

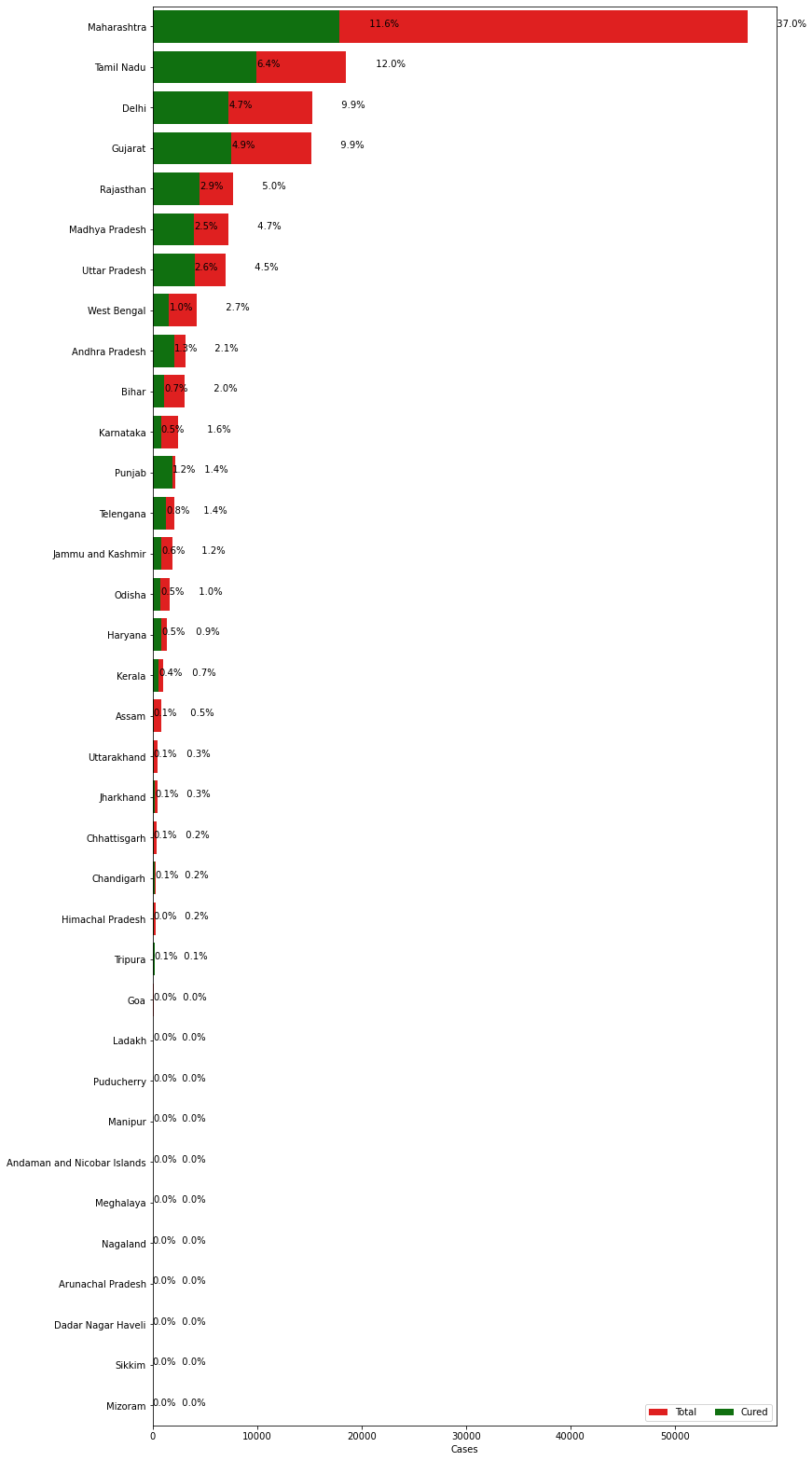
ax.annotate(percentage, (x, y))

i += 1

#In the above barplot we see the actual figures of cured , total, deaths, confirmed cases but here we simply see that figures into percentage .

Let us see the percentage barplot.

Output -



**Overall Covid19 Status in India**

* print("Total infected cases in India: ", total\_cases\_india)

print("Total cured cases in India: ", cured\_cases\_india)

print("Total active cases in India: ", active\_cases\_india)

print("Total death cases in India: ", death\_cases\_india)

plot\_pie(active\_cases\_india, cured\_cases\_india, death\_cases\_india, "India")

# By using pie() function here we summarize all covid-19 (total\_cases\_india, cured\_cases\_india , active\_cases\_india, death\_cases\_india )data in one piechart.

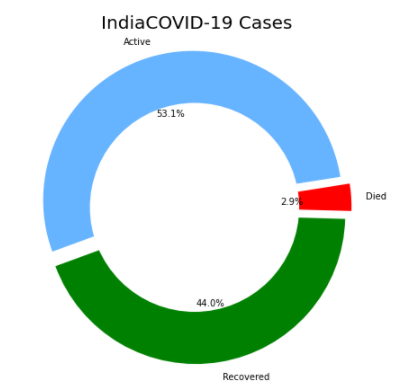
Output –

Total infected cases in India: 154001.0

Total cured cases in India: 67692.0

Total active cases in India: 81778.0

Total death cases in India: 4531.0



**VISUALISING THE SPREADS GEOGRAPHICALLY**

India\_coord.rename(columns = {"Name of State / UT" : "State/UnionTerritory"},inplace = True)

set(India\_coord['State/UnionTerritory'].values).symmetric\_difference(set(state\_df['State/UnionTerritory'].values))

{'Andaman And Nicobar ',

'Andaman and Nicobar Islands',

'Arunachal Pradesh',

'Arunachal Pradesh ',

'Assam',

'Assam ',

'Bihar',

'Bihar ',

'Chandigarh',

'Chandigarh ',

'Chhattisgarh',

'Chhattisgarh ',

'Dadar Nagar Haveli',

'Dadra And Nagar Haveli ',

'Goa',

'Goa ',

'Gujarat',

'Himachal Pradesh',

'Himachal Pradesh ',

'Jammu and Kashmir',

'Jharkhand',

'Jharkhand ',

'Ladakh',

'Lakshadweep ',

'Madhya Pradesh',

'Madhya Pradesh ',

'Manipur',

'Manipur ',

'Meghalaya',

'Meghalaya ',

'Mizoram',

'Mizoram ',

'Nagaland',

'Nagaland ',

'Odisha',

'Orissa ',

'Puducherry',

'Puducherry ',

'Sikkim',

'Sikkim ',

'Tripura',

'Tripura ',

'Union Territory of Jammu and Kashmir',

'Union Territory of Ladakh',

'West Bengal',

'West Bengal '}

India\_coord['State/UnionTerritory'] = India\_coord['State/UnionTerritory'].str.strip()

state\_df['State/UnionTerritory'] = state\_df['State/UnionTerritory'].str.strip()

set(India\_coord['State/UnionTerritory'].values).symmetric\_difference(set(state\_df['State/UnionTerritory'].values))

{'Andaman And Nicobar',

'Andaman and Nicobar Islands',

'Dadar Nagar Haveli',

'Dadra And Nagar Haveli',

'Gujarat',

'Jammu and Kashmir',

'Ladakh',

'Lakshadweep',

'Odisha',

'Orissa',

'Union Territory of Jammu and Kashmir',

'Union Territory of Ladakh'}

To create the map first install the python library pip install folium then import

The folium and it provide the class folium.map() that takes the location

Parameter in terms of latitude and longitude and generate a map.

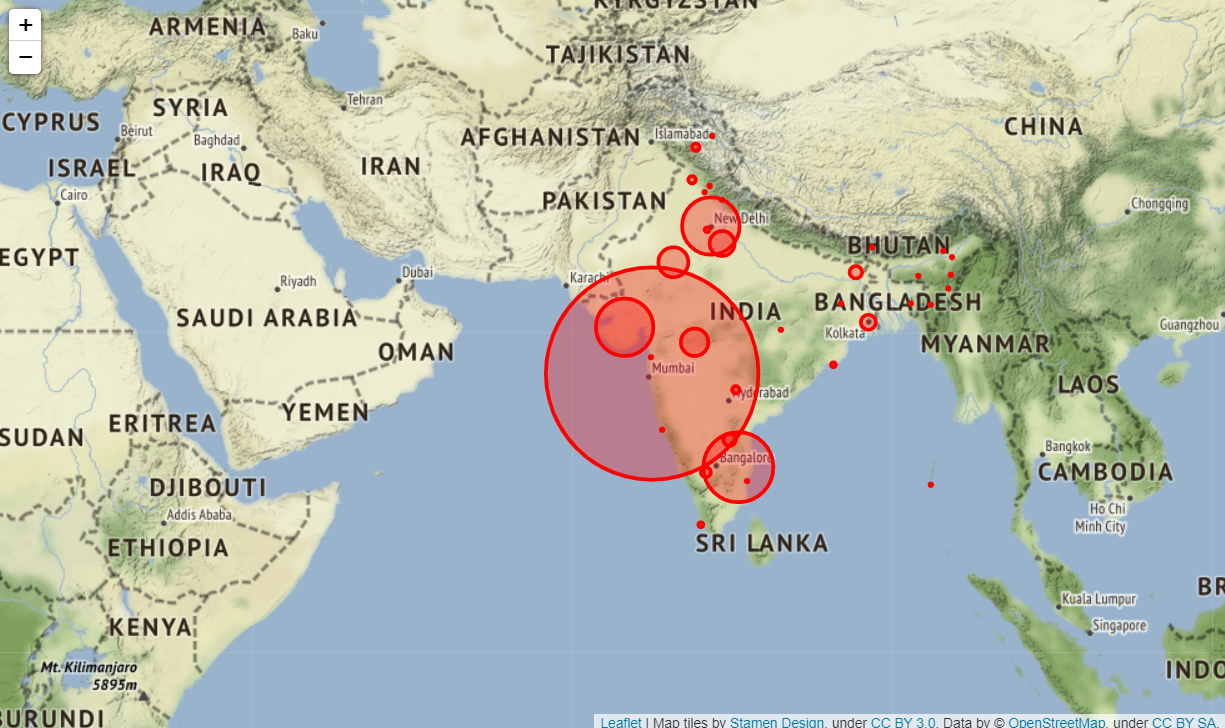
We have generated some map it is shown below.

map = folium.Map(location=[20, 70], zoom\_start=4,tiles='Stamenterrain')

for lat, lon, value, name in zip(df\_full['Latitude'], df\_full['Longitude'], df\_full['Confirmed'], df\_full['State/UnionTerritory']):

folium.CircleMarker([lat, lon], radius=value\*0.0015, popup = ('<strong>State</strong>: ' + str(name).capitalize() + '<br>''<strong>Total Cases</strong>: ' + str(value) + '<br>'),color='red',fill\_color='red',fill\_opacity=0.3 ).add\_to(map)

map

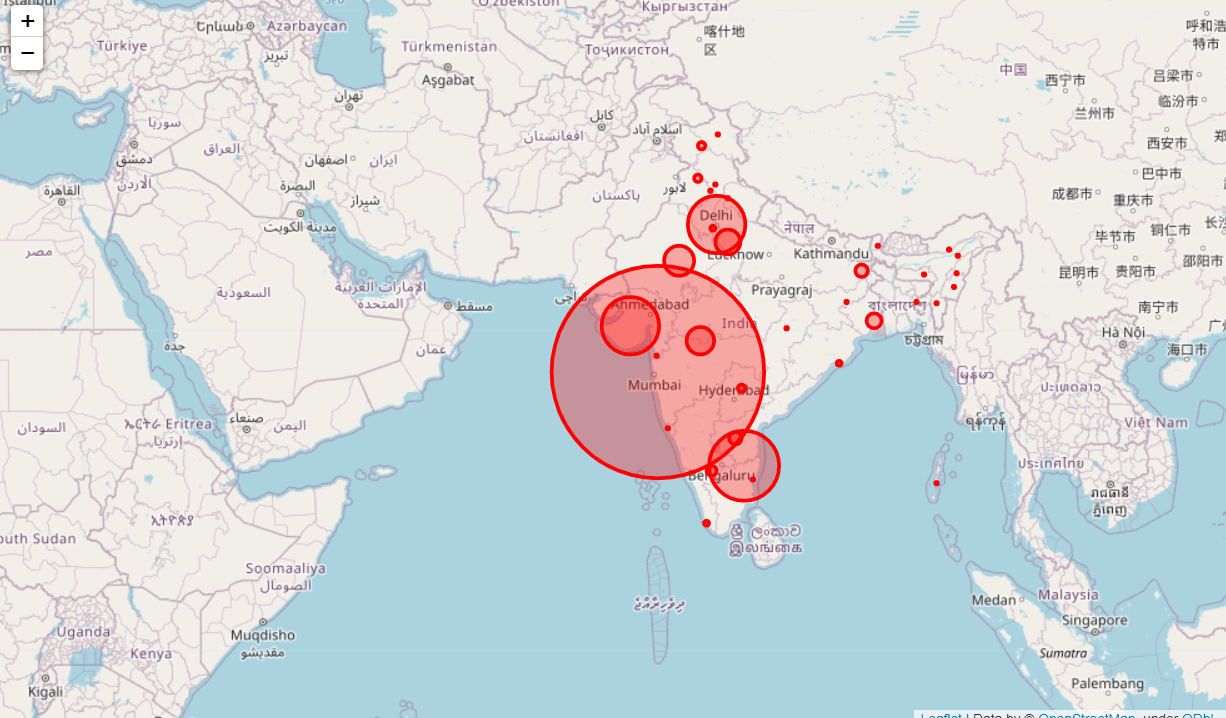


map = folium.Map(location=[20, 70], zoom\_start=4,tiles='OpenStreetMap')

for lat, lon, value, name in zip(df\_full['Latitude'], df\_full['Longitude'], df\_full['Confirmed'], df\_full['State/UnionTerritory']):

folium.CircleMarker([lat, lon], radius=value\*0.0015, popup = ('<strong>State</strong>: ' + str(name).capitalize() + '<br>''<strong>Total Cases</strong>: ' + str(value) + '<br>'),color='red',fill\_color='red',fill\_opacity=0.3 ).add\_to(map)

map

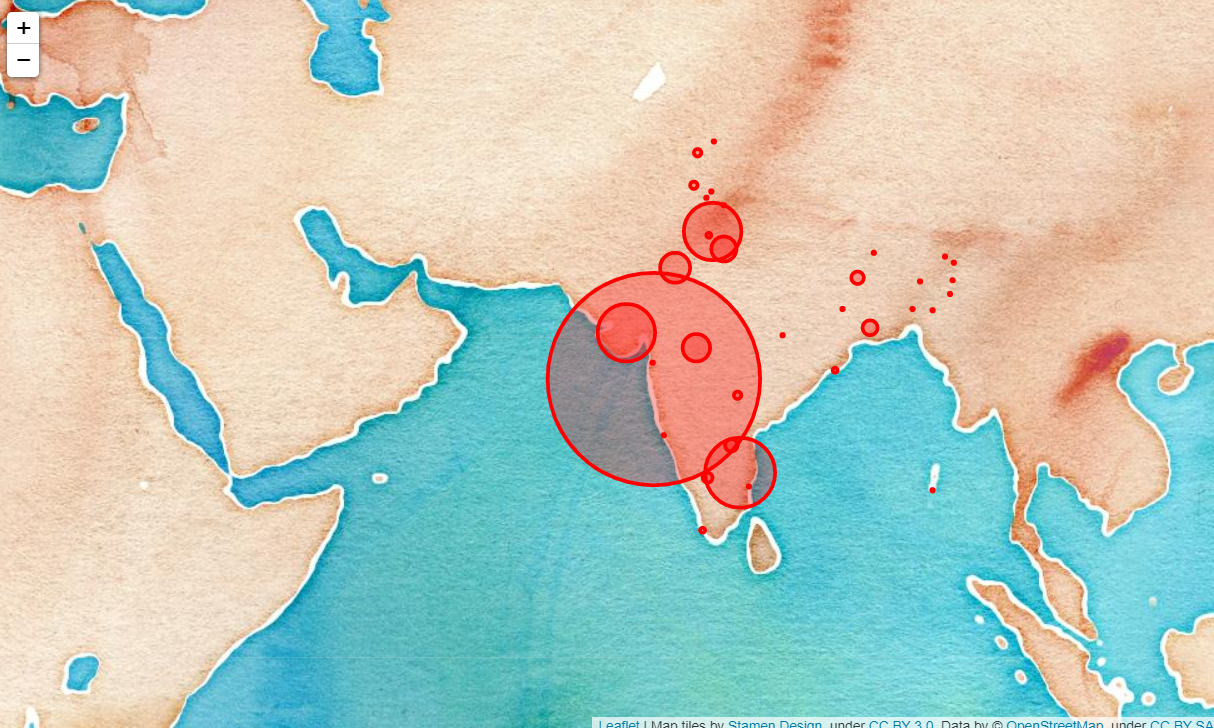


map = folium.Map(location=[20, 70], zoom\_start=4,tiles='Stamenwatercolor')

for lat, lon, value, name in zip(df\_full['Latitude'], df\_full['Longitude'], df\_full['Confirmed'], df\_full['State/UnionTerritory']):

folium.CircleMarker([lat, lon], radius=value\*0.0015, popup = ('<strong>State</strong>: ' + str(name).capitalize() + '<br>''<strong>Total Cases</strong>: ' + str(value) + '<br>'),color='red',fill\_color='red',fill\_opacity=0.3 ).add\_to(map)

map



**Lets check the trend of the virus**

Here we are checking the status of covid-19 by groupby()function that is used to split the data into groups in this we are checking the new cases, deaths by using groupby function.

df\_daywise\_India = df\_final\_India.groupby("Date")['Confirmed','Cured','Deaths',"New Cases"].sum().reset\_index()

df\_daywise\_India

| **Date** | **Confirmed** | **Cured** | **Deaths** | **New Cases** |
| --- | --- | --- | --- | --- |
| **0** | 2020-01-31 | 1.0 | 0.0 | 0.0 | 0.0 |
| **1** | 2020-02-01 | 2.0 | 0.0 | 0.0 | 1.0 |
| **2** | 2020-02-02 | 3.0 | 0.0 | 0.0 | 1.0 |
| **3** | 2020-02-03 | 3.0 | 0.0 | 0.0 | 0.0 |
| **4** | 2020-02-04 | 3.0 | 0.0 | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... |
| **115** | 2020-05-25 | 136203.0 | 57721.0 | 4021.0 | 6673.0 |
| **116** | 2020-05-26 | 142410.0 | 60491.0 | 4167.0 | 6207.0 |
| **117** | 2020-05-27 | 147754.0 | 64426.0 | 4337.0 | 5344.0 |
| **118** | 2020-05-28 | 154001.0 | 67692.0 | 4531.0 | 6247.0 |
| **119** | 2020-05-29 | 154001.0 | 67692.0 | 4531.0 | 0.0 |

120 rows × 5 columns

fig = go.Figure()

fig.add\_trace(go.Scatter(x=df\_daywise\_India['Date'], y = df\_daywise\_India['Confirmed'], mode='lines+markers',name='Total Cases'))

fig.update\_layout(title\_text='Trend of Coronavirus Cases in India (Cumulative cases)',plot\_bgcolor='rgb(230, 230, 230)')

fig.show()

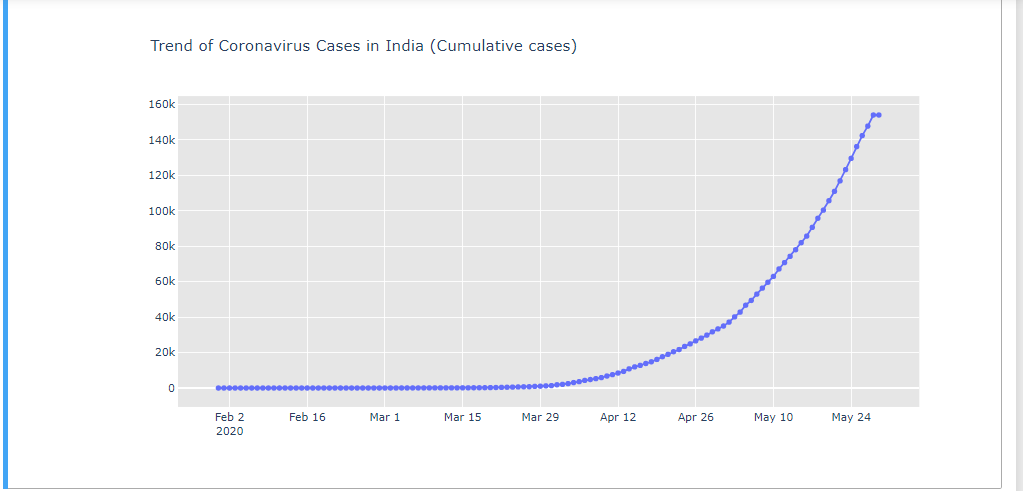
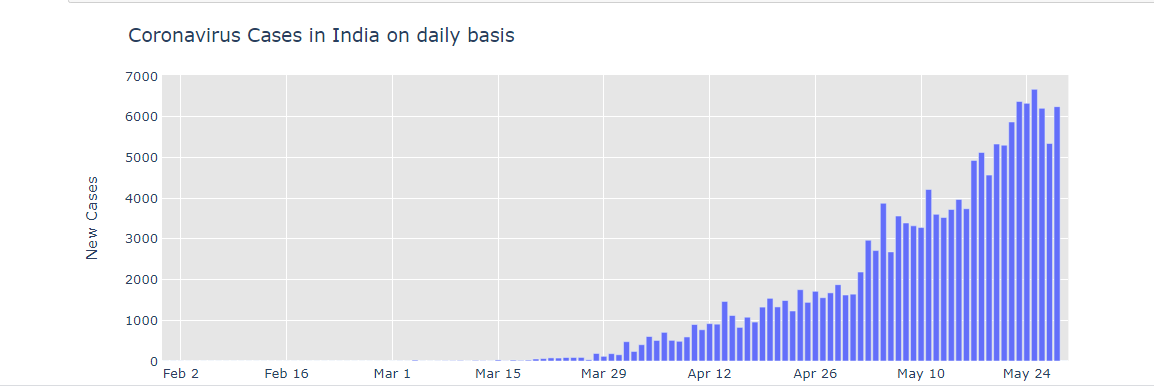


fig = px.bar(df\_daywise\_India, x="Date", y="New Cases", barmode='group', height=400)

fig.update\_layout(title\_text='Coronavirus Cases in India on daily basis',plot\_bgcolor='rgb(230, 230, 230)')

fig.show()



In this case we are plotting the graph with the use of different colors

fig = px.bar(df\_daywise\_India, x="Date", y="Confirmed", color='Confirmed', orientation='v', height=600

title='Confirmed Cases in India', color\_discrete\_sequence = px.colors.cyclical.IceFire)

'''Colour Scale for plotly

https://plot.ly/python/builtin-colorscales/

'''

fig.update\_layout(plot\_bgcolor='rgb(230, 230, 230)')

fig.show()

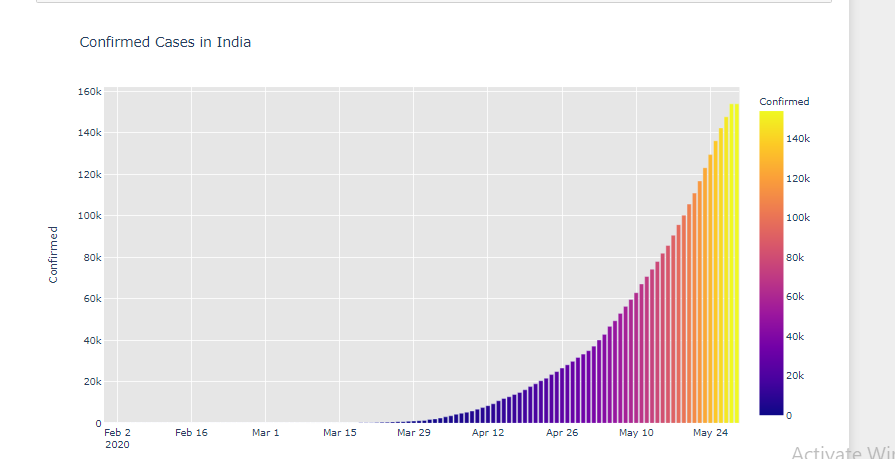
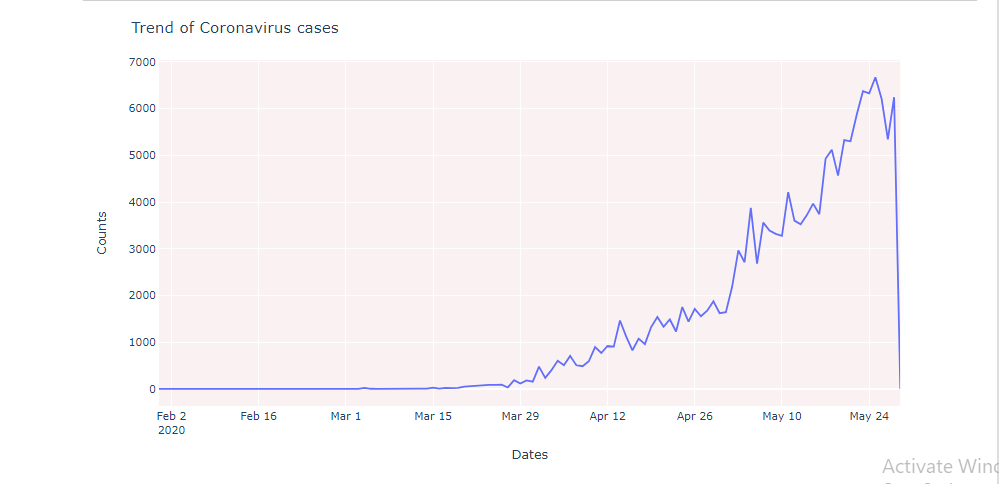


fig = px.line(x=df\_daywise\_India['Date'], y=df\_daywise\_India['New Cases'], labels = {'x': "Dates",'y': "Counts"})

fig.update\_layout( showlegend=False,title\_text="Trend of Coronavirus cases")

fig.update\_layout(plot\_bgcolor='rgb(250, 242, 242)')

fig.show()



**Forecasting Using fbprophet**

* from fbprophet import Prophet

FBProphet uses time as a regressor and tries to fit several linear and nonlinear function of time as components. By default, FBProphet will fit the data using a linear model but it can be changed to the nonlinear model (logistics growth) from its arguments.

* df = df\_daywise\_India.iloc[:-1,]

df\_train = df.loc[df['Date']<= "2020-05-23",:]

df\_test = df.loc[df['Date'] > "2020-05-23",:]

In .iloc function we apply index other than column name and in .loc function we have to give column name.

Here we declared some variable that contains different data-

confirmed\_train = df\_train[['Date','Confirmed']]

confirmed\_test = df\_test[['Date','Confirmed']]

This contains the data of the Date and Confirmed column of df\_train and df\_test dataset.

deaths\_train = df\_train[['Date','Deaths']]

deaths\_test = df\_test[['Date','Deaths']]

This contains the data pf the Date and Deaths column of df\_train and df\_test dataset.

recovered\_train = df\_train[['Date','Cured']]

recovered\_test = df\_test[['Date','Cured']]

This contains the data pf the Date and Recovered column of df\_train and df\_test dataset.

We stored data in variables with date and Confirmed, Deaths, Cured in different variables for plotting graph for visualizing Number of Confirmed, Deaths, Recovered cases.

* confirmed\_train.columns = ['ds','y']

confirmed\_train.tail()

We show confirmed cases table in train data part with last some records.

We assigned ‘ds’ for date

‘y’ for number of confirmed cases.

|  | **ds** | **y** |
| --- | --- | --- |
| **109** | 2020-05-19 | 100325.0 |
| **110** | 2020-05-20 | 105654.0 |
| **111** | 2020-05-21 | 110956.0 |
| **112** | 2020-05-22 | 116827.0 |
| **113** | 2020-05-23 | 123202.0 |

* m = Prophet()

m.fit(confirmed\_train)

future = m.make\_future\_dataframe(periods=5,freq = "D")

future.tail(5)

fit() function is used for training of model using data examples. Fit function adjusts weights according to data values so that better accuracy can be achieved. The make\_future\_dataframe function takes the model object and a number of periods to forecast and produces a suitable dataframe.

|  | **ds** |
| --- | --- |
| **114** | 2020-05-24 |
| **115** | 2020-05-25 |
| **116** | 2020-05-26 |
| **117** | 2020-05-27 |
| **118** | 2020-05-28 |

* forecast = m.predict(future)

forecast

We predict confirmed train cases and show result of predicted values.

* result\_df = forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail(5)

result\_df['Actual'] = confirmed\_test['Confirmed']

result\_df

We are adding Actual column having original values of Confirmed cases.

|  | **ds** | **yhat** | **yhat\_lower** | **yhat\_upper** | **Actual** |
| --- | --- | --- | --- | --- | --- |
| **114** | 2020-05-24 | 118135.005672 | 116031.134173 | 120166.491494 | 129530.0 |
| **115** | 2020-05-25 | 121943.212815 | 119920.546379 | 123995.693852 | 136203.0 |
| **116** | 2020-05-26 | 125768.732457 | 123674.219518 | 127855.242383 | 142410.0 |
| **117** | 2020-05-27 | 129581.923540 | 127534.129833 | 131621.306928 | 147754.0 |
| **118** | 2020-05-28 | 133369.567558 | 131333.049412 | 135430.553741 | 154001.0 |

* trace0 = go.Scatter(

x = result\_df['ds'],

y = result\_df['Actual'],

mode = 'lines+markers',

name='Actuals',

line = dict(color = '#dd0000', shape = 'linear'),

opacity = 0.3,

connectgaps=True

)

trace1 = go.Scatter(

x = result\_df['ds'],

y = result\_df['yhat'],

name='Predicted',

mode = 'lines+markers',

marker = dict(

size = 10,

color = '#44dd00'),

opacity = 0.3

)

data = [trace0, trace1]

layout = go.Layout(

yaxis=dict(

title="Results for Prophet (Total Cases)"

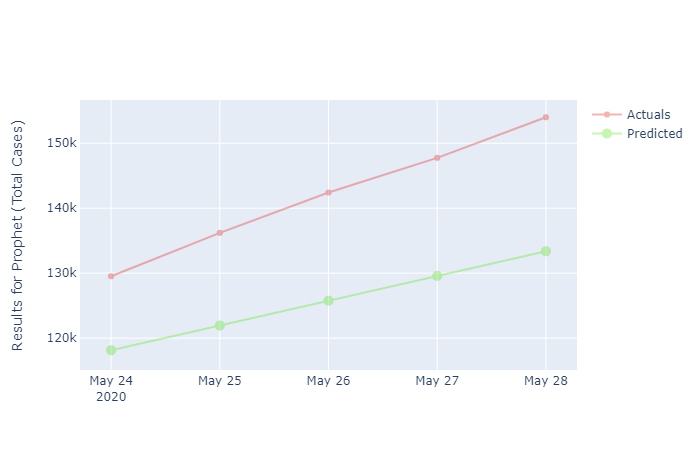
)

)

fig = go.Figure(data=data, layout=layout)

fig.show()

In that we are plotting graph for total cases. go.Scatter() can be used both for plotting points (makers) or lines, depending on the value of mode. By fig.show() function graph will be display.



This is a graph which shows total cases having actual and predicted.

The trace0 shown in red is the actual number of confirmed cases over the span from 24th may 2020 to 28th may 2020. We have plotted column ‘ds’ and ‘Actual’ of dataframe ‘result\_df’ on the x and y axis respectively. Similarly trace1 shown in green is the predicted number of confirmed cases over the span from 24th may 2020 to 28th may 2020. Here, we have plotted column ‘ds’ and ‘yhat’ of the datarame ‘result\_df’ on the x and y axis respectively.

* recovered\_train.columns = ['ds','y']

recovered\_train.tail()

We show confirmed cases table in train data part with last some records.

We assigned ‘ds’ for date

‘y’ for number of Recovered cases.

| **ds** | **y** |
| --- | --- |
| **109** | 2020-05-19 | 39174.0 |
| **110** | 2020-05-20 | 42298.0 |
| **111** | 2020-05-21 | 45300.0 |
| **112** | 2020-05-22 | 48534.0 |
| **113** | 2020-05-23 | 51784.0 |

Last five records are pinted as tail function used.

* m = Prophet()

m.fit(recovered\_train)

future = m.make\_future\_dataframe(periods=5,freq = "D")

future.tail(5)

By using fit function we train model for recovered data of training part.

And print the some of that last records.

|  | **ds** |
| --- | --- |
| **114** | 2020-05-24 |
| **115** | 2020-05-25 |
| **116** | 2020-05-26 |
| **117** | 2020-05-27 |
| **118** | 2020-05-28 |

* forecast = m.predict(future)

forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail(5)

result\_df = forecast.tail(5)

result\_df['Actual'] = recovered\_test['Cured']

result\_df

We predicting recovered cases and adding new column in result\_df column. Printing data of recovered with predicted and actual values.

* trace0 = go.Scatter(

x = result\_df['ds'],

y = result\_df['Actual'],

mode = 'lines+markers',

name='Actuals',

line = dict(color = '#dd0000', shape = 'linear'),

opacity = 0.3,

connectgaps=True

)

trace1 = go.Scatter(

x = result\_df['ds'],

y = result\_df['yhat'],

name='Predicted',

mode = 'lines+markers',

marker = dict(

size = 10,

color = '#44dd00'),

opacity = 0.3

)

data = [trace0, trace1]

layout = go.Layout(

yaxis=dict(

title="Results for Prophet (Recovered)"

)

)

fig = go.Figure(data=data, layout=layout)

fig.show()

We are ploting graph of recovered cases. We are ploting by go.Scatter plot with predicted and actual values.



The trace0 shown in red is the actual number of recovered cases over the span from 24th may 2020 to 28th may 2020. We have plotted column ‘ds’ and ‘Actual’ of dataframe ‘result\_df’ on the x and y axis respectively. Similarly trace1 shown in green is the predicted number of recovered cases over the span from 24th may 2020 to 28th may 2020. Here, we have plotted column ‘ds’ and ‘yhat’ of the datarame ‘result\_df’ on the x and y axis respectively.

* deaths\_train.columns = ['ds','y']

deaths\_train.tail()

We are printing data of death cases of training data.

We assigned ‘ds’ for date

‘y’ for number of Death cases.

| **ds** | **y** |
| --- | --- |
| **109** | 2020-05-19 | 3163.0 |
| **110** | 2020-05-20 | 3303.0 |
| **111** | 2020-05-21 | 3435.0 |
| **112** | 2020-05-22 | 3583.0 |
| **113** | 2020-05-23 | 3720.0 |

* m = Prophet(seasonality\_mode= 'multiplicative')

m.fit(deaths\_train)

future = m.make\_future\_dataframe(periods=5,freq = "D")

future.tail(5)

|  | **ds** |
| --- | --- |
| **114** | 2020-05-24 |
| **115** | 2020-05-25 |
| **116** | 2020-05-26 |
| **117** | 2020-05-27 |
| **118** | 2020-05-28 |

We are taking Prophet function with multiplicative seasonality mode, and we are training model by using fit function for deaths cases. Printing last five records of it by using tail function.

* forecast = m.predict(future)

forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail(5)

result\_df = forecast.tail(5)

result\_df['Actual'] = deaths\_test['Deaths']

result\_df

We are predicting records for death cases and using predict function and new column is added to result\_df.

* trace0 = go.Scatter(

x = result\_df['ds'],

y = result\_df['Actual'],

mode = 'lines+markers',

name='Actuals',

line = dict(color = '#dd0000', shape = 'linear'),

opacity = 0.3,

connectgaps=True

)

trace1 = go.Scatter(

x = result\_df['ds'],

y = result\_df['yhat'],

name='Predicted',

mode = 'lines+markers',

marker = dict(

size = 10,

color = '#44dd00'),

opacity = 0.3

)

data = [trace0, trace1]

layout = go.Layout(

yaxis=dict(

title="Results for Prophet (Death)"

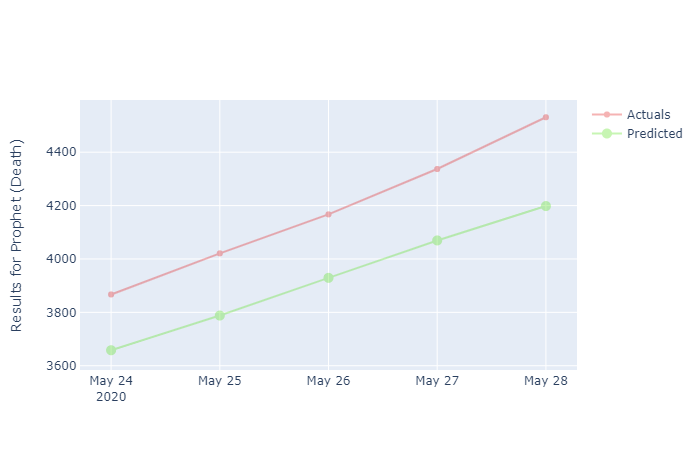
)

)

fig = go.Figure(data=data, layout=layout)

fig.show()

We are plotting graph for Death cases with actual and predicated values of Death cases.



The trace0 shown in red is the actual number of Death cases over the span from 24th may 2020 to 28th may 2020. We have plotted column ‘ds’ and ‘Actual’ of dataframe ‘result\_df’ on the x and y axis respectively. Similarly trace1 shown in green is the predicted number of death cases over the span from 24th may 2020 to 28th may 2020. Here, we have plotted column ‘ds’ and ‘yhat’ of the datarame ‘result\_df’ on the x and y axis respectively.

=========================================================

* **CONCLUSION**

1. In our project, we importing two files.
2. Covid\_19\_India.csv
3. Indian Coordinates
4. We then prepare our data for analysis with the help of data cleaning by removing and modifying data that is incorrect, incomplete, duplicated or in not proper format.
5. Then we perform the merging operations wherin we join two dataframes. In this way we get all the information about a particular entry common to both the two datasets.
6. Each states’s cases are analyzed using pie chart. Also, overall country’s cases are analyzed.
   * + - States like Maharashtra, Gujarat, MP, West Bengal have a death rate of nearly 5 percent.
       - States like Tamilnadu, MP, UP, Rajasthan, Andhra Pradesh, Telangana, Haryana, Kerela have recovered more than 50 percent.
       - Punjab has almost 90 percent of cases recovered.
       - No death cases have been recorded in Tripura, Goa, Ladakh, Pondicherry, and Manipura.
       - Andaman and Nicobar Islands have all the cases recovered.
       - Dadra and Nagar Haveli, Sikkim, Nagaland has all the cases active.
       - Maharashtra has the highest number of cases recorded.
       - Overall in India 53 % are active, 3%have died due the virus, 44% have recovered.
7. While merging two datasets, some data gets repeated or may be missing, data cleaning is an important step to obtain appropriate data for analysis.
8. The maps plotted become a clear visual indicator of COVID hotspots.
9. From the we can interpret that:
10. First table lists the number of confirmed, cured, deaths and number of new cases per day. From this we can conclude that the number of COVID cases soared from mid April.
11. The first graph shows that the curve was flat until mid March and it grew exponentially thereafter.