iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** V **Month of publication:** May 2020

DOI: http://doi.org/10.22214/ijraset.2020.5076

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue V May 2020- Available at www.ijraset.com

Tracking the Spread of COVID-19 Cases in India using Data Visualizing and Forecasting Techniques

Mohammed Azam Sayeed[1], Noor Ayesha[2]
[1]*Department of Computer sciences, SET Jain University Bangalore -562112*

Abstract: This paper presents various data visualization techniques using Plotly and matplotlib libraries to extract insights from covid-19 data in detail for India, paper attempts to assist in taking pre-emptive steps to reduce risk of India becoming next hotspot and reducing loss of lives and economy. Also the paper explains various forecasting techniques that can help countries governments, health workers and police department to assess the risk and take necessary steps to curve the growth of infections.
Keywords: Plotly, matplotlib, Corona virus, India, holt winter, fbprophet, ARIMA models, Data visualization, Web Scraping.

I. UNPRECEDENTED TIMES: CORONAVIRUS PANDEMIC

The novel coronavirus that originated in Wuhan, China, has spread to almost all countries and declared as pandemic. The extent of this outbreak is rapid. It is hard to accurately assess the lethality of this virus and it appears to be far more fatal than the coronavirus that caused SARS and MERS. Scientists have identified two new strains of the coronavirus, indicating it's already has been mutated at least once. The biggest challenge is an unknown number of people have been infected by the virus without becoming symptomatic. These people are carriers of the virus without themselves showing any signs. Initially, people who showed no signs of infection were not quarantined and this leads to the spread of the virus at a massive rate. The virus is also shown to affect its hosts very disproportionately. Children seem to be less likely to be infected while the middle-aged and older adults are inexplicably infected. Men are more likely to die from the infection compared to women, and also people with a weaker immune system, Type 2 diabetes and high blood pressure. But recently many perfectly healthy young individuals have died from the infection making it even harder to understand the effect of COVID-19. The symptoms of COVID-19 are general flu-like and some patients develop a severe form of pneumonia. Patients develop symptoms like fever, muscle and body aches, coughs and sore throat about after six days of contracting the infection. Most people feel very miserable and weak and get better on their own, but a minority of patients will get worse after 5-7 days of illness and the patients have shortness of breath and worsening cough. The cough is dry and not wet. It's even seen that patients have strong headaches. And the symptoms very mildly different from the flu. Individuals infected from the virus may not have a cold. But some people don't get sick while being infected and are spreading the virus to new hosts. These people should be not out and about spreading the disease. People who got infected and have been successfully cured have also got infected by the COVID-19 again. Making it even harder to contain the outbreak. There is no antibiotic to treat the COVID-19 and it may not be available until the spring of 2021. All this makes it even more important to take preventive actions.

A. Coronavirus modes of spread:

[1]The coronavirus is thought to spread mainly from person to person. This can happen between people who are in close contact with one another. Droplets that are produced when an infected person coughs or sneezes may land in the mouths or noses of people who are nearby, or possibly be inhaled into their lungs. A person infected with coronavirus — even one with no symptoms — may emit aerosols when they talk or breathe. Aerosols are infectious viral particles that can float or drift around in the air for up to three hours. Another person can breathe in these aerosols and become infected with the coronavirus. This is why everyone should cover their nose and mouth when they go out in public. Coronavirus can also spread from contact with infected surfaces or objects. For example, a person can get COVID-19 by touching a surface or object that has the virus on it and then touching their own mouth, nose, or possibly their eyes.

B. Approach to slow the spread of covid19:

[1]Anyone who comes into close contact with someone who has COVID-19 is at increased risk of becoming infected themselves, and of potentially infecting others. Contact tracing can help prevent further transmission of the virus by quickly identifying and informing people who may be infected and contagious, so they can take steps to not infect others.

Contact tracing begins with identifying everyone that a person recently diagnosed with COVID-19 has been in contact with since they became contagious. In the case of COVID-19, a person may be contagious 48 to 72 hours before they started to experience symptoms. The contacts are notified about their exposure. They may be told what symptoms to look out for, advised to isolate themselves for a period of time, and to seek medical attention as needed if they start to experience symptoms.

C. Importance of Social Distancing:

[1]The COVID-19 virus primarily spreads when one person breathes in droplets that are produced when an infected person coughs or sneezes. In addition, any infected person, with or without symptoms, could spread the virus by touching a surface. The coronavirus could remain on that surface and someone else could touch it and then touch their mouth, nose or eyes. That's why it's so important to try to avoid touching public surfaces or at least try to wipe them with a disinfectant.

Social distancing refers to actions taken to stop or slow down the spread of a contagious disease. For an individual, it refers to maintaining enough distance (6 feet or more) between yourself and another person to avoid getting infected or infecting someone else. School closures, directives to work from home, library closings, and cancelling meetings and larger events help enforce social distancing at a community level.

Slowing down the rate and number of new coronavirus infections is critical to reduce the risk that large numbers of critically ill patients cannot receive life-saving care. Highly realistic projections show that unless we begin extreme social distancing now — every day matters — our hospitals and other healthcare facilities will not be able to handle the likely influx of patients.

D. Necessity of Lockdown in Countries

[2]In the absence of treatment or a vaccine, ceasing most human contact is really the only way to stop the spread of the virus. Essentially, the less contact people have with each other, the less the virus can spread. Given the rapid spread of the virus, social lockdown is urgent to bring overall transmission down, and see whether testing followed by isolation could be effective – this is all in an attempt to 'flatten the curve' or reduce infections and spread cases out over a longer time frame to avoid overwhelming health systems.

Since the new coronavirus can spread unnoticed so easily, many governments have felt the best way to ensure people have minimal contact with each other is to order total lockdowns, with people only being allowed to leave to get food or medicine, and to practise social distancing when they do leave their houses. Countries that had epidemics first, such as China and South Korea, have brought cases down dramatically through widespread testing and social distancing.

The rationale is to ensure that people with serious illness can seek medical care, and those who are infectious but asymptomatic or have mild illness don't pass it on to anyone else.

II. COLLECTION OF DATASETS : WEB SCRAPING

A. Extracting Real Time Data using Web Scraping:

1) *Need for Web Scraping:* Web scrapping is used for Extracting unstructured data from the websites and transforming it into a structured data. This allows the data to be stored in internal storage for data pre-processing and analytics.

Applications of Web scraping

a) *Tracking Competitive pricing:* Used to Extract product details and prices of competitors to stay relevant to the market.
b) *Sentiment Analysis:* Web scraping helps in analysing customer reaction on products from reviews and rating gathered from various forums.
c) *Market Research:* Service/Product launch can used web scraping to study the market and help curate product to market demands.
d) *Industry Scrutinization:* Webscraping will help access the competitors in the market.
e) Monitoring Brand value and Lead generation- used for Networking and better contacts for product advertisements.
f) *Content Aggregation:* Gather Information from various sources for further analysis , web scraping can be used to process unstructured data and transform to usable structured data.

This paper uses Web scraping for Content Aggregation as explained above for real-time data collection.

2) *Python Libraries required for Web scraping*

a) *Beautiful Soup Framework:* This Framework is used for pulling out information from the webpage. It uses the HTML tags and attributes to extract contents such as list,tables, paragraphs.

```
<!DOCTYPE html>                          -> html 5 webpages starts with this tag
<html>                                   -> Opening of the html tag, encloses all contents within this tag
<body>                                   ->  opening body tag, encloses all visible content of webpage
<h1> Heading </h1>                       -> Heading tag
<p> Paragraph content </p>               -> paragraph tag
<table>                                  ->  table contents given in table,tr (table row) and td tags (table cell)
    <tr>
        <td> Cell A  </td>
        <td> Cell B  </td>
    </tr>
</table>
< a href =”g.com”>press link </a>        -> a tag is used to provide hyperlinks in web pages, href is called attribute of a
</body> </html>
```

Fig. 1 HTML Basic Tags which is used by Beautiful Soup

b) Request: used for fetching URLs. This takes care of basic redirections, authentication, cookies etc. Since both the data posted on the used URLs given below are public content data, web scraping is most suitable and legal to generate datasets.

B. Implementation of Web Scraping for above Request URLs

We are using Two Request URL for DataSets:

1) *https://www.mohfw.gov.in/ :* This is official website from indian government for real time reported confirmed, recovered and deceased covid19 cases count per states of India

```
url = 'https://www.mohfw.gov.in/'

web_content = requests.get(url).content
soup = BeautifulSoup(web_content, "html.parser")
extract_contents = lambda row: [x.text.replace('\n', '') for x in row]
stats = [] # initialize stats
all_rows = soup.find_all('tr') # find all table rows
for row in all_rows:
    stat = extract_contents(row.find_all('td')) # find all data cells
    # notice that the data that we require is now a list of length 5
    if len(stat) == 5:
        stats.append(stat)
new_cols = ["Sr.No", "States/UT","Confirmed","Recovered","Deceased"]
state_data = pd.DataFrame(data = stats, columns = new_cols)
state_data['Confirmed'] = state_data['Confirmed'].map(int)

state_data['Recovered'] = state_data['Recovered'].map(int)

state_data['Deceased']  = state_data['Deceased'].map(int)

table = PrettyTable()

table.field_names = (new_cols)

for i in stats:

    table.add_row(i)

table.add_row(["","Total",

            sum(state_data['Confirmed']),

            sum(state_data['Recovered']),

            sum(state_data['Deceased'])])

print(table)
```

Sr.No	States/UT	Confirmed	Recovered	Deceased
1	Andaman and Nicobar Islands	33	16	0
2	Andhra Pradesh	1525	441	33
3	Arunachal Pradesh	1	1	0
4	Assam	43	32	1
5	Bihar	471	98	3
6	Chandigarh	88	17	0
7	Chhattisgarh	43	36	0
8	Delhi	3738	1167	61
9	Goa	7	7	0
10	Gujarat	4721	735	236
11	Haryana	360	227	4
12	Himachal Pradesh	40	30	1
13	Jammu and Kashmir	639	247	8
14	Jharkhand	111	20	3
15	Karnataka	598	255	25
16	Kerala	498	392	4
17	Ladakh	22	17	0
18	Madhya Pradesh	2719	524	145
19	Maharashtra	11506	1879	485
20	Manipur	2	2	0
21	Meghalaya	12	0	1
22	Mizoram	1	0	0
23	Odisha	154	55	1
24	Puducherry	8	5	0
25	Punjab	772	112	20
26	Rajasthan	2666	1116	62
27	Tamil Nadu	2526	1312	28
28	Telengana	1057	441	26
29	Tripura	2	2	0
30	Uttarakhand	58	37	0
31	Uttar Pradesh	2455	656	43
32	West Bengal	795	139	33
	Total	37671	10018	1223

Fig. 2 Pretty table (right) generated from Web scrapping from https://www.mohfw.gov.in/ link which is also converted to pandas data frame (left) on March 3rd 2020

2) *https://news.google.com/covid19/map?hl=en-IN&gl=IN&ceid=IN:en*: This is website from google which makes public data of confirmed, recovered and deaths covid19 cases globally.

```
url = 'https://news.google.com/covid19/map?hl=en-IN&gl=IN&ceid=IN:en'
# make a GET request to fetch the raw HTML content
web_content = requests.get(url).content
# parse the html content
soup = BeautifulSoup(web_content, "html.parser")
# remove any newlines and extra spaces from left and right
extract_contents = lambda row: [x.text.replace('\n', '') for x in row]
statsW = [] # initialize stats
all_rowsW = soup.find_all('tr') # find all table rows
locsW = []
for row in all_rowsW:
    loc=extract_contents(row.find_all('span'))
    locsW.append(loc)
    stat = extract_contents(row.find_all('td'))
    statsW.append(stat)
#converting to Pandas Dataframe
z1=pd.DataFrame(statsW[2:],columns=locsW[0][1:])
z1['Country']=locsW[2:]
z1['Country']=z1['Country'].apply(pd.Series)

#dtype conversion from object to int
z2=z1
#z2['Recovered']=z2['Recovered'].str.replace("—","0")
z2['Recovered']=z2['Recovered'].str.replace(",","").astype(int)
z2['Confirmed']=z2['Confirmed'].str.replace(",","").astype(int)
z2['Deaths']=z2['Deaths'].str.replace(",","").astype(int)
sortedz=z2.sort_values(by='Confirmed',ascending=False)
#printing sorted dataframe based on Confirmed Cases across globe
sorted.head(39)
```

Out[8]:

	Confirmed	Cases per 1 million people	Recovered	Deaths	Country
0	11,83,663	3,592	1,53,204	68,276	United States
1	2,17,466	4,617	1,18,902	25,264	Spain
2	2,10,717	3,498	81,654	28,884	Italy
3	1,86,599	2,809	—	28,446	United Kingdom
4	1,65,666	1,992	1,26,153	6,866	Germany
5	1,45,268	990	18,095	1,356	Russia
6	1,31,287	1,957	50,784	24,895	France
7	1,26,045	1,516	63,151	3,397	Turkey
8	1,01,147	479	42,991	7,025	Brazil
9	97,424	1,169	78,422	6,203	Iran
10	84,393	60	77,766	4,643	China
11	59,474	1,568	24,908	3,682	Canada
12	49,906	4,330	12,309	7,844	Belgium
13	45,928	1,429	13,550	1,266	Peru
14	42,533	31	11,707	1,373	India
15	40,571	2,325	—	5,056	Netherlands
16	29,905	3,483	24,500	1,473	Switzerland
17	29,538	1,692	3,300	1,564	Ecuador
18	27,011	789	4,134	184	Saudi Arabia
19	25,282	2,460	1,689	1,043	Portugal
20	22,317	2,160	1,543	2,679	Sweden
21	22,088	175	13,447	2,061	Mexico
22	21,506	4,370	13,386	1,303	Ireland
23	20,186	92	5,590	462	Pakistan
24	19,663	1,029	10,041	260	Chile
25	18,778	3,292	1,408	18	Singapore
26	16,705	1,775	3,196	99	Belarus
27	16,193	1,764	9,634	230	Israel
28	15,558	1,748	13,228	598	Austria
29	15,551	5,661	1,664	12	Qatar
30	15,084	120	3,981	541	Japan
31	14,163	1,432	2,762	126	United Arab Emirates
32	13,693	357	3,945	678	Poland
33	13,163	678	4,889	780	Romania
34	12,331	294	1,619	303	Ukraine
35	11,192	42	1,876	845	Indonesia
36	10,801	209	9,217	252	South Korea
37	9,563	1,642	6,987	484	Denmark
38	9,464	1,359	1,551	193	Serbia
39	9,455	56	1,063	177	Bangladesh

Fig.3 Sorted first 39 Countries DataFrame based on Highest Confirmed Cases (right) generated from Web scrapping from https://news.google.com/covid19/map?hl=en-IN&gl=IN&ceid=IN:en link (left) on March 3rd 2020

III. IMPORTANCE OF TRACKING INDIA COVID19 SPREAD IN GLOBAL PERSPECTIVE

1) From the Pandas Dataframe generated from Fig 3 we find that on march 3rd 2020 India has about 42533 number of Confirmed cases , 11707 and 1373 for Recovered and Deaths cases which Unfortunately makes India ranks in 15th position globally for highest confirmed grim list of covid19 cases reported. India faces a unique situation being the 2nd most populous country in the world only next to china which makes it possibility of the next hotspot for corona virus. Also social distances should be very closely monitored in India as people per square foot distance is much higher than china due to country size. Which is clearly depicted by infamous 2007 World Health Chart – Gapminder Data plot of Countries lifeExp per gdpPerCapita , the size of the bubble being the population size , considerably much larger than rest of the countries in Fig 4.
2) WHO is constantly tracking the status of India handling the covid19 situation , as India has shown the world before the successful control of diseases such as swine flu.
3) High Chances of India to be able to curb the curve of reported confirmed coronavirus virus cases, through being one of the largest manufacturer of HCQ tablets and already has exported large HCQ tablets across the world , also has great Healthcare and Engineering which are already working on low cost ventilators , India strategies can be used by WHO in other countries to reduce the loss of human lives if India succeed in covid battle. This makes it even important to control/monitor corona virus in India.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue V May 2020- Available at www.ijraset.com

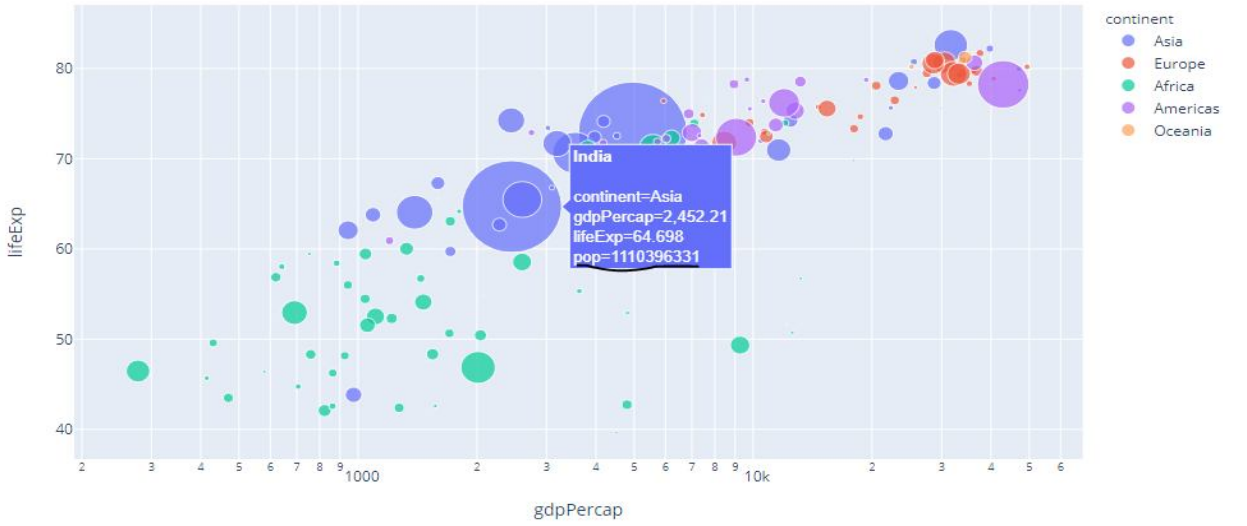


Fig.4 Infamous World Health Chart – Gapminder plot, India being very high population depicted by bubble size

```
In [10]: 1 sortedz=z2.sort_values(by='Confirmed',ascending=False).head(16)
         2 sortedz
```

Out[10]:

	Confirmed	Cases per 1 million people	Recovered	Deaths	Country
0	1183663	3,592	153204	68276	United States
1	217466	4,617	118902	25264	Spain
2	210717	3,498	81654	28884	Italy
3	186599	2,809	0	28446	United Kingdom
4	165666	1,992	126153	6866	Germany
5	145268	990	18095	1356	Russia
6	131287	1,957	50784	24895	France
7	126045	1,516	63151	3397	Turkey
8	101147	479	42991	7025	Brazil
9	97424	1,169	78422	6203	Iran
10	84393	60	77766	4643	China
11	59474	1,566	24908	3682	Canada
12	49906	4,330	12309	7844	Belgium
13	45928	1,429	13550	1286	Peru
14	42533	31	11707	1373	India
15	40571	2,325	0	5056	Netherlands

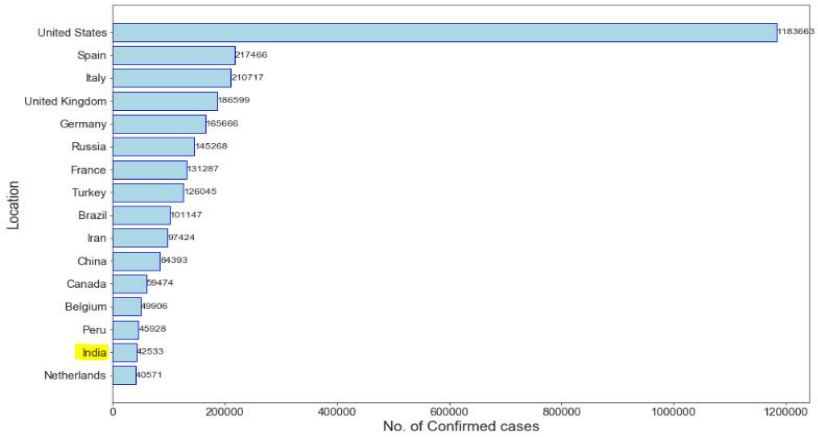


Fig.5 India has 15th highest Confirmed covid cases globally having greater risk of being next hotspot as per March 3rd 2020 reported data

```
In [32]: 1 import plotly.express as px
         2
         3
         4 fig = px.choropleth(z1, locations="iso",
         5                     color="Confirmed", # lifeExp is a column of gapminder
         6                     hover_name="Country", # column to add to hover information
         7                     color_continuous_scale=px.colors.sequential.Plasma)
         8
         9 fig.show()
```

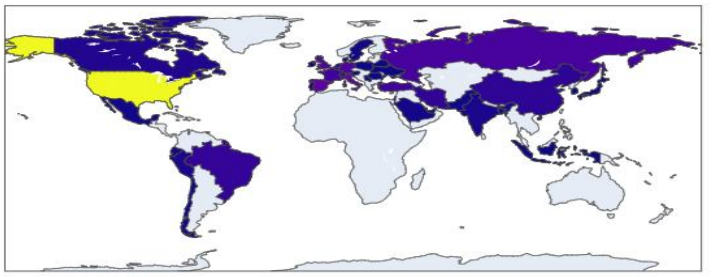


Fig.6 Confirmed Cases of covid19 globally for first 39 countries data in Fig 3 using Choropleth Maps

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue V May 2020- Available at www.ijraset.com

A. Exploring Different Data visualization provided by plotly Module for Tracking India Covid Confirmed Cases:

1) *Table Plot:* This can be used for colouring the cells in pandas Dataframe to show the highest value and relative variations in rows values through gradient colouring for Confirmed, Deceased and Recovered data values respectively.

```
In [59]: state_data.sort_values('Confirmed', ascending= False).style.background_gradient(cmap='Blues',subset=["Confirmed"])\
                        .background_gradient(cmap='Reds',subset=["Deceased"])\
                        .background_gradient(cmap='Greens',subset=["Recovered"])
```

Out[59]:

	Sr.No	States/UT	Confirmed	Recovered	Deceased
18	19	Maharashtra	11506	1879	485
9	10	Gujarat	4721	735	238
7	8	Delhi	3738	1167	61
17	18	Madhya Pradesh	2719	524	145
25	26	Rajasthan	2666	1116	62
26	27	Tamil Nadu	2526	1312	28
30	31	Uttar Pradesh	2455	656	43
1	2	Andhra Pradesh	1525	441	33
27	28	Telengana	1057	441	26
31	32	West Bengal	795	139	33
24	25	Punjab	772	112	20
12	13	Jammu and Kashmir	639	247	8
14	15	Karnataka	598	255	25
15	16	Kerala	498	392	4
4	5	Bihar	471	98	3
10	11	Haryana	360	227	4
22	23	Odisha	154	55	1
13	14	Jharkhand	111	20	3

Fig.7 Gradient coloring on Pandas

2) *Bar Plots and horizontal Bar plots:* Through Bar plot we find that the highest Confirmed cases are reported in Maharashtra with 11506, followed by next 5 states being Gujarat 4721, Delhi 3738, Madhya Pradesh 2719 , Rajasthan 2666, Tamil Nadu 2526. Same order of states is observed with recovered and Deceased counts. But we find Union territories apart from Delhi, and North east India are not infected as severely. Least 5 risk states are Arunachal Pradesh 1 case, Mizoram 1 cases, Tripura 2 cases, Manipur 2 cases, Goa 2 cases etc

```
In [32]: # barplot to show total confirmed cases Statewise
sns.set_style("ticks")
plt.figure(figsize = (15,10))
plt.barh(state_data["States/UT"], state_data["Confirmed"].map(int),
         align = 'center', color = 'lightblue', edgecolor = 'blue')
plt.xlabel('No. of Confirmed cases', fontsize = 18)
plt.ylabel('States/UT', fontsize = 18)
plt.gca().invert_yaxis() # this is to maintain the order in which the states appear
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.title('Total Confirmed Cases Statewise', fontsize = 20)

for index, value in enumerate(state_data["Confirmed"]):
    plt.text(value, index, str(value), fontsize = 12, verticalalignment = 'center')
plt.show()
```

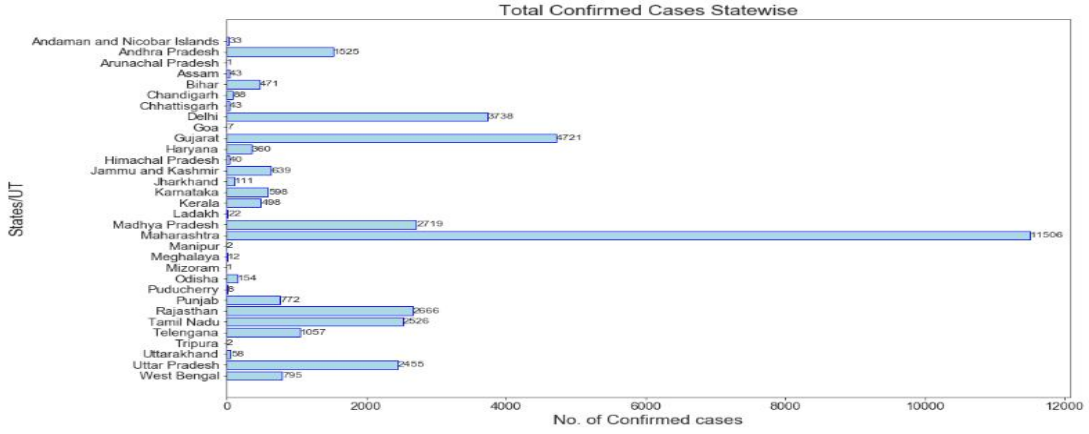


Fig.8 Barplot for states with respect to Confirmed Cases

```
fig = go.Figure(data=[
    go.Bar(name='Confirmed', x=state_data['States/UT'], y=state_data['Confirmed'],text=state_data['Confirmed']
           ,textposition='auto'),
    go.Bar(name='Recovered', x=state_data['States/UT'], y=state_data['Recovered'],text=state_data['Recovered']
           ,textposition='auto'),
    go.Bar(name='Deceased', x=state_data['States/UT'], y=state_data['Deceased'])
])
# Change the bar mode
fig.update_layout(barmode='stack')
fig.show()
```

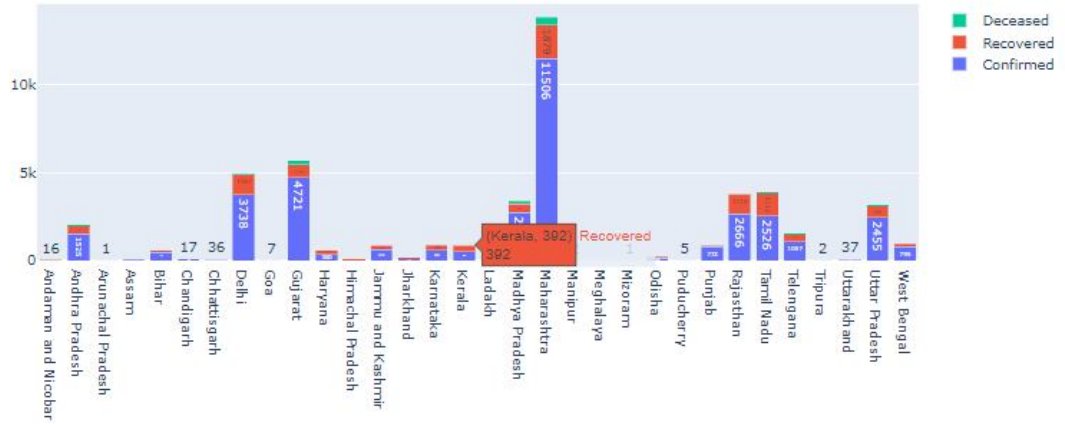


Fig.9 Horizontal Bar plot of states on Deceased, Recovered and Confirmed cases with annotations

In place Comparison with the cases of confirmed, death and recovered can be depicted more vividly using the horizontal bar plot having distinct colour for each column and annotation for the same.

3) Pie Chart and Bubble Chart

```
# donut chart representing nationwide total confirmed, cured and deceased cases
group_size = [sum(state_data['Confirmed']),
              sum(state_data['Recovered']),
              sum(state_data['Deceased'])]

group_labels = ['Confirmed\n' + str(sum(state_data['Confirmed'])),
                'Recovered\n' + str(sum(state_data['Recovered'])),
                'Deceased\n'  + str(sum(state_data['Deceased']))]
custom_colors = ['skyblue','yellowgreen','tomato']

plt.figure(figsize = (5,5))
plt.pie(group_size, labels = group_labels, colors = custom_colors)
central_circle = plt.Circle((0,0), 0.5, color = 'white')
fig = plt.gcf()
fig.gca().add_artist(central_circle)
plt.rc('font', size = 12)
plt.title('Nationwide total Confirmed, Recovered and Deceased Cases', fontsize = 16)
plt.show()
```

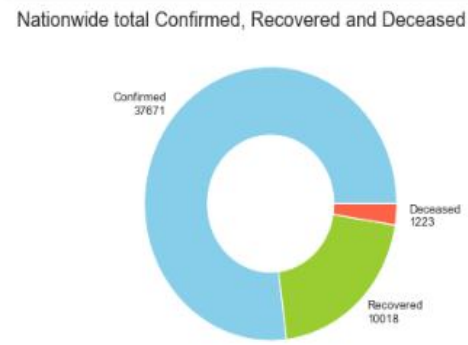


Fig.10 Pie/Donut Chart of Total Confirmed, Recovered and Deceased cases in India

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue V May 2020- Available at www.ijraset.com

The Above Pie chart gives the overall picture of India with total of 37671 Confirmed Covid cases, 10018 Recovered Cases and 1223 Deceased cases as per march 3rd 2020 data. Main objective is to reduce the gap between confirmed and recovered depicted in Pie Chart. The Bubble Chart gives assessment on how serious the situation is in the state, where x axis being Deceased cases to the Confirmed across states, the States having bigger bubble size indicate more successful recovery but higher on x and y indicates as more people are dying. Bubbles like Delhi and Tamil nadu signify High Confirmed Cases but low Death and large number of recovery , but States like Madhya Pradesh and Bihar indicate High Death and Confirmed cases but low recovery rate probably due to poverty and moderate health infrastructure . Outlier like Maharastra have high count for recovery,death and confirmed and Union territories like Goa have negligible reported cases.

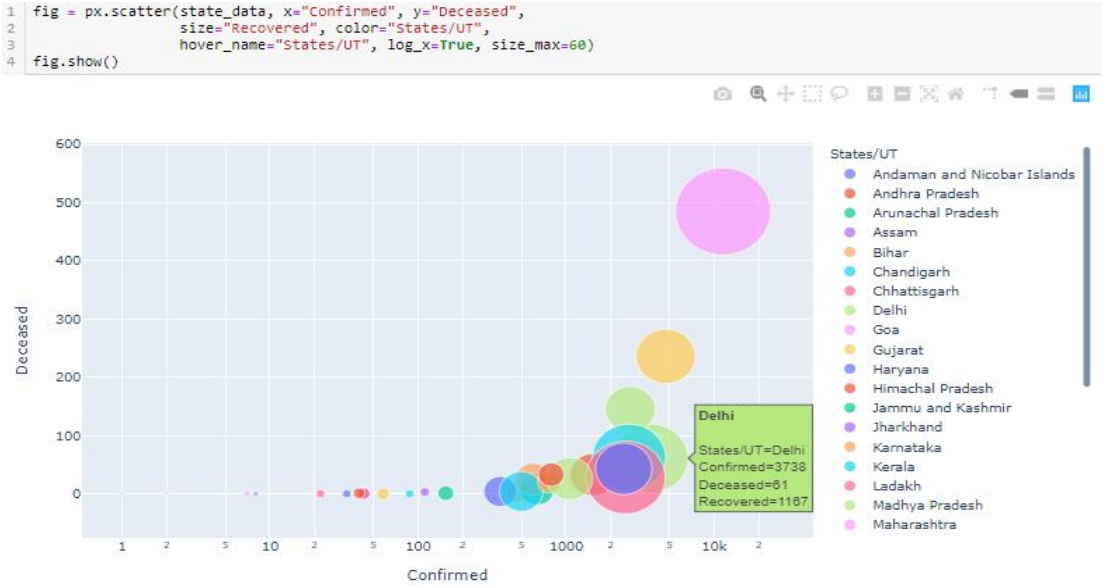


Fig.11 Bubble Chart ,x represent Deceased, y represent Confirmed and size of bubble as recovered , well annotated graph indicating sererity

4) *Scatter plot and Dot Plots:* The below is an example of scatter plot in Fig 12 states with corresponding Confirmed cases recordings.

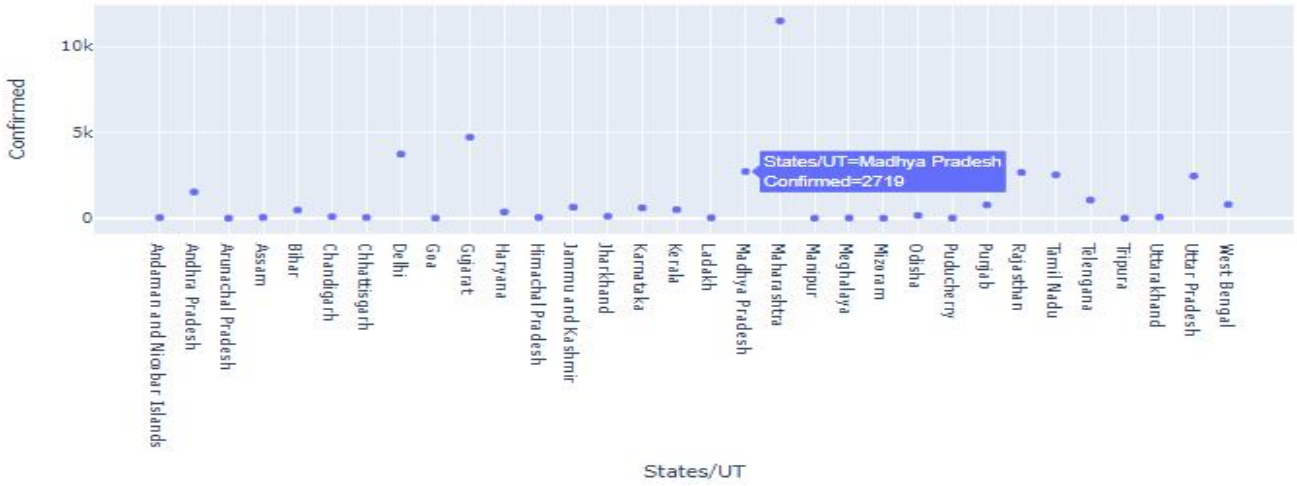


Fig.12 Scatter plot of Confirmed cases per state

Simpler representation for the bubble chart in Fig 11 can be given as scatter plot which is easier to represent without involving the recovery data for visualization given in below Fig 13. It surpising to see Oridsa performing well despite its poor economic conditions due to strict social distancing norms.

```
fig = px.scatter(data_frame=state_data,x='Confirmed', y='Deceased',hover_name='States/UT')
fig.show()
```

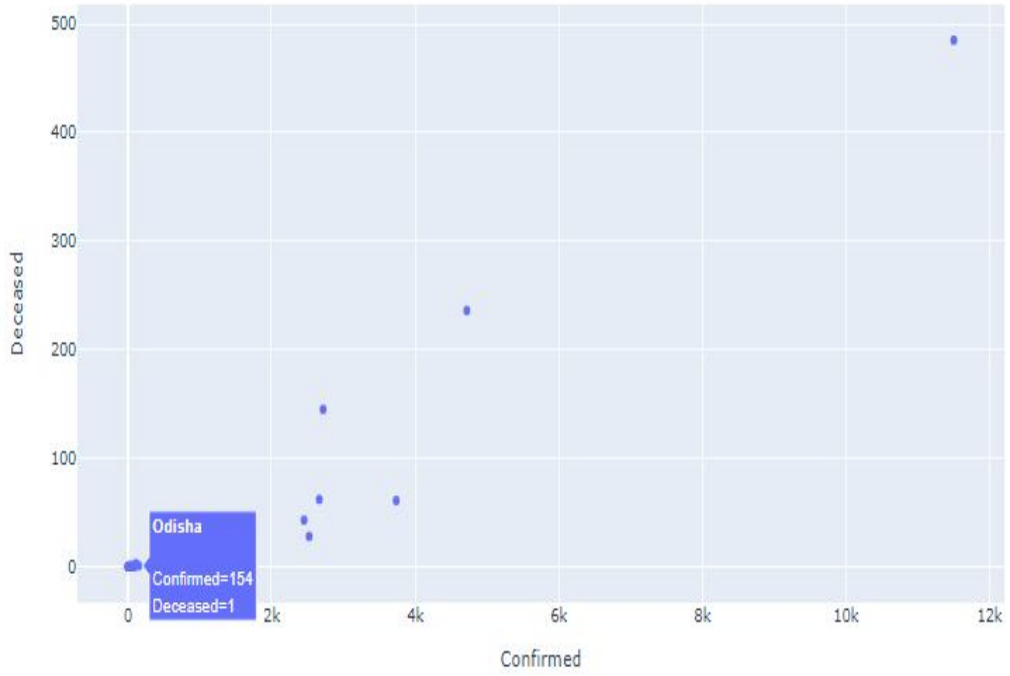


Fig.13 Simpler representation of Bubble Chart in Fig 11 without recovery dimension

The Dot plot in below Fig 14 show the variation and distribution of data points values for each state independently on Confirmed , Recovered and Deceased covid19 cases. Data visualization can also be done using combination of bar plot and scatter plot as fig 15 confirmed cases as bar plot and recovery as scatter plot. This helps to manage police and health care department deployment across state needs.

```
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=state_data['Confirmed'],
    y=state_data['States/UT'],
    marker=dict(color="crimson", size=12),
    mode="markers",
    name="Confirmed",
))

fig.add_trace(go.Scatter(
    x=state_data['Recovered'],
    y=state_data['States/UT'],
    marker=dict(color="blue", size=10),
    mode="markers",
    name="Recovered",
))

fig.add_trace(go.Scatter(
    x=state_data['Deceased'],
    y=state_data['States/UT'],
    marker=dict(color="pink", size=8),
    mode="markers",
    name="Deceased",
))

fig.show()
```

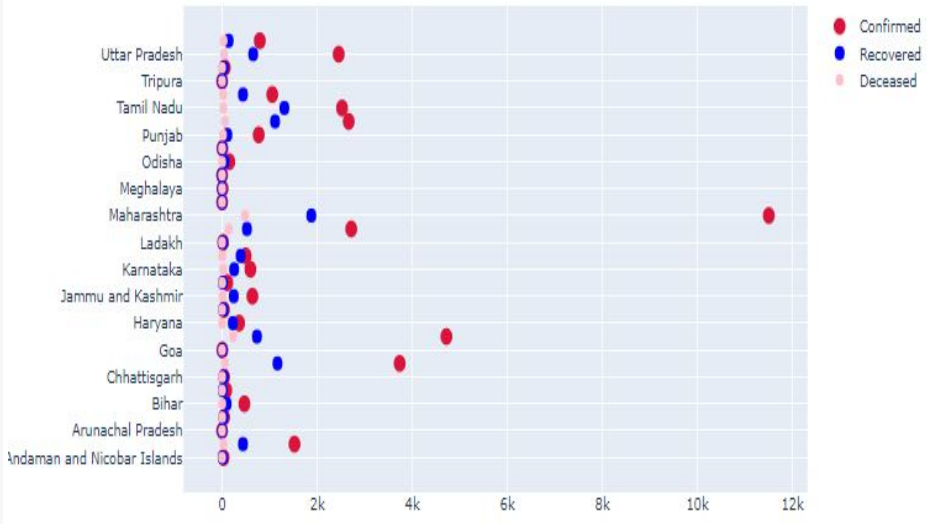
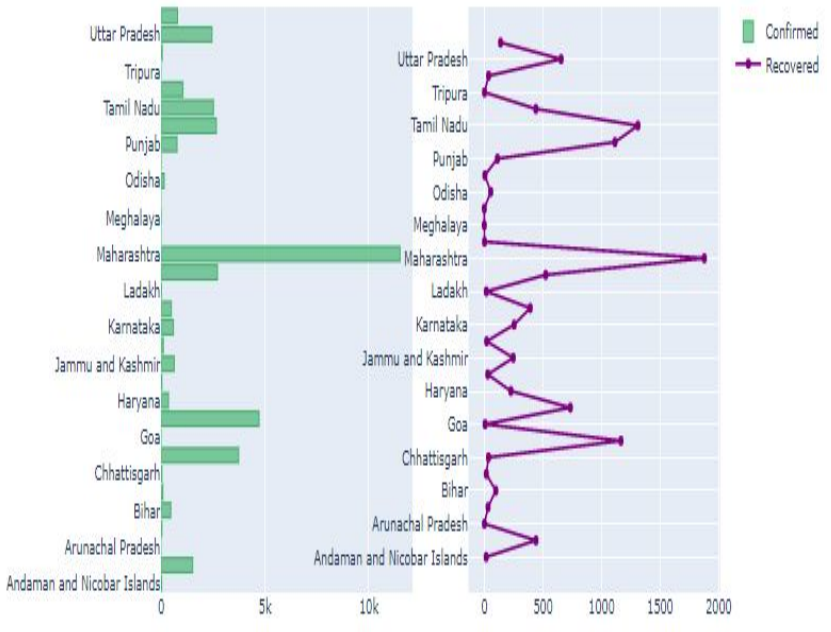


Fig.14 Dot plot Implementation (left) and visualization (right)




```
fig = make_subplots(rows=1, cols=2, specs=[[{}, {}]],
shared_xaxes=True,
                shared_yaxes=False, vertical_spacing=0.001)
fig.append_trace(go.Bar(
    x=state_data['Confirmed'],
    y=state_data['States/UT'],
    marker=dict(
        color='rgba(50, 171, 96, 0.6)',
        line=dict(
            color='rgba(50, 171, 96, 1.0)',
            width=1),
    ),
    name='Confirmed',
    orientation='h',
), 1, 1)
fig.append_trace(go.Scatter(
    x=state_data['Recovered'], y=state_data['States/UT'],
    mode='lines+markers',
    line_color='rgb(128, 0, 128)',
    name='Recovered',
), 1, 2)
fig.update_layout()
fig.show()
```

Fig.15 Combination of Bar plot and Line plot for Confirmed and Recovered values across state individually with implementation on right

5) *Box Plot and Gantt Charts:* [3]Box plot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points. The spacings between the different parts of the box indicate the degree of dispersion (spread) and skewness in the data, and show outliers

a) Elements of box Pot

i) Minimum: The lowest data point excluding any outliers. * Maximum : the largest data point excluding any outliers.

ii) Median (Q2 / 50th Percentile): the middle value of the dataset. * First quartile (Q1 / 25th Percentile) : also known as the lower quartile qn(0.25), is the median of the lower half of the dataset. * Third quartile (Q3 / 75th Percentile) : also known as the upper quartile qn(0.75), is the median of the upper half of the dataset. * Interquartile Range (IQR) : is the distance between the upper and lower quartile.

Box plot in Fig 16 depicts the variation of data of Confirmed Cases throughout India.

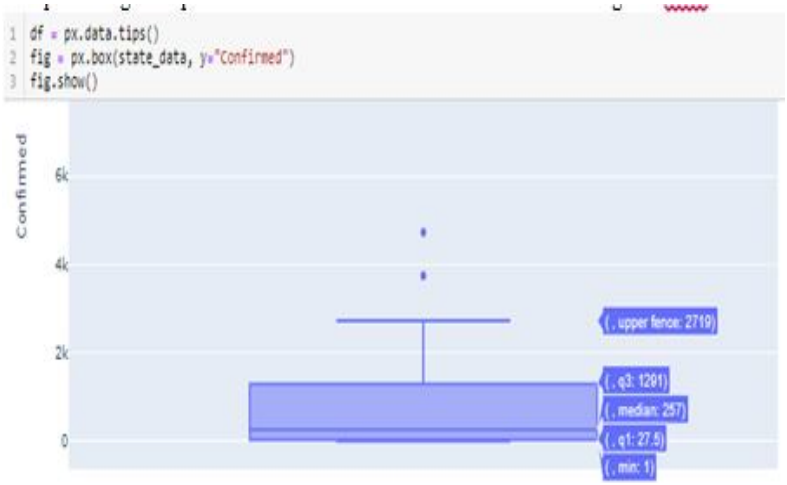


Fig.16 Box plot for variation in Confirmed Cases in India

A Gantt chart is a type of bar chart that is generally used illustrates a project schedule. The chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis. The width of the horizontal bars in the graph shows the duration of each activity. We have used the Gantt chart as Fig 17 to illustrate when was the first Corona virus case was reported in India in comparison to Mainland China, it is observed that china first officially reported case in Nov 2019 and has reported no new infection after mid April 2020 , which indicate that the infection is currently under control in China. But in India first case was Feb 2020 and still cases are being reported so India is still under the threat of coronavirus and next coming months are very crucial for India.

```
import plotly.figure_factory as ff

df = [dict(Task="Mainland China", Start='2019-11-22', Finish='2020-05-01', Complete=18),
      dict(Task="India", Start='2020-01-31', Finish='2020-05-03', Complete=17)]

fig = ff.create_gantt(df,colors='Blues', index_col='Complete', show_colorbar=True)
fig.show()
```

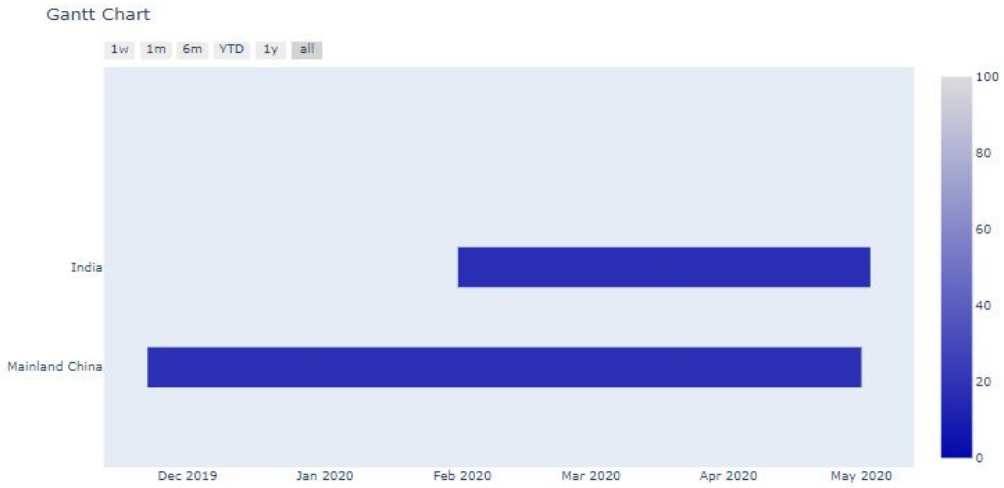


Fig.17 Gantt chart to figure the timeline of covid 19 in India in comparison to China

6) *Sunburst Charts and TreeMaps:* Sunburst Charts are quite interactive and well annotated to see cases from real-time collected data ,each state area is relatively proportionality compared to other states Confirmed data which is similar to donut/pie chart but gives more clarity on individual states reporting. TreeMaps also represent relatively area as states but is divided as hierarchical tree structure than pie structure given in Fig 19.

```
import plotly.express as px
import numpy as np

fig = px.sunburst(state_data, path=['States/UT'], values='Confirmed',
                  color='Confirmed', hover_data=['States/UT'],
                  color_continuous_scale='RdBu')
fig.show()
```

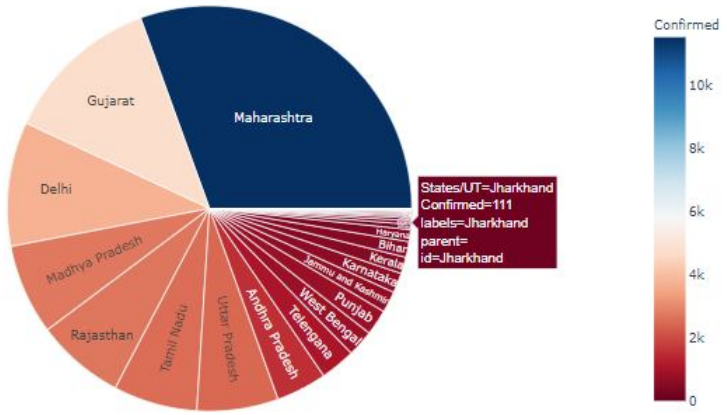


Fig.18 Sunburst Chat with high interactive capabilities.

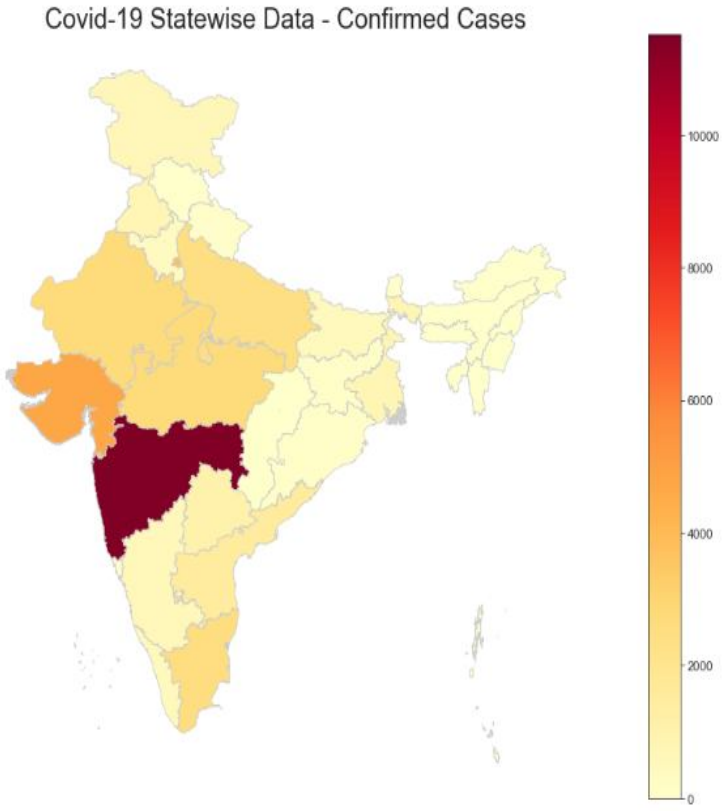
```
import plotly.express as px
fig = px.treemap(state_data, path=['States/UT'], values='Confirmed', height=600, width=1000)

fig.update_layout(
    title_x = 0.5,
    geo=dict(
        showframe = False,
        showcoastlines = False,
    ))

fig.show()
```

Fig.19 Tree Map of States Data with annotations

7) *Choropleth Maps:* Lastly we depict the situation of coronavirus using choropleth Maps using the confirmed cases across states plotted on India map downloaded from GeoFrame , choropleth map are immensely used in Mass Media reporting due to the power of collective representation of situation and easier identification using gradient colouring. The above Plots can be used for other parameters such as recovered and deaths to better and in depth assessments of the situation which can help both the goverments, healthcare workers and police officers for getting overview of status.




```
map_data = gpd.read_file('Indian_States.shp')
map_data.rename(columns = {'st_nm':'States/UT'}, inplace = True)
map_data.head()
map_data['States/UT'] = map_data['States/UT'].str.replace('&',
'and')
map_data['States/UT'].replace('Arunanchal Pradesh',
 'Arunachal Pradesh', inplace = True)
map_data['States/UT'].replace('Telangana', 'Telengana', inplace =
True)
map_data['States/UT'].replace('NCT of Delhi', 'Delhi', inplace =
True)
merged_data = pd.merge(map_data, state_data, how = 'left',
on = 'States/UT')
merged_data.fillna(0, inplace = True)
merged_data.drop('Sr.No', axis = 1, inplace = True)
merged_data.head()
fig, ax = plt.subplots(1, figsize=(20, 12))
ax.axis('off')
ax.set_title('Covid-19 Statewise Data - Confirmed Cases',
fontdict = {'fontsize': '25', 'fontweight' : '3'})
# plot the figure

merged_data.plot(column = 'Confirmed', cmap='YlOrRd',

linewidth=0.8, ax=ax, edgecolor='0.8', legend = True)

plt.show()
```

Fig.21 Implementation (right) and Visualization (left) of choropleth maps for India states with respect to confirmed cases

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue V May 2020- Available at www.ijraset.com

IV. FORECASTING CONFIRMED CASES IN INDIA

A. Time Series DataSet From Kaggle: https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset

We would require Timeseries data for Forecasting models, using the above public data covid_19_data.csv for analysing and working on forecasting models for India in consideration. The csv has the following columns

SNo, ObservationDate, Province/State, Country/Region, Last Update, Confirmed (cumulative),Deaths (cumulative) and Recovered (cumulative)

B. Loading and Handling Time Series in Pandas

1) Importing packages: The below packages in Fig was imported for forecasting segment of the paper implementation.

In [1]:
```
##https://www.kaggle.com/neelkudu28/covid-19-data-analysis-forecasting-for-india
## conda install -c conda-forge fbprophet
import requests, pandas as pd, numpy as np
from pandas import DataFrame
from io import StringIO
import time, json
from datetime import date
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20, 20
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import datetime as dt
from datetime import timedelta
from statsmodels.tsa.api import Holt,SimpleExpSmoothing,ExponentialSmoothing
from sklearn.metrics import mean_squared_error,r2_score
import statsmodels.api as sm
from fbprophet import Prophet
```

Fig.22 Packages Imported for Forecasting

2) Reading CSV file as Pandas and extracting Data particular to India: The Kaggle Dataset covid_19_data in csv format is read as an Pandas DataFrame as given below. Since our research is exclusive to Country India we are filtering the Data based on Country/Region as “India”, named as covid_india in Fig23.

```
covid=pd.read_csv("covid_19_data.csv")
covid.head()
```

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
0	1	01/22/2020	Anhui	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
1	2	01/22/2020	Beijing	Mainland China	1/22/2020 17:00	14.0	0.0	0.0
2	3	01/22/2020	Chongqing	Mainland China	1/22/2020 17:00	6.0	0.0	0.0
3	4	01/22/2020	Fujian	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
4	5	01/22/2020	Gansu	Mainland China	1/22/2020 17:00	0.0	0.0	0.0

```
#Extracting India's data
covid_india=covid[covid['Country/Region']=="India"]
covid_india.head()
```

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
430	431	01/30/2020	NaN	India	1/30/20 16:00	1.0	0.0	0.0
491	492	01/31/2020	NaN	India	1/31/2020 23:59	1.0	0.0	0.0
547	548	02/01/2020	NaN	India	1/31/2020 8:15	1.0	0.0	0.0
607	608	02/02/2020	NaN	India	2020-02-02T06:03:08	2.0	0.0	0.0
672	673	02/03/2020	NaN	India	2020-02-03T21:43:02	3.0	0.0	0.0

Fig.23 Reading and Extracting India related Data

3) *Converting dataframe into useable time series format for processing:* Since the Columns Confirmed, Deaths and Recovered are cumulative values based on ObservationDate column , there are occurrences of repeating ObservationDate values which is intuitive as cases can be reported Multiple times a day, thus to make it to useable format, we are grouping Confirmed, Death and Recovered Column values based on ObservationDate which in turn is converted to datetime format using pd.to_datetime() function. The resulting dataFrame is named india_datewise which is the finalized dataframe which will be used throughout the project for forecasting models.

```
#Converting the date into Datetime format
covid_india["ObservationDate"]=pd.to_datetime(covid_india["ObservationDate"])
covid_india.head()
```

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
430	431	2020-01-30	NaN	India	1/30/20 16:00	1.0	0.0	0.0
491	492	2020-01-31	NaN	India	1/31/2020 23:59	1.0	0.0	0.0
547	548	2020-02-01	NaN	India	1/31/2020 8:15	1.0	0.0	0.0
607	608	2020-02-02	NaN	India	2020-02-02T06:03:08	2.0	0.0	0.0
672	673	2020-02-03	NaN	India	2020-02-03T21:43:02	3.0	0.0	0.0

```
#Grouping the data based on the Date
india_datewise=covid_india.groupby(["ObservationDate"]).agg({"Confirmed":'sum',"Recovered":'sum',"Deaths":'sum'})
india_datewise.head()
```

ObservationDate	Confirmed	Recovered	Deaths
2020-01-30	1.0	0.0	0.0
2020-01-31	1.0	0.0	0.0
2020-02-01	1.0	0.0	0.0
2020-02-02	2.0	0.0	0.0
2020-02-03	3.0	0.0	0.0

Fig.24 india_datewise Dataframe for forecasting models

4) *Understanding Distribution of Timeseries Data particular to a Country (India):* The below is Representation of india_datewise overview , with Number of Total Confirmed , Recovered and Death Cases. We also can deduce Approximate number of Confirmed/Recovered/Death Cases per day and last day. Similarly Active Cases and Closed cases deduced can visualized using bar plot in Fig 26 and 27

```
#Adding week column to perfom weekly analysis further ahead
india_datewise["WeekofYear"]=india_datewise.index.weekofyear
```

```
#Adding week column to perfom weekly analysis further ahead
india_datewise["WeekofYear"]=india_datewise.index.weekofyear
```

```
india_datewise["Days Since"]=(india_datewise.index-india_datewise.index[0])
india_datewise["Days Since"]=india_datewise["Days Since"].dt.days
```

```
print("Number of Confirmed Cases",india_datewise["Confirmed"].iloc[-1])
print("Number of Recovered Cases",india_datewise["Recovered"].iloc[-1])
print("Number of Death Cases",india_datewise["Deaths"].iloc[-1])
print("Number of Active Cases",india_datewise["Confirmed"].iloc[-1]-india_datewise["Recovered"].iloc[-1]
                                                                -india_datewise["Deaths"].iloc[-1])
print("Number of Closed Cases",india_datewise["Recovered"].iloc[-1]+india_datewise["Deaths"].iloc[-1])
print("Approximate Number of Confirmed Cases per day",round(india_datewise["Confirmed"].iloc[-1]/india_datewise.shape[0]))
print("Approximate Number of Recovered Cases per day",round(india_datewise["Recovered"].iloc[-1]/india_datewise.shape[0]))
print("Approximate Number of Death Cases per day",round(india_datewise["Deaths"].iloc[-1]/india_datewise.shape[0]))
print("Number of New Cofirmed Cases in last 24 hours are",india_datewise["Confirmed"].iloc[-1]
                                                            -india_datewise["Confirmed"].iloc[-2])
print("Number of New Recoverd Cases in last 24 hours are",india_datewise["Recovered"].iloc[-1]
                                                            -india_datewise["Recovered"].iloc[-2])
print("Number of New Death Cases in last 24 hours are",india_datewise["Deaths"].iloc[-1]
                                                            -india_datewise["Deaths"].iloc[-2])
```

```
Number of Confirmed Cases 37257.0
Number of Recovered Cases 10007.0
Number of Death Cases 1223.0
Number of Active Cases 26027.0
Number of Closed Cases 11230.0
Approximate Number of Confirmed Cases per day 401.0
Approximate Number of Recovered Cases per day 108.0
Approximate Number of Death Cases per day 13.0
Number of New Cofirmed Cases in last 24 hours are 2394.0
Number of New Recoverd Cases in last 24 hours are 939.0
Number of New Death Cases in last 24 hours are 69.0
```

Fig.25 Metadata of DataFrame

The Distribution of the india_datawise dataframe is visualized in Fig 26 and 27. Fig 26 depicts the Distribution of Number of Active Cases in India , that is difference of Confirmed cases to recovered cases. Fig 27 depicts the Distribution of Number of Closed Cases in India , that is the difference of Recovered cases to Number of Deaths.

```
plt.figure(figsize=(15,5))
sns.barplot(x=india_datewise.index.date,y=india_datewise["Confirmed"]-india_datewise["Recovered"]-india_datewise["Deaths"])
plt.xticks(rotation=90)
plt.ylabel("Number of Cases")
plt.xlabel("Date")
plt.title("Distribution of Number of Active Cases in India over Date")
```

```
Text(0.5, 1.0, 'Distribution of Number of Active Cases in India over Date')
```

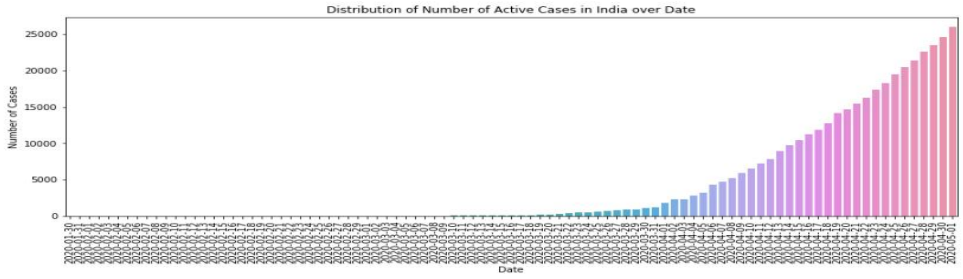


Fig.26 Distribution of Number of Active Cases in India over Date

```
plt.figure(figsize=(15,5))
sns.barplot(x=india_datewise.index.date,y=india_datewise["Recovered"]+india_datewise["Deaths"])
plt.xticks(rotation=90)
plt.ylabel("Number of Cases")
plt.xlabel("Date")
plt.title("Distribution of Number of Closed Cases in India over Date")
```

```
ext(0.5, 1.0, 'Distribution of Number of Closed Cases in India over Date')
```

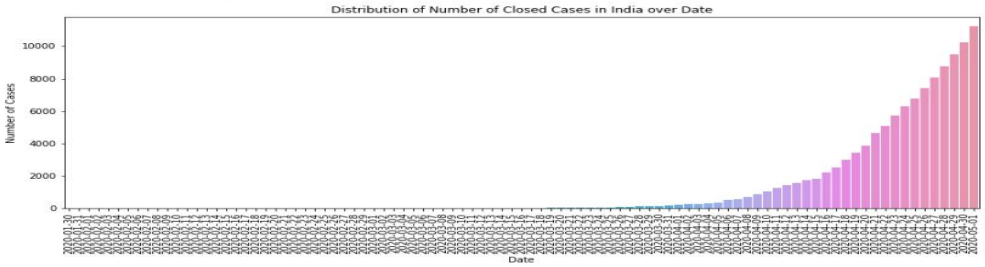
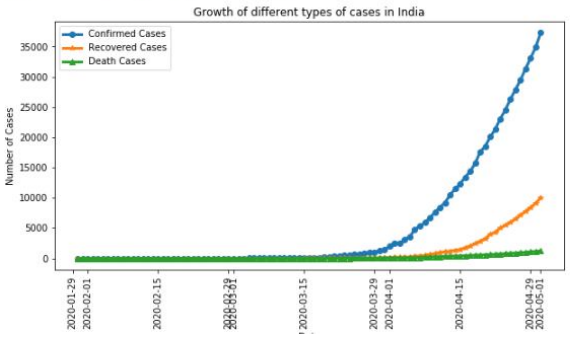


Fig.27 Distribution of Number of Closed Cases in India over Date

5) The Growth of Confirmed Cases Line plot to be forecasted

```
plt.figure(figsize=(10,5))
plt.plot(india_datewise["Confirmed"],label="Confirmed Cases",marker='o',linewidth=3)
plt.plot(india_datewise["Recovered"],label="Recovered Cases",marker='*',linewidth=3)
plt.plot(india_datewise["Deaths"],label="Death Cases",marker="^",linewidth=3)
plt.xticks(rotation=90)
plt.ylabel("Number of Cases")
plt.xlabel("Date")
plt.title("Growth of different types of cases in India")
plt.legend()
```

```
matplotlib.legend.Legend at 0x13e1b3a0dd8>
```




```
plt.rcParams['figure.figsize'] = 5, 5
plt.plot(india_datewise["Confirmed"],marker='o')
```

```
[<matplotlib.lines.Line2D at 0x1bff0e97c88>]
```

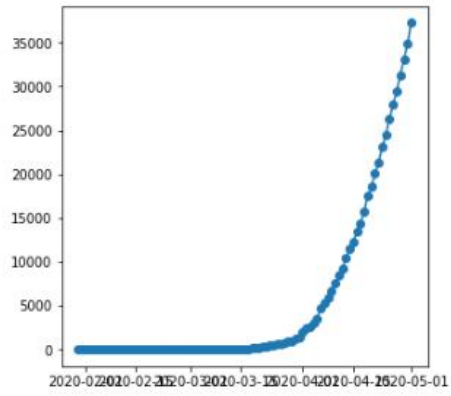


Fig.28 Line plot of Growth of Confirmed,Death and Recovered cases in India across date (left) and Line plot of Only Confirmed cases in India across date(right)

Fig 28 (left) is the line plot of time series data of india_datawise dataframe prepared in above step 3 of Confirmed Cases in india across date. This will be used for forecasting/predicting the trajectory of the graph for next few sequence of dates which can be used by governments to access and pre-emptively take decisive decisions to curb the exponential growth. The same technique can be used across countries and with other parameters such as Active cases, closed cases , recovered etc. For the scope of this paper we will focus on Fig 28 (left) forecasting Confirmed cases that are likely to be reported and understand the stance of country in battling the coronavirus.

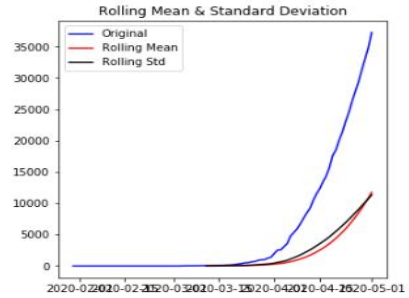
C. Implementation of Forecasting Models

1) ARIMA Model

a) Step 1: Checking Stationarity in time series Data. [6] A TS is said to be stationary if its statistical properties such as mean, variance remain constant over time. Most of the TS models work on the assumption that the TS is stationary. Intuitively, we can sat that if a TS has a particular behaviour over time, there is a very high probability that it will follow the same in the future. Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series. Stationarity is defined using very strict criterion. However, for practical purposes we can assume the series to be stationary if it has constant statistical properties over time, ie. constant mean, constant variance and autocovariance that does not depend on time. We are using Dickey-Fuller Test to check for Stationarity, This is one of the statistical tests for checking stationarity. Here the null hypothesis is that the TS is non-stationary. The test results comprise of a Test Statistic and some Critical Values for difference confidence levels. If the 'Test Statistic' is less than the 'Critical Value', we can reject the null hypothesis and say that the series is stationary.

```
def test_stationarity(timeseries):
    #Determing rolling statistics
    rcParams['figure.figsize'] = 5, 5
    rolmean = timeseries.rolling(window=42,center=False).mean()
    rolstd = timeseries.rolling(window=42,center=False).std()
    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic',
            'p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

ts_month=india_datewise["Confirmed"]
test_stationarity(ts_month)
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -0.154594
p-value                          0.943722
#Lags Used                      12.000000
Number of Observations Used     80.000000
Critical Value (1%)             -3.514869
Critical Value (5%)             -2.898409
Critical Value (10%)            -2.586439
dtype: float64
```

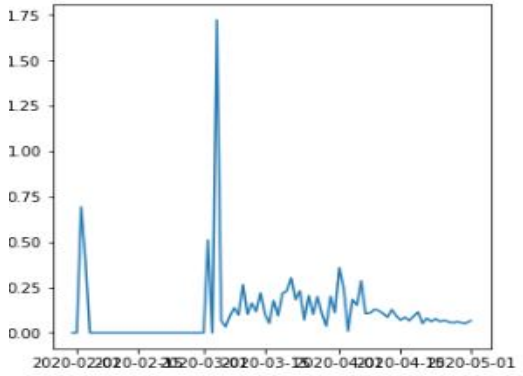
Fig.29 Function to find Stationarity of Timeseries (left) and Output (right)

From Fig 29 we are using Dickey- fuller test to test the stationarity of Time series, from P-value being very high it is evident that the time series Confirmed across date is not stationary and need to be transformed for stationarity.

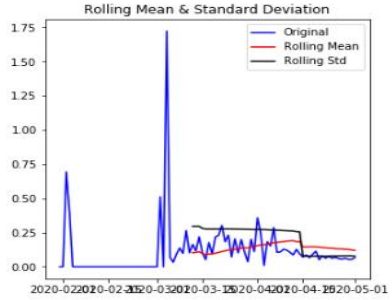
b) Step 2: Transforming time series Data Confirmed Cases across dates for Stationarity. We need to reduce trend (varying mean over time) , this can be significantly reduced by log transformation. The below Fig 30 shows how the log transform on time series data Confirmed cases across date would be transformed by reducing trend. Then we apply the Dickey fuller test by the function as Fig 29 (left) , as p value is very low value , this transformed time series can be used by arima model for forecasting.

```
ts_month_log = np.log(ts_month)
ts_month_log_diff = ts_month_log - ts_month_log.shift()
plt.plot(ts_month_log_diff)
```

```
<matplotlib.lines.Line2D at 0x1bff31ab940>]
```




```
ts_month_log_diff.dropna(inplace=True)
test_stationarity(ts_month_log_diff)
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -5.393147
p-value                          0.000004
#Lags Used                       1.000000
Number of Observations Used     90.000000
Critical Value (1%)             -3.505190
Critical Value (5%)             -2.894232
Critical Value (10%)            -2.584210
dtype: float64
```

Fig.30 Making Timeseries Stationary by reducing Trend(left) and DickeyFuller Test Results(right)

c) *Step 3:* Determining p and q value for ARIMA model using ACF and PACF [6]ARIMA stands for Auto-Regressive Integrated Moving Averages. The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. The predictors depend on the parameters (p,d,q) of the ARIMA model:

i) Number of AR (Auto-Regressive) terms (p): AR terms are just lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).

ii) Number of MA (Moving Average) terms (q): MA terms are lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.

iii) Number of Differences (d): These are the number of nonseasonal differences, i.e. in this case we took the first order difference. So either we can pass that variable and put d=0 or pass the original variable and put d=1. Both will generate same results.

An importance concern here is how to determine the value of 'p' and 'q'.

- Autocorrelation Function (ACF): It is a measure of the correlation between the the TS with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant 't1'…'t2' with series at instant 't1-5'…'t2-5' (t1-5 and t2 being end points).
- Partial Autocorrelation Function (PACF): This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Eg at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

In below plot Fig 31 the two dotted lines on either sides of 0 are the confidence intervals. These can be used to determine the 'p' and 'q' values as:

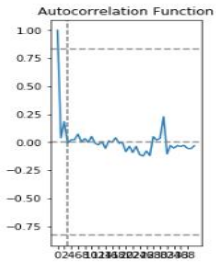
p – The lag value where the PACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=3.

q – The lag value where the ACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=3.

These values as p=3 and q=3 will be used for ARIMA model parameters.

```
lag_acf = acf(ts_month_log_diff)
#Plot ACF:
plt.subplot(121)
plt.rcParams['figure.figsize'] = 5, 5
plt.xticks(np.arange(0, 40,2))
plt.axvline(x=3,linestyle='--',color='gray')
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-7.96/np.sqrt(len(ts_month_log_diff))
            ,linestyle='--',color='gray')
plt.axhline(y=7.96/np.sqrt(len(ts_month_log_diff))
            ,linestyle='--',color='gray')
plt.title('Autocorrelation Function')
```

```
Text(0.5, 1.0, 'Autocorrelation Function')
```




```
lag_pacf = pacf(ts_month_log_diff, method='ols')
#Plot PACF:
plt.subplot(122)
plt.rcParams['figure.figsize'] = 5, 5
plt.plot(lag_pacf)
plt.xticks(np.arange(0, 40,2))
plt.axhline(y=0,linestyle='--',color='gray')
plt.axvline(x=3,linestyle='--',color='gray')
plt.axhline(y=-7.96/np.sqrt(len(ts_month_log_diff))
            ,linestyle='--',color='gray')
plt.axhline(y=7.96/np.sqrt(len(ts_month_log_diff))
            ,linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```

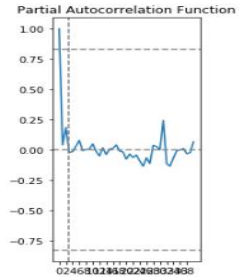


Fig.31 ACF(left) and PACF(right) for p,q values for ARIMA model

d) *Step 4:* Using ARIMA model for forecasting actual Timeseries (Confirmed cases across date in India)

```
size = int(len(ts_month_log) - 5)
train, test = ts_month_log[0:size], ts_month_log[size:len(ts_month_log)]
history = [x for x in train]
predictions = list()
print('Printing Predicted vs Expected Values...')
print('\n')
for t in range(len(test)):
    model = ARIMA(history, order=(3,1,3))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(float(yhat))
    obs = test[t]
    history.append(obs)
print('predicted=%f, expected=%f' % (np.exp(yhat), np.exp(obs)))
```

```
Printing Predicted vs Expected Values...


predicted=38502.817445, expected=37257.000000
```

```
didx = pd.DatetimeIndex(start ='2020-04-28', freq ='D',periods = len(z1))
didx
```

```
DatetimeIndex(['2020-04-28', '2020-04-29', '2020-04-30', '2020-05-01',
               '2020-05-02'],
              dtype='datetime64[ns]', freq='D')
```

```
z1=pd.DataFrame({'Pred':predictions}).apply(np.exp)
z1.index=didx
round(z1)
```

	Pred
2020-04-28	30961.0
2020-04-29	32658.0
2020-04-30	34633.0
2020-05-01	36560.0
2020-05-02	38503.0

```
error = mean_squared_error(test, predictions)
print('\n')
print('Printing Mean Squared Error of Predictions...')
print('Test MSE: %.6f' % error)
predictions_series = pd.Series(predictions, index = test.index)
```

```
Printing Mean Squared Error of Predictions...
Test MSE: 0.001947
```

Fig.32 ARIMA model(left) and collecting Predicted value after Exponential Transformation (right)

We use the provide timeseries data Confirmed Cases across date in India and parameters p,q on fitting ARIMA model. This model is then is used for forecasting in our case next 5 consecutive dates Predicted Confirmed Cases, the predicted is in log scale which is inverse transformed using Exponential Transform with date index as subsequent date to provided data using pd.DatatimeIndex() function. We get around 0.001947 RMSE value which means the error is negligible.

In below Fig 33 we plotted the predicted next 5 Confirmed values over the corresponding future dated

```
plt.plot(ts_month[:'2020-04-28'])
plt.rcParams['figure.figsize'] = 5, 5
plt.plot(z1)
plt.xticks(rotation=90)
```

```
(array([737456., 737470., 737485., 737499., 737516., 737530., 737546.]),
 <a list of 7 Text xticklabel objects>)
```

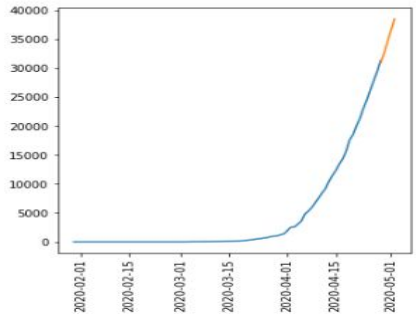


Fig.33 Forecasting using ARIMA model predicted values

2) *Holt winter Model:* Forecasting can be done using Holt winder model instead of ARIMA model which is much simpler in terms of implementation as given below and has almost same accuracy than ARIMA

```
model_train=india_datewise.iloc[:int(india_datewise.shape[0]*0.95)]
#model_train=india_datewise.iloc[:]
valid=india_datewise.iloc[int(india_datewise.shape[0]*0.95):]
y_pred=valid.copy()
```

```
holt=Holt(np.asarray(model_train["Confirmed"])).fit(smoothing_level=0.4, smoothing_slope=1.7)
```

```
y_pred["Holt"]=holt.forecast(len(valid))
rmse_holt_linear=np.sqrt(mean_squared_error(y_pred["Confirmed"],y_pred["Holt"]))
model_scores=[]
model_scores.append(rmse_holt_linear)
print("Root Mean Square Error Holt's Linear Model: ",rmse_holt_linear)
```

```
Root Mean Square Error Holt's Linear Model:  252.75944528927562
```

Fig.34 Implementing Holt Winter model for Confirmed cases across date

```
plt.figure(figsize=(10,5))
plt.plot(model_train.Confirmed,label="Train Set",marker='o')
valid.Confirmed.plot(label="Validation Set",marker='*')
y_pred.Holt.plot(label="Holt's Linear Model Predicted Set",marker='^')
plt.ylabel("Confirmed Cases")
plt.xlabel("Date Time")
plt.title("Confirmed Cases Holt's Linear Model Prediction")
plt.xticks(rotation=90)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x13e1d7abf60>
```

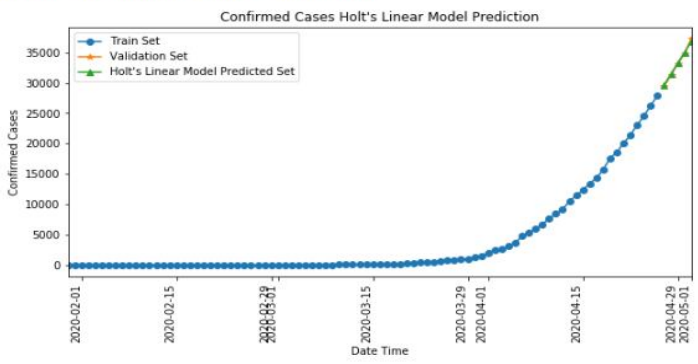


Fig.35 Forecasted values for future dates using Holt winter model

3) Fbprophet from Facebook: These are newer API provided by Facebook for Automating the process of Forecasting Timeseries data which is very easy to use and provide good accuracy results.

```
from fbprophet import Prophet
prophet_c=Prophet(interval_width=0.95,weekly_seasonality=True,)
prophet_confirmed=pd.DataFrame(zip(list(india_datewise.index),list(india_datewise["Confirmed"])),columns=['ds','y'])
prophet_c.fit(prophet_confirmed)
forecast_c=prophet_c.make_future_dataframe(periods=17)
forecast_confirmed=forecast_c.copy()
confirmed_forecast=prophet_c.predict(forecast_c)
print(prophet_c.plot(confirmed_forecast))
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
Figure(720x432)
```

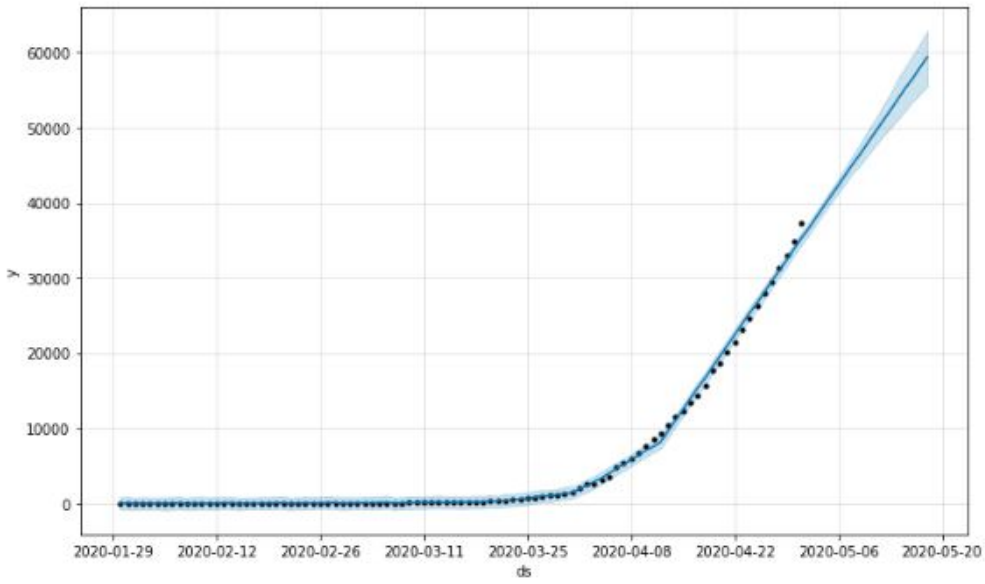


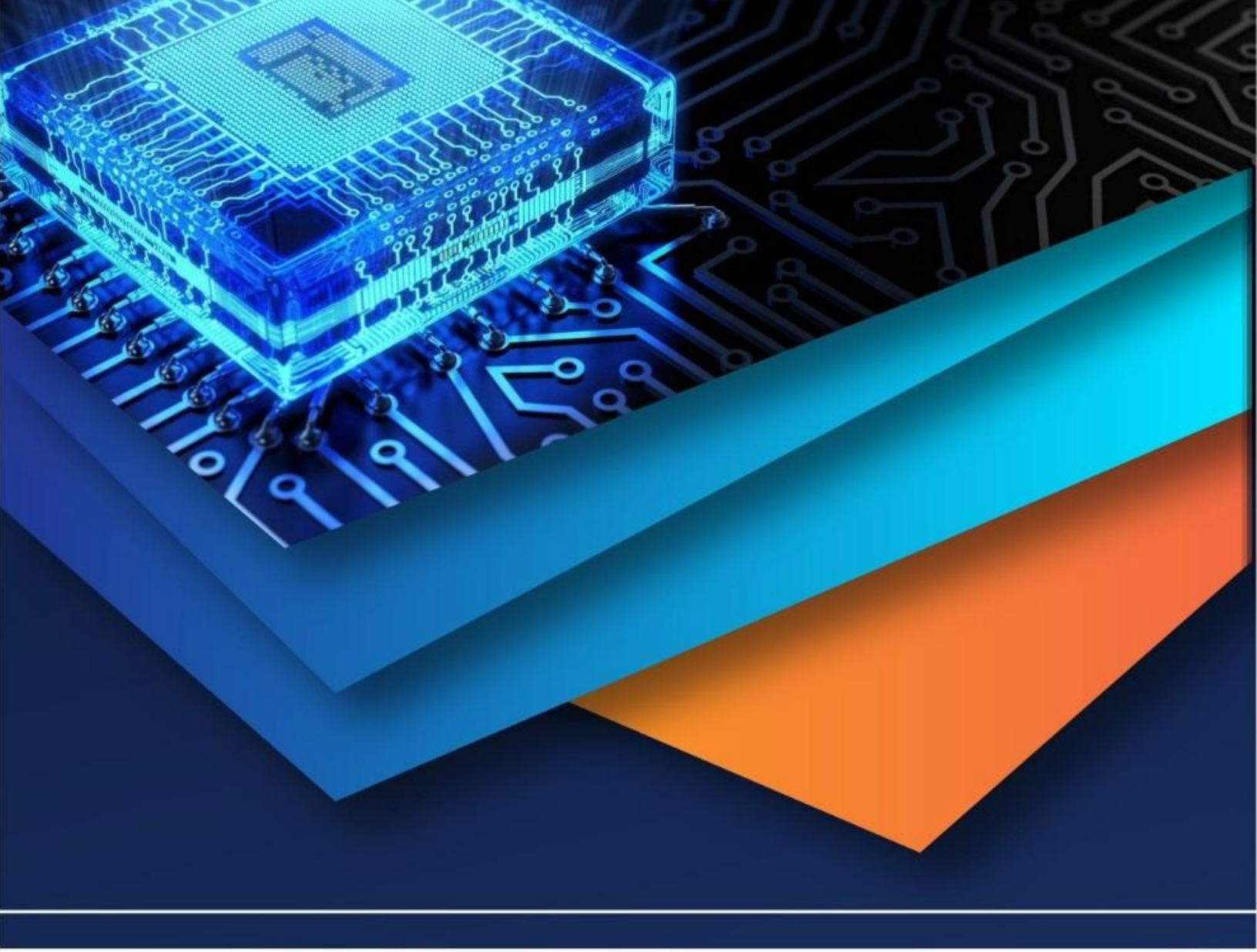
Fig.36 Forecasted values using Facebook fbprophet API

V. APPLICATIONS OF METHODOLOGY

The proposed system is capable of Tracking Coronavirus infection spread in depth for the concerned country using Highly intuitive Data Visualization through plotly and matplotlib to deduce important insights within the country states/provinces also the paper presents various forecasting techniques which can be used by Governments , health workers and police department to better monitor and take preemptive decisions to curb the spread of infections

REFERENCES

[1] https://www.health.harvard.edu/diseases-and-conditions/preventing-the-spread-of-the-coronavirus
[2] https://www.gavi.org/vaccineswork/why-coronavirus-lockdown-necessary
[3] https://en.wikipedia.org/wiki/Box_plot
[4] https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset#covid_19_data.csv.
[5] https://intellipaat.com/community/47361/how-to-learn-web-scraping
[6] AARSHAY JAIN "A comprehensive beginner's guide to create a Time Series Forecast (with Codes in Python and R)" https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
[7] Saurav Mishra "nCov - A Geo Statistical Analysis" https://www.kaggle.com/sauravmishra1710/covid-19-sars-cov-2-a-statistical-analysis



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY