COVID-19 Analysis, Visualization and Forecasting

Data Science Intensive Capstone Project I

Thanks to Springboard Mentor
Ash Yousefi

Reeti Bhagat
bhagat.reeti@gmail.com

Table of Contents

COVID-19 Analysis, Visualization and Predictions

Introduction to Covid-19

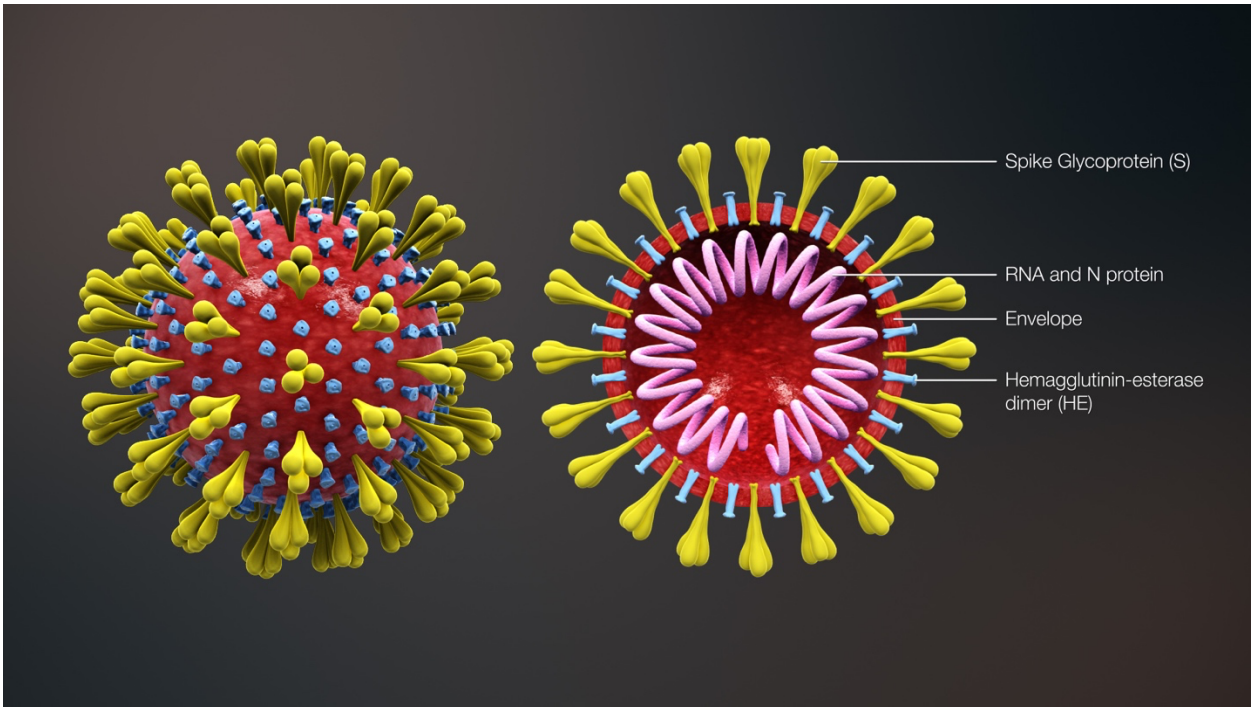


Figure 1 Covid-19 (Image Credits: Scientific Animations under a CC BY-SA 4.0 license)

Coronavirus is a family of viruses that can cause illness, which can vary from *common cold* and *cough* to sometimes more severe disease. **Middle East Respiratory Syndrome (MERS-CoV)** and **Severe Acute Respiratory Syndrome (SARS-CoV)** were such severe cases with the world already has faced.

SARS-CoV-2 (n-coronavirus) is the new virus of the coronavirus family, which first *discovered* in 2019, which has not been identified in humans before. It is a *contiguous* virus which started from **Wuhan** in **December 2019**. Which later declared as **Pandemic** by **WHO** due to high rate spreads throughout the world. Currently (on the date 29 Aug 2020), this leads to a total of *900K+ Deaths* across the globe.

Pandemic is spreading all over the world; it becomes more important to understand about this spread. This is an effort to analyze the cumulative data of confirmed, deaths, and recovered cases over time. In this notebook, the main focus is to analyze the spread trend of this virus all over the world and its predictions.

Sourcing and loading data

2019 Novel Coronavirus COVID-19 (2019-nCoV) Data Repository by Johns Hopkins CSSE (LINK)

Dataset consists of time-series data from 22 JAN 2020 to Till AUG 8,2020.

New Time-series dataset

- Time_series_covid19_confirmed_global.csv (Link Raw File)
- Time_series_covid19_deaths_global (Link Raw File)

Downloading and Installing Prerequisite

Install:

- pip install pycountry_convert
- pip install folium
- pip install calmap
- pip install altair
- pip install prophet==0.6
- pip install pmdarima

Imports and Datasets

- Pandas - for dataset handling
- NumPy - Support for Pandas and calculations
- Matplotlib - for visualization (Plotting graphs)
- pycountry_convert - Library for getting continent (name) to from their country names
- folium - Library for Map
- pmdarima - Prediction Models
- plotly - for interactive plots
- altair- for visualization

Loading the raw csv dataset and reading the data

```
3]:
# Retriving Dataset
df_confirmed = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/css
df_deaths = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_c

# Depricated
df_recovered = pd.read_csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/css
```

```
:
df_confirmed.head()
```

:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	8/19/20	8/20/20	8/21/20	8/22/20	8/23/20
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	37599	37856	37894	37953	37999
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	7812	7967	8119	8275	8427
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	39847	40258	40667	41068	41460
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	1024	1024	1045	1045	1045
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	2015	2044	2068	2134	2171

Data Pre-processing

The main reason behind data processing is that **data almost never comes in a form that is ready for us**. And in my personal experience, a large amount of time spent on a data science project is on manipulating data.

Data processing runs the following steps:

1. Changing the country names as required by pycountry_convert Lib.
2. Melt and merge the raw confirmed, deaths and recovered CSV data into one Data Frame.
3. performs the data cleanings due to missing values, wrong datatypes
4. Data Aggregation: Add an active case column *Active*, which is calculated by active case = confirmed — deaths — recovered. Aggregate data into Country/Region wise and group them by Date and Country/Region. After that, add day wise *New cases*, *New deaths* and *New recovered* by deducting the corresponding cumulative data on the previous day.

Changing the country names as required by pycountry_convert Lib.

```
3]:
df_confirmed = df_confirmed.rename(columns={"Province/State":"state","Country/Region": "country"})
df_deaths = df_deaths.rename(columns={"Province/State":"state","Country/Region": "country"})
df_recovered = df_recovered.rename(columns={"Province/State":"state","Country/Region": "country"})

## Changing the conuntry names as required by pycountry_convert Lib

df_confirmed.loc[df_confirmed['country'] == "US", "country"] = "USA"
df_deaths.loc[df_deaths['country'] == "US", "country"] = "USA"
df_recovered.loc[df_recovered['country'] == "US", "country"] = "USA"

df_confirmed.loc[df_confirmed['country'] == 'Korea, South', "country"] = 'South Korea'
df_deaths.loc[df_deaths['country'] == 'Korea, South', "country"] = 'South Korea'
df_recovered.loc[df_recovered['country'] == 'Korea, South', "country"] = 'South Korea'

df_confirmed.loc[df_confirmed['country'] == 'Congo (Kinshasa)', "country"] = 'Democratic Republic of the Congo'
df_deaths.loc[df_deaths['country'] == 'Congo (Kinshasa)', "country"] = 'Democratic Republic of the Congo'
df_recovered.loc[df_recovered['country'] == 'Congo (Kinshasa)', "country"] = 'Democratic Republic of the Congo'

df_confirmed.loc[df_confirmed['country'] == "Cote d'Ivoire", "country"] = "Côte d'Ivoire"
df_deaths.loc[df_deaths['country'] == "Cote d'Ivoire", "country"] = "Côte d'Ivoire"
df_recovered.loc[df_recovered['country'] == "Cote d'Ivoire", "country"] = "Côte d'Ivoire"

df_confirmed.loc[df_confirmed['country'] == "Reunion", "country"] = "Réunion"
df_deaths.loc[df_deaths['country'] == "Reunion", "country"] = "Réunion"
df_recovered.loc[df_recovered['country'] == "Reunion", "country"] = "Réunion"

```

Melt and merge the raw confirmed, deaths and recovered CSV data into one Data Frame.

Before merging, we need to use `melt()` to unpivot Data Frames from current wide format into long format. In other words, we are kind of transposing all date columns into values. Here are the main settings for that:

- Use `'Province/State'`, `'Country/Region'`, `'Lat'`, `'Long'` as identifier variables. We will later use them for merging.
- Unpivot date columns (As we saw previously `columns[4:]`) with variable column `'Date'` and value column `'Confirmed'`

```
dates=df_confirmed.columns[4:]
confirmed_df_melt=df_confirmed.melt(
id_vars=['state','country','Lat','Long'],
value_vars=dates,
var_name='Date',
value_name='Confirmed'
)


deaths_df_melt=df_deaths.melt(
id_vars=['state','country','Lat','Long'],
value_vars=dates,
var_name='Date',
value_name='Deaths'
)


recovered_df_melt=df_recovered.melt(
id_vars=['state','country','Lat','Long'],
value_vars=dates,
var_name='Date',
value_name='Recovered'
)

print(confirmed_df_melt.shape)
print(deaths_df_melt.shape)
print(recovered_df_melt.shape)

```

```
(52668, 6)
(52668, 6)
(50094, 6)
```

Above should return new long Data Frames. All of them are ordered by ***Date*** and ***Country/Region*** because raw data was already ordered by ***Country/Region*** and the date columns are already in ASC order.

Here is the example of df_confirmed:

```
In [5]: confirmed_df_melt
```

	state	country	Lat	Long	Date	Confirmed
0	NaN	Afghanistan	33.939110	67.709953	1/22/20	0
1	NaN	Albania	41.153300	20.168300	1/22/20	0
2	NaN	Algeria	28.033900	1.659600	1/22/20	0
3	NaN	Andorra	42.506300	1.521800	1/22/20	0
4	NaN	Angola	-11.202700	17.873900	1/22/20	0
...	...	...	...	...	...	...
52663	NaN	Sao Tome and Principe	0.186400	6.613100	8/6/20	878
52664	NaN	Yemen	15.552727	48.516388	8/6/20	1768
52665	NaN	Comoros	-11.645500	43.333300	8/6/20	396
52666	NaN	Tajikistan	38.861000	71.276100	8/6/20	7665
52667	NaN	Lesotho	-29.610000	28.233600	8/6/20	742

In addition, we have to remove recovered data for Canada due to mismatch issue (🤦‍♂️ Canada recovered data is counted by Country-wise rather than Province/State-wise).

```
## Removing data for Canada  mismatch as canada recovered data is counted by Country wise rather than Province/Stat
recovered_df_melt=recovered_df_melt[recovered_df_melt['country']!= 'Canada']
```

After that, we use merge() to merge the 3 Data Frames one after another

```
## merging  3 dataframes one after another
full_table = pd.merge(left=confirmed_df_melt, right=deaths_df_melt, how='left',
                      on=['state', 'country', 'Date', 'Lat', 'Long'])

full_table = pd.merge(left=full_table, right=recovered_df_melt, how='left',
                      on=['state', 'country', 'Date', 'Lat', 'Long'])
full_table.sample(4)
```

Now, we should get a full table with Confirmed, Deaths and Recovered columns

Performing Data Cleaning

There are 2 tasks we would like to do

Converting Date from string to datetime

```
#convert to proper date format
full_table['Date']=pd.to_datetime (full_table['Date'])
# checking for missing value
full_table.isnull().sum()
```

```
state        36630
country          0
Lat              0
Long             0
Date             0
Confirmed        0
Deaths           0
Recovered     3762
dtype: int64
```

Replacing missing value NaN and changing datatypes

```
#fill na with 0
full_table['Recovered']=full_table['Recovered'].fillna(0)
##Handling the missing values
full_table[['state']]=full_table[['state']].fillna('None')
```

```
1]:
#checking datatypes
full_table.dtypes
```

```
1]: state                 object
    country               object
    Lat                  float64
    Long                 float64
    Date          datetime64[ns]
    Confirmed              int64
    Deaths                 int64
    Recovered            float64
    dtype: object
```

```
2]:
#fixing dtypes
full_table['Recovered']=full_table['Recovered'].astype(int)
```

Data Aggregation

So far, all the ***Confirmed***, ***Deaths***, ***Recovered*** are existing data from raw CSV dataset. Let's add an active cases column ***Active***, which is calculated by `active = confirmed — deaths — recovered`, mortality rate, recovery rate and closed cases.

```
:
#Grouped by day,country
datewise = full_table.groupby(['Date', 'country'])['Confirmed', 'Deaths', 'Recovered'].sum().reset_index()
```

```
:
#Calculating the Mortality Rate, Recovery Rate,active and closed cases
datewise["Mortality Rate"]=(datewise["Deaths"]/datewise["Confirmed"])*100
datewise["Recovery Rate"]=(datewise["Recovered"]/datewise["Confirmed"])*100
datewise["Active Cases"]=datewise["Confirmed"]-datewise["Recovered"]-datewise["Deaths"]
datewise["Closed Cases"]=datewise["Recovered"]+datewise["Deaths"]
```

And here is what `datewise_agg` looks like now.

Date	Confirmed	Recovered	Deaths	Mortality Rate	Recovery Rate	Active Cases	Closed Cases
2020-01-22	555	28	17	3.102190	5.109489	510	45
2020-01-23	654	30	18	2.799378	4.665630	606	48
2020-01-24	941	36	26	2.826087	3.913043	879	62
2020-01-25	1434	39	42	2.987198	2.773826	1353	81
2020-01-26	2118	52	56	2.698795	52.361446	2010	108
...	...	...	...	...	...	...	...
2020-08-02	18079723	10584823	690065	583.348796	12271.460212	6804835	11274888
2020-08-03	18282208	10807151	694396	582.975753	12415.934763	6780661	11501547
2020-08-04	18540789	11028008	701347	582.449568	12509.246657	6811434	11729355
2020-08-05	18811953	11249203	708424	579.680783	12586.733134	6854326	11957627
2020-08-06	19097149	11437996	714940	577.423015	12599.382598	6944213	12152936

After that, add day wise ***New cases***, ***New deaths*** and ***New recovered*** by deducting the corresponding cumulative data on the previous day.

```
# new cases
temp = datewise.groupby(['country', 'Date', ])['Confirmed', 'Deaths', 'Recovered']
temp = temp.sum().diff().reset_index()

mask = temp['country'] != temp['country'].shift(1)

temp.loc[mask, 'Confirmed'] = np.nan
temp.loc[mask, 'Deaths'] = np.nan
temp.loc[mask, 'Recovered'] = np.nan

# renaming columns
temp.columns = ['country', 'Date', 'New cases', 'New deaths', 'New recovered']

# merging new values

df_covid19 = pd.merge(datewise, temp, on=['country', 'Date'])

# filling na with 0

df_covid19 = df_covid19.fillna(0)

# fixing data types

cols = ['New cases', 'New deaths', 'New recovered']
df_covid19[cols] = df_covid19[cols].astype('int')
#
df_covid19['New cases'] = df_covid19['New cases'].apply(lambda x: 0 if x<0 else x)
```

And here is what df_covid19 looks like now.

	Date	country	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases	Closed Cases	New cases	New deaths	New recovered
28850	2020-06-23	Japan	17879	965	16096	5.397394	90.027406	818	17061	59	10	139
7667	2020-03-02	Senegal	1	0	0	0.000000	0.000000	1	0	1	0	0
5322	2020-02-19	Ethiopia	0	0	0	0.000000	0.000000	0	0	0	0	0
16647	2020-04-19	Madagascar	121	0	39	0.000000	32.231405	82	39	1	0	4

Web scraping of data from world meter to add features like population density and other features and merged with df_covid19 dataset.

```
#loading the file of world poulation
world_population=pd.read_csv('global_pop_data.csv')

#subsetting
world_population = world_population[['Country (or dependency)', 'Population (2020)', 'Density (P/Km²)', 'Land Area
world_population.columns = ['Country (or dependency)', 'Population (2020)', 'Density', 'Land Area', 'Med Age', 'Urb

#Replace united states by US
world_population.loc[world_population['Country (or dependency)']=='United States', 'Country (or dependency)'] = 'US

# Remove the % character from Urban Pop values
world_population['Urban Pop'] = world_population['Urban Pop'].str.rstrip('%')

## Replace Urban Pop and Med Age "N.A" by their respective modes, then transform to int
world_population.loc[world_population['Urban Pop']=='N.A.', 'Urban Pop'] = int(world_population.loc[world_populatic
world_population['Urban Pop'] = world_population['Urban Pop'].astype('int16')
world_population.loc[world_population['Med Age']=='N.A.', 'Med Age'] = int(world_population.loc[world_population['M
world_population['Med Age'] = world_population['Med Age'].astype('int16')

#now join dataset to previous data set
final_data=pd.merge(
    left=df_covid19,
    right=world_population,
    left_on='country',
    right_on='Country (or dependency)',
    how='left'
)

#dropping country(or dependency data)
final_dataset=final_data.drop('Country (or dependency)',axis=1)

```

Here, final_dataset looks like:

Date	country	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases	Closed Cases	New cases	New deaths	New recovered	Population (2020)	Density	Land Area	Med Age	Urban Pop
2020-07-15	Germany	200890	9080	186000	4.519887	92.587983	5810	195080	434	2	900	83783942.0	240.0	348560.0	46.0	76.0
2020-03-08	Montenegro	0	0	0	0.000000	0.000000	0	0	0	0	0	628066.0	47.0	13450.0	39.0	68.0
2020-06-23	Barbados	97	7	85	7.216495	87.628866	5	92	0	0	0	287375.0	668.0	430.0	40.0	31.0
2020-06-08	Jamaica	599	10	405	1.669449	67.612688	184	415	1	0	0	2961167.0	273.0	10830.0	31.0	55.0
2020-07-12	Thailand	3217	58	3088	1.802922	95.990053	71	3146	1	0	0	69799978.0	137.0	510890.0	40.0	51.0

Adding continents to datasets and final datasets looks like as shown:

```
#getting all countries
countries = np.asarray(df_confirmed["country"])
countries1 = np.asarray(df_covid19["country"])

#Continent_code to Continent_names
continents = {
     'NA': 'North America',
    'SA': 'South America',
    'AS': 'Asia',
    'OC': 'Australia',
    'AF': 'Africa',
    'EU' : 'Europe',
    'na' : 'Others'
}

# Defininng Function for getting continent code for country.
def country_to_continent_code(country):
    try:
        return pc.country_alpha2_to_continent_code(pc.country_name_to_country_alpha2(country))
    except :
        return 'na'

# #Collecting Continent Information
df_confirmed.insert(2,"continent", [continents[country_to_continent_code(country)] for country in countries[:]])
df_deaths.insert(2,"continent",  [continents[country_to_continent_code(country)] for country in countries[:]])
#df_recovered.insert(2,"continent",  [continents[country_to_continent_code(country)] for country in countries[:]])
df_covid19.insert(1,"continent",  [continents[country_to_continent_code(country)] for country in countries1[:]])

```

```
# checking data with continent
df_covid19.sample(10)
```

	Date	continent	country	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases	Closed Cases	New cases	New deaths	New recovered
21252	2020-05-14	Australia	Australia	7019	98	6334	1.396210	90.240775	587	6432	30	0	37
35013	2020-07-26	Africa	Democratic Republic of the Congo	8831	204	5510	2.310044	62.393840	3117	5714	30	0	205
11232	2020-03-21	Africa	Rwanda	17	0	0	0.000000	0.000000	17	0	0	0	0

Exploratory Data Analysis

Exploratory data analysis, also known as *Data Exploration,* is a tool to understand a dataset better through summarizing, visualizing and becoming familiar with the important characteristics of a data set. EDA is not only a critical step in building any Machine Learning model but a useful way to generate insights from the data.

Follow me GitHub for code: https://github.com/reetibhagat/capstone-1-covid-19/blob/master/notebooks/Exploratory_data_analyis_capstone_1.ipynb

I will start with general analysis of data.

General Analysis of Data

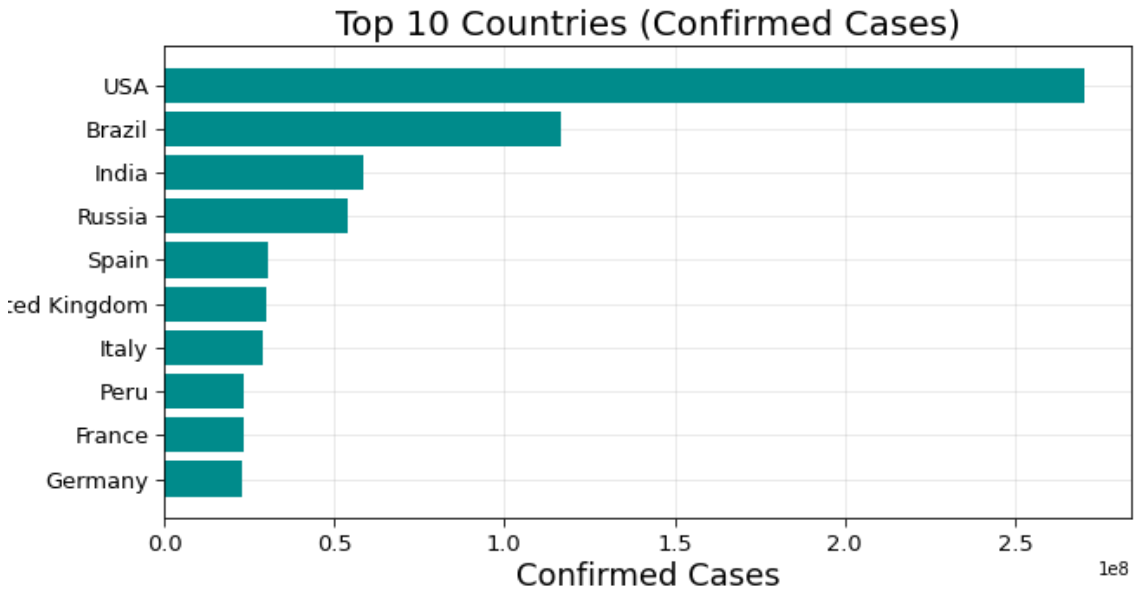
Getting country wise and continent wise data.

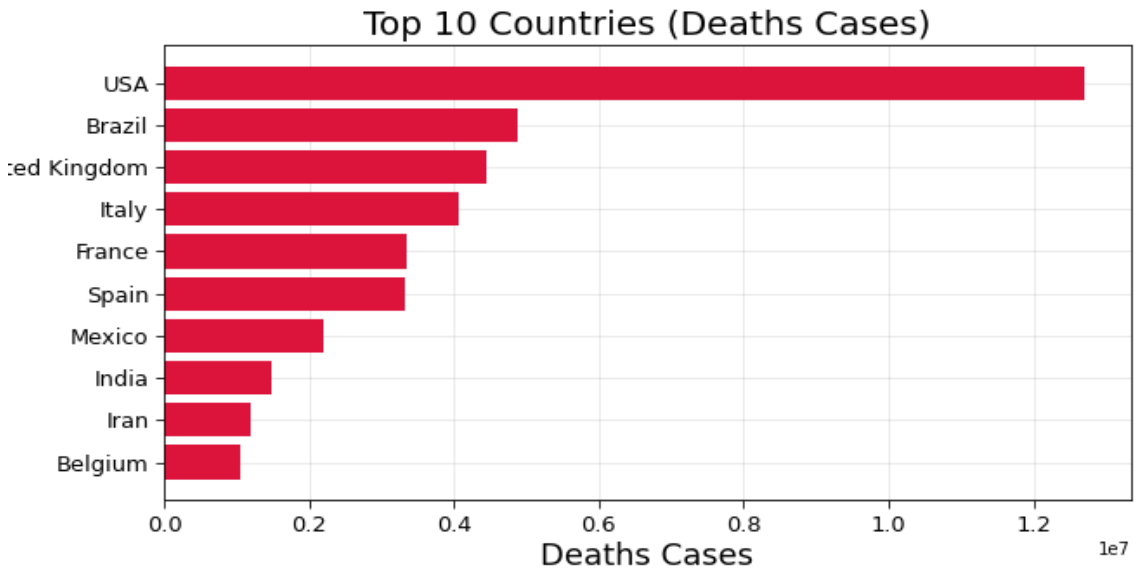
country	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases
USA	270667192	12712324	71099993	769.16	3462.41	186854875
Brazil	116598088	4874345	75028306	694.77	6203.11	36695437
India	58637857	1489724	35492203	416.16	7347.25	21655930
Russia	53872139	759927	31566182	159.63	7621.11	21546030
Spain	30340143	3317656	16597343	1539.13	8277.12	10425144
United Kingdom	29808290	4460192	140638	1943.43	559.59	25207460
Italy	29223661	4059247	17675810	2044.36	7927.04	7488604
Peru	23480144	846483	13785801	435.34	6220.32	8847860
France	23464802	3351191	8000341	2037.64	5066.61	12113270
Germany	23171655	962864	19036532	536.02	11389.19	3172259

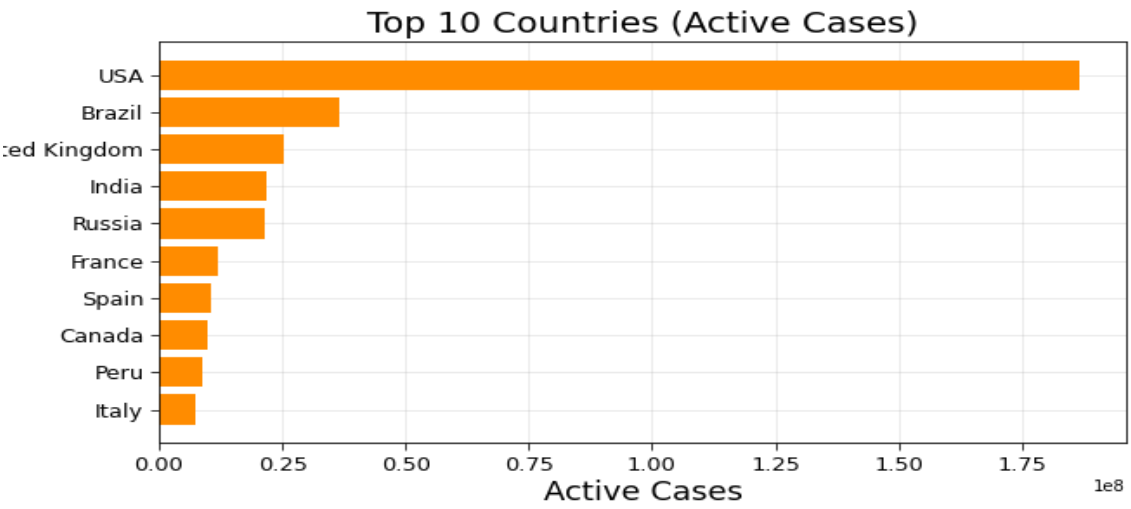
Continent-wise Data

continent	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases
Africa	37709406	904389	19805801	23529.65	283044.01	16999216
Asia	219219516	5601618	144850378	15027.05	336605.64	68767520
Australia	1337210	16006	982688	419.58	38472.80	338516
Europe	249950948	20723423	120922012	30722.22	313984.74	108305513
North America	312311311	16014189	90841316	13642.47	148984.47	205455806
Others	889051	13722	444799	4978.22	59165.06	430530
South America	186568641	7142013	116023973	6994.25	66492.83	63402655

Top 10 countries Confirmed, Deaths, Active and recovered cases

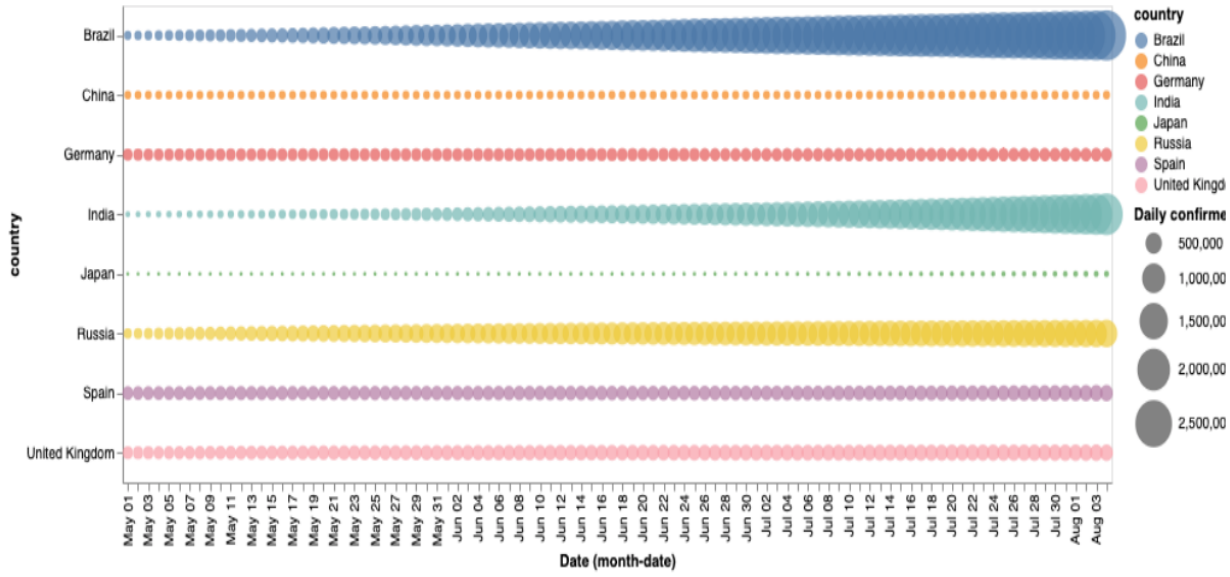






Showing the coronavirus spreads

I have used altair tool for interactive visualization to compare cases of different countries.



Correlation analysis and visualization of Map

Correlation Analysis
Plotting Heat map of correlation of confirmed cases, recovered cases, deaths and active cases.

```
]:  1 # Country wise correlation
    2
    3 df_countries_cases.iloc[:,:].corr().style.background_gradient(cmap='Reds').forma
```

]:

	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases
Confirmed	1.000	0.927	0.897	0.117	-0.065	0.963
Deaths	0.927	1.000	0.790	0.286	-0.096	0.907
Recovered	0.897	0.790	1.000	0.097	0.030	0.745
Mortality Rate	0.117	0.286	0.097	1.000	-0.265	0.104
Recovery Rate	-0.065	-0.096	0.030	-0.265	1.000	-0.115
Active Cases	0.963	0.907	0.745	0.104	-0.115	1.000

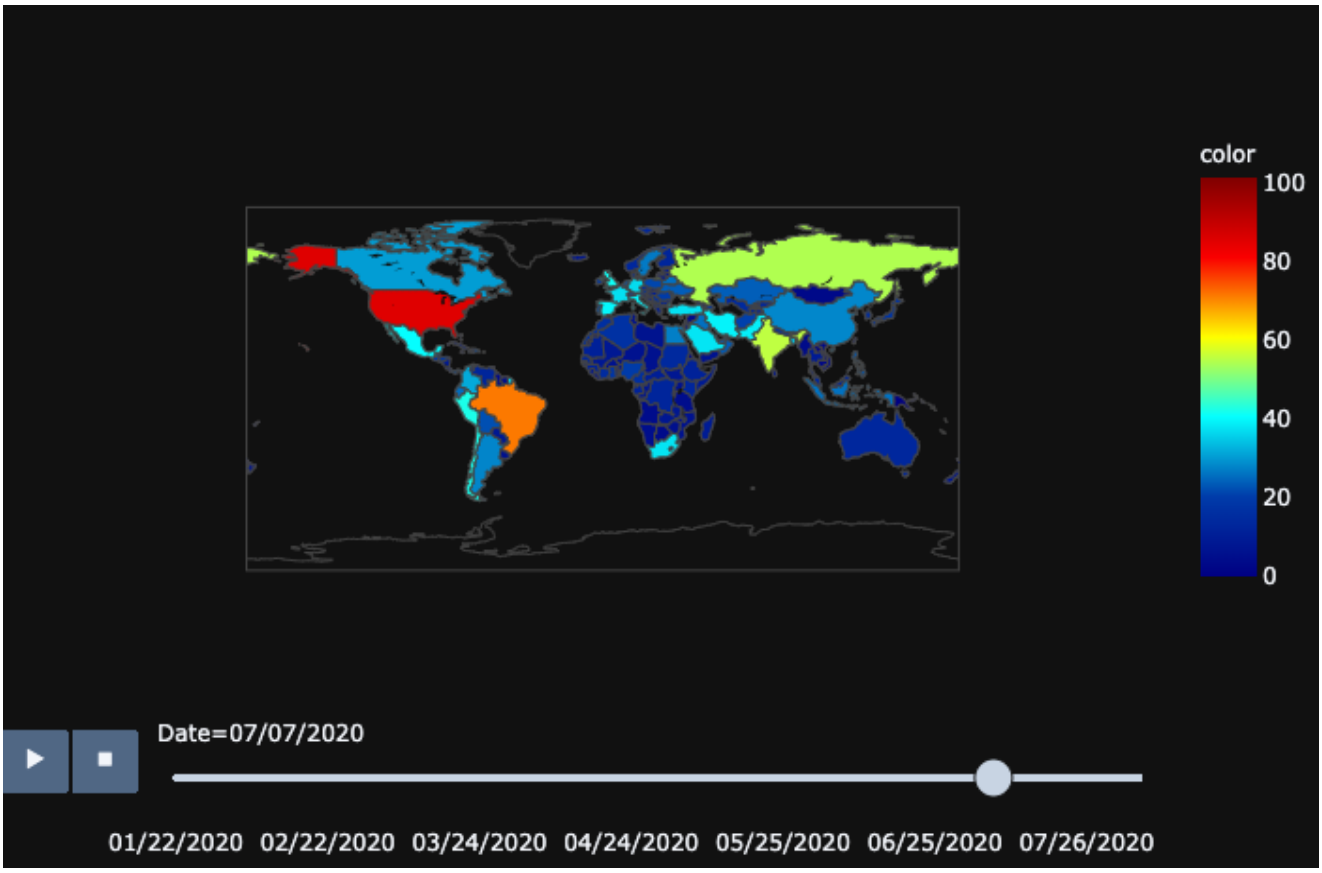
```
:  1 # Continent wise correlation
   2 df_continents_cases.iloc[:,:].corr().style.background_gradient(cmap='Reds').format("{:.3f}")
```

:

	Confirmed	Deaths	Recovered	Mortality Rate	Recovery Rate	Active Cases
Confirmed	1.000	0.869	0.874	0.450	0.423	0.920
Deaths	0.869	1.000	0.677	0.614	0.394	0.842
Recovered	0.874	0.677	1.000	0.426	0.533	0.615
Mortality Rate	0.450	0.614	0.426	1.000	0.863	0.361
Recovery Rate	0.423	0.394	0.533	0.863	1.000	0.249
Active Cases	0.920	0.842	0.615	0.361	0.249	1.000

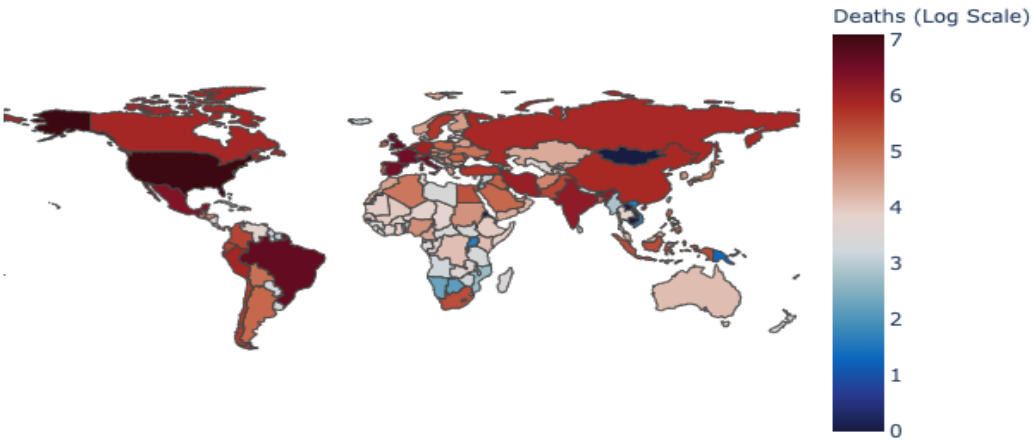
Visualization of Map

```
# countrywise visual global spread in the world map using choropleth

fi_1 = px.choropleth(df_data, locations="country", locationmode='country names',
                     color=np.power(df_data["Confirmed"],0.3)-2 , hover_name="country",
                     hover_data=["Confirmed"],
                     range_color= [0, max(np.power(df_data["Confirmed"],0.3))],
                      animation_frame="Date",
                     color_continuous_scale=px.colors.sequential.Peach
                    )
fi_1.update_geos(fitbounds="locations", visible=True)
fi_1.update_coloraxes(colorscale="jet")
fi_1.update(layout_coloraxis_showscale=True)
fi_1.update_layout( margin={"r":0,"l":0,"b":0}, height=700,template='plotly_dark')
```

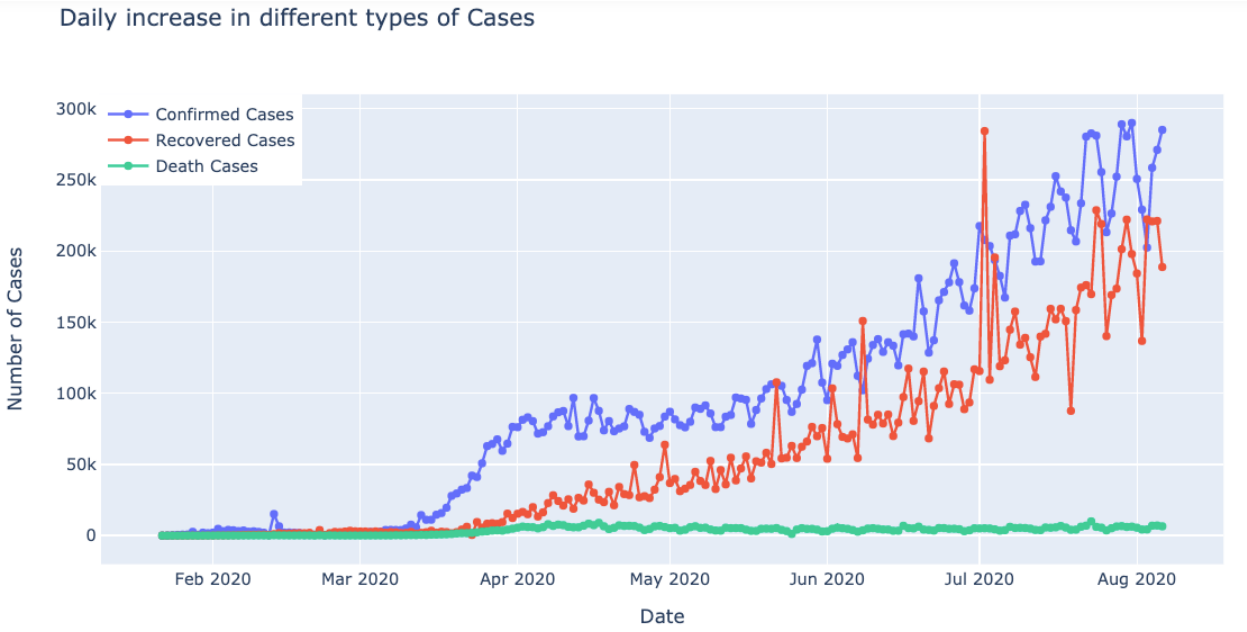


Confirmed Cases Heat Map (Log Scale)

Deaths Heat Map (Log Scale)

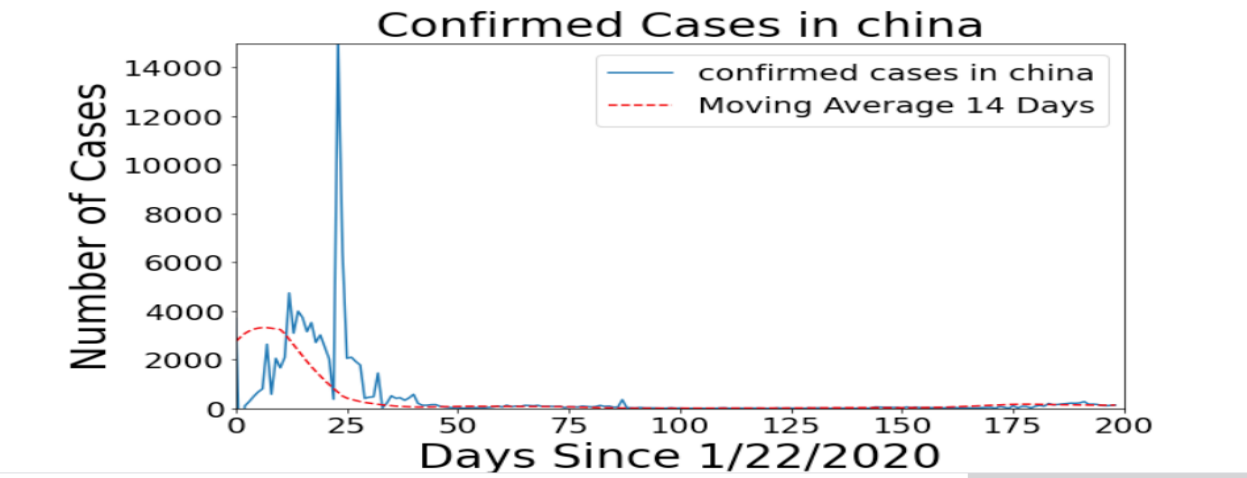


Daily Increase in Different types of Cases

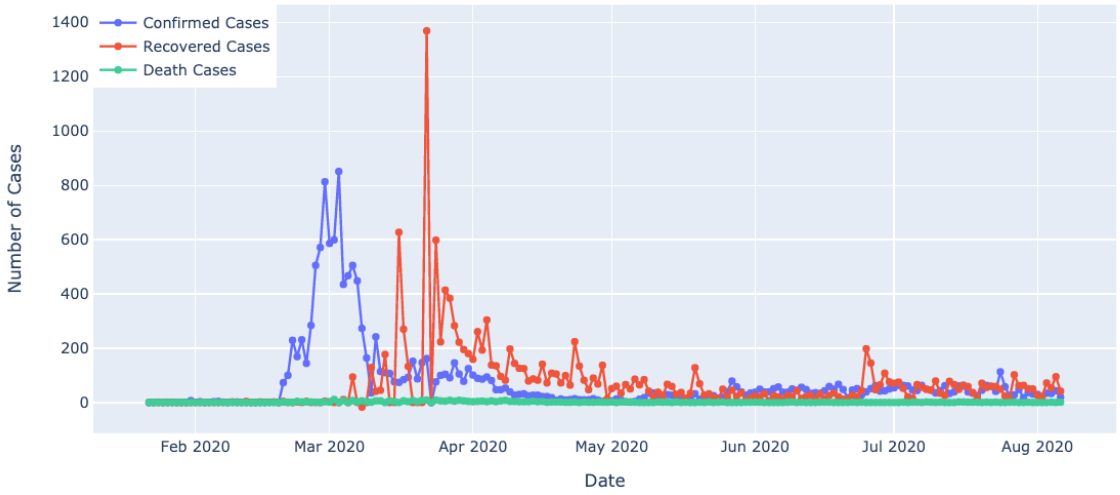


Comparison of daily increase in cases in South Korea, Australia, New Zealand and Germany, China, USA and rest of world

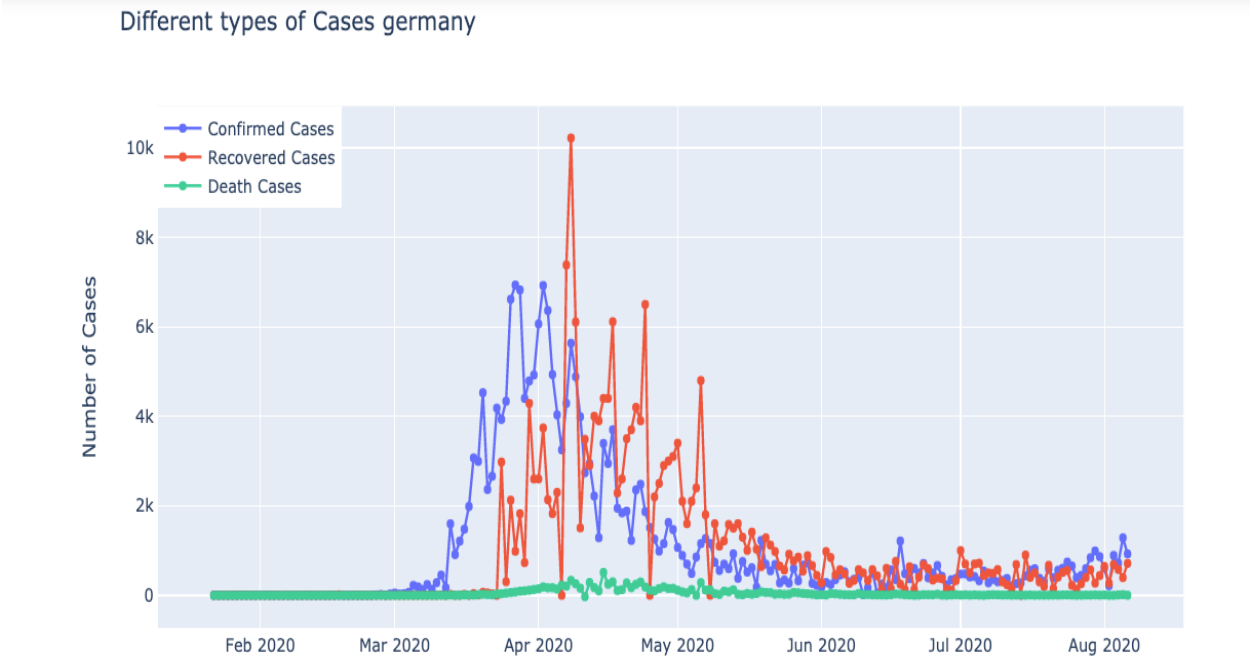
China



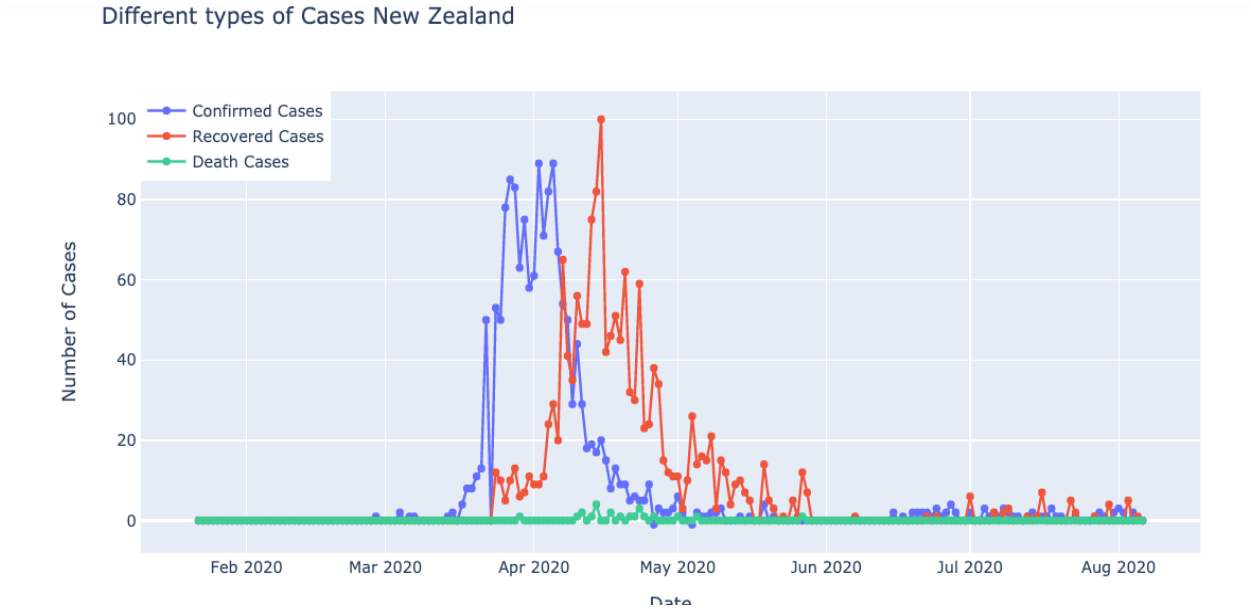
South Korea



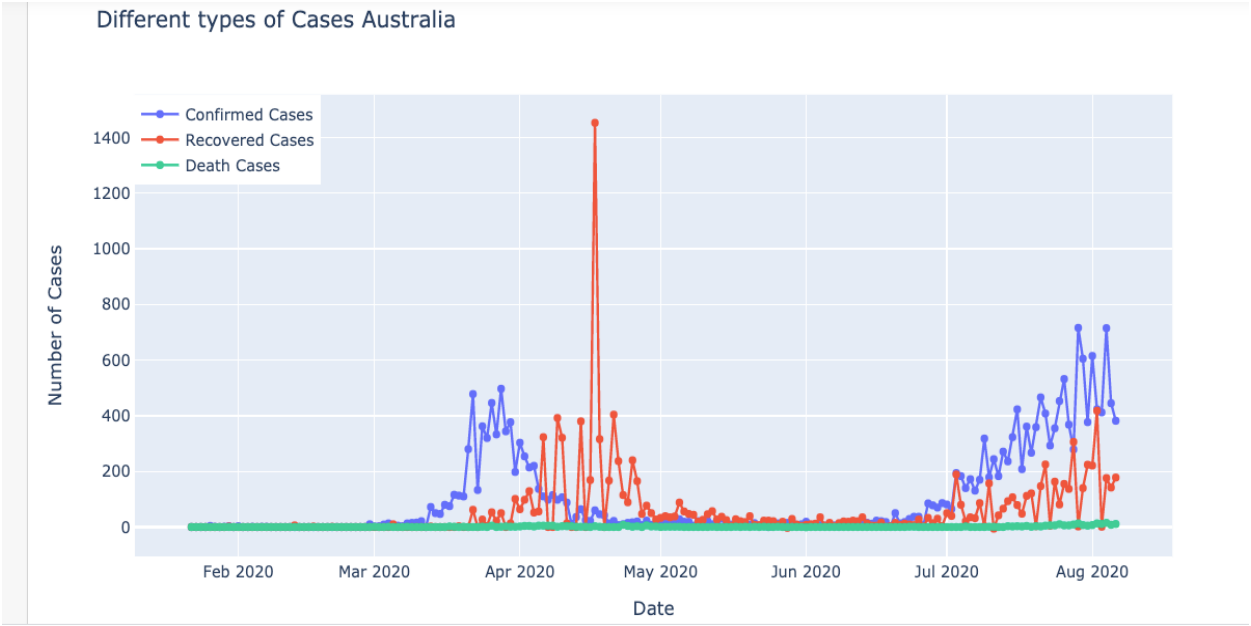
Germany



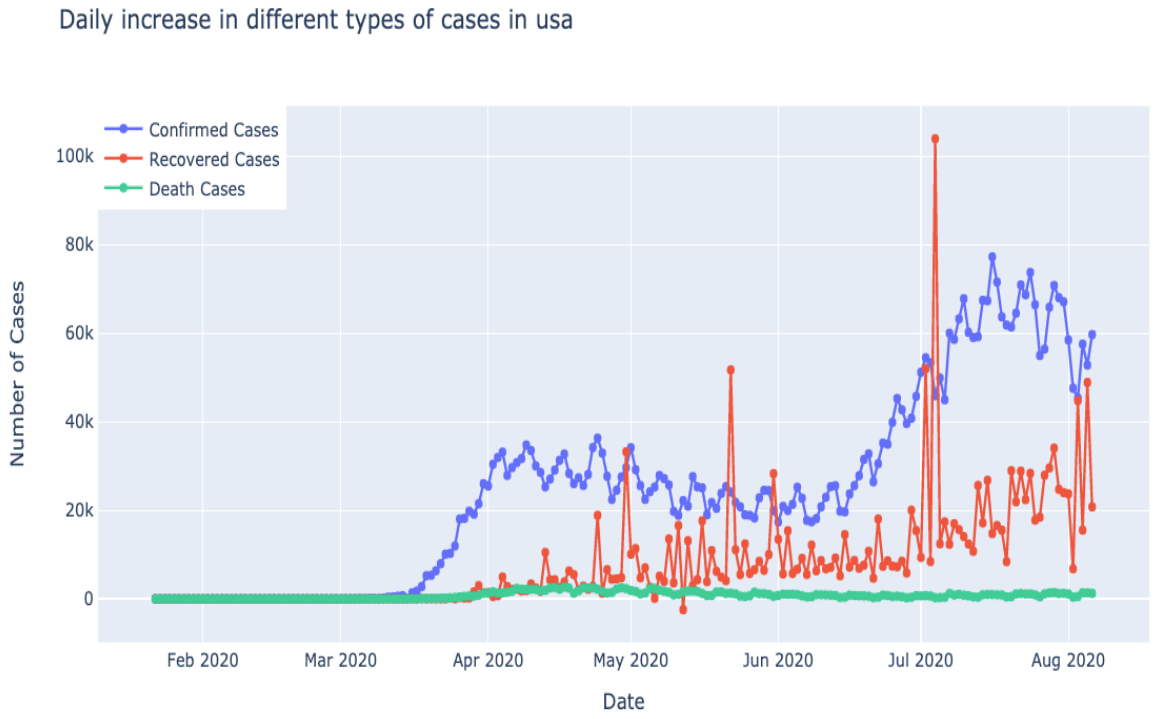
New Zealand



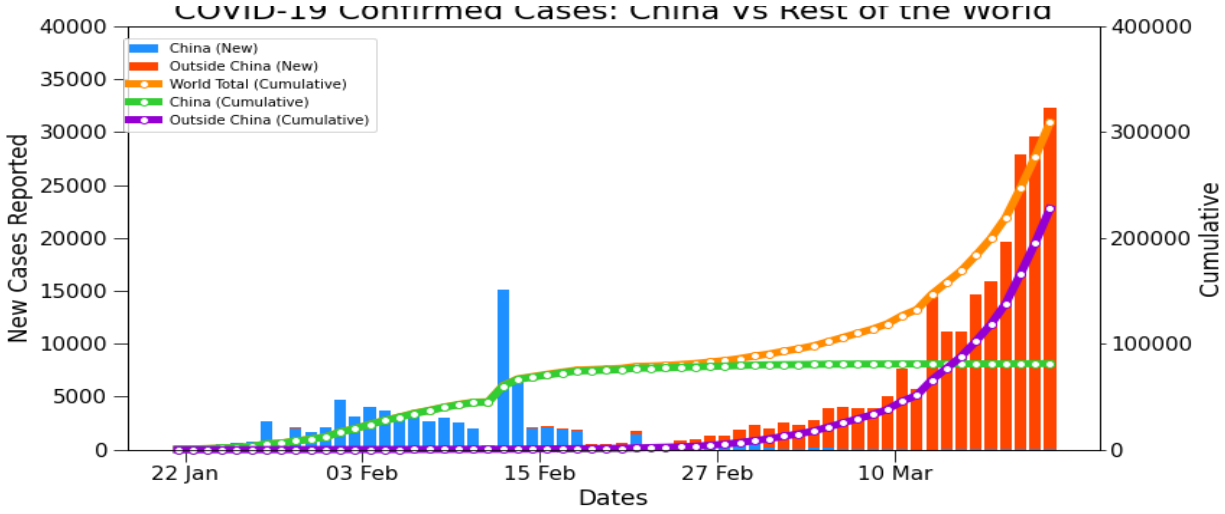
Australia



USA



China vs rest of world



Clustering of Countries

The clustering of countries can be done considering different features. Here I'm trying to cluster different countries based on the Mortality and Recovery rate of individual country.

As we all are well aware that COVID-19 has different Mortality Rate among different countries based on different factors and so is the Recovery Rate because of pandemic controlling practices followed by the individual country. Also, Mortality Rate and Recovery Rate both together takes into account all types of cases Confirmed, Recovered and Deaths.

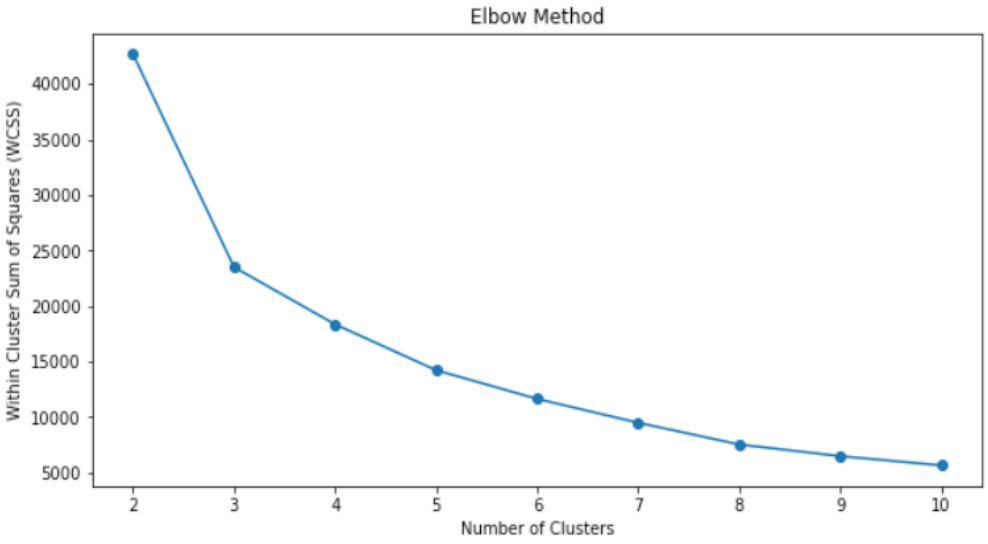
Let's check out how these clusters look like!

```
:
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
X=df_covid19[["Mortality Rate","Recovery Rate"]]
#Standard Scaling since K-Means Clustering is a distance based alogrithm
std= StandardScaler()
X=std.fit_transform(X)
```

```
:
wcss=[]
sil=[]
for i in range(2,11):
    clf=KMeans(n_clusters=i,init='k-means++',random_state=42)
    clf.fit(X)
    labels=clf.labels_
    centroids=clf.cluster_centers_
    sil.append(silhouette_score(X, labels, metric='euclidean'))
    wcss.append(clf.inertia_)
```

```
]:  1 x=np.arange(2,11)
    2 plt.figure(figsize=(10,5))
    3 plt.plot(x,wcss,marker='o')
    4 plt.xlabel("Number of Clusters")
    5 plt.ylabel("Within Cluster Sum of Squares (WCSS)")
    6 plt.title("Elbow Method")
```

```
]: Text(0.5, 1.0, 'Elbow Method')
```



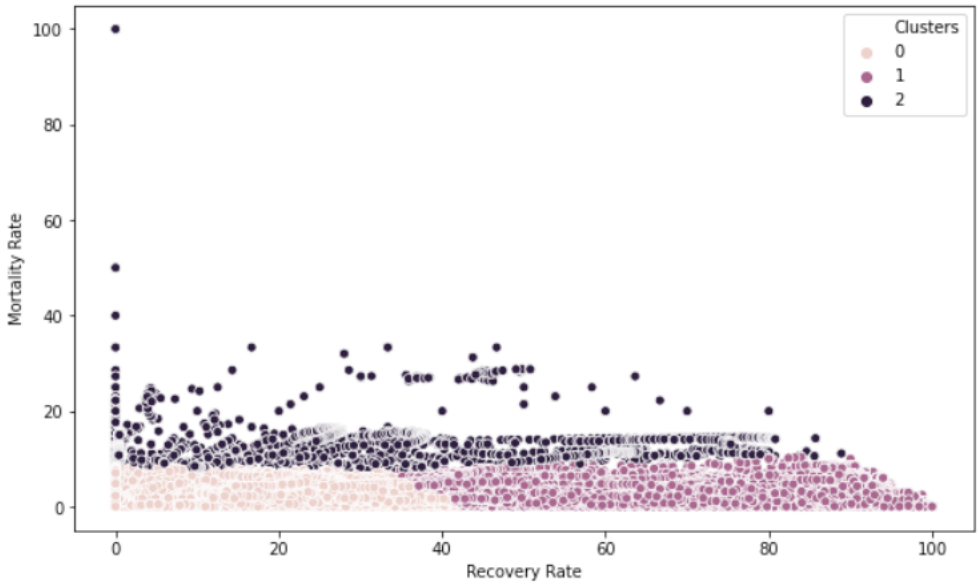
Summary of clusters

Using predict()

```
:  1 clf_final=KMeans(n_clusters=3,init='k-means++',random_state=6)
   2 clf_final.fit(X)
   3 df_covid19["Clusters"]=clf_final.predict(X)
```

```
:  1 ## SUMMARY OF CLUSTERS
   2 print("Few Countries belonging to Cluster 0: ",list(df_covid19[df_covid19["Clusters"]==0].head(10)['country']))
   3 print("Few Countries belonging to Cluster 1: ",list(df_covid19[df_covid19["Clusters"]==1].head(10)['country']))
   4 print("Few Countries belonging to Cluster 2: ",list(df_covid19[df_covid19["Clusters"]==2].head(10)['country']))
   5
```

```
Few Countries belonging to Cluster 0:  ['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola', 'Antigua and Barbud
a', 'Argentina', 'Armenia', 'Australia', 'Austria']
Few Countries belonging to Cluster 1:  ['Sri Lanka', 'Sri Lanka', 'Sri Lanka', 'Sri Lanka', 'Cambodia', 'Finland', 'N
epal', 'Russia', 'Sri Lanka', 'Australia']
Few Countries belonging to Cluster 2:  ['Philippines', 'Philippines', 'Philippines', 'Philippines', 'Philippines', 'P
hilippines', 'Philippines', 'Philippines', 'Philippines', 'Philippines']
```

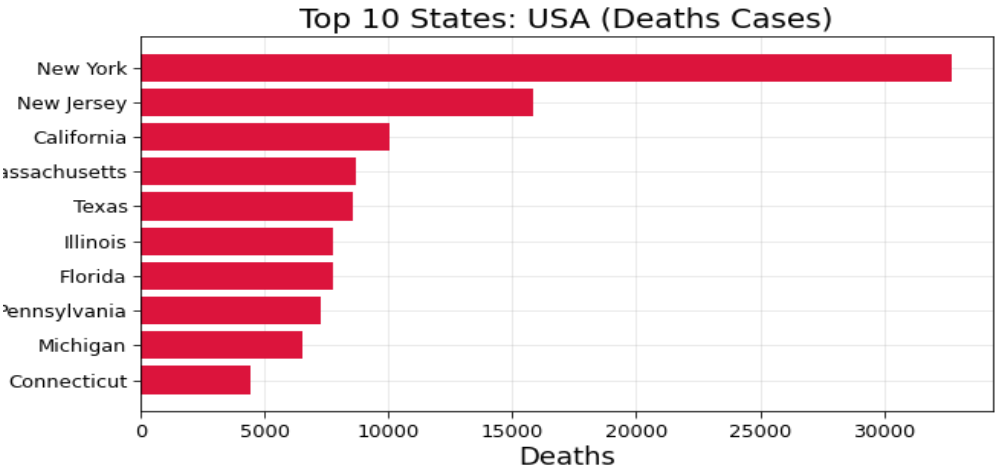


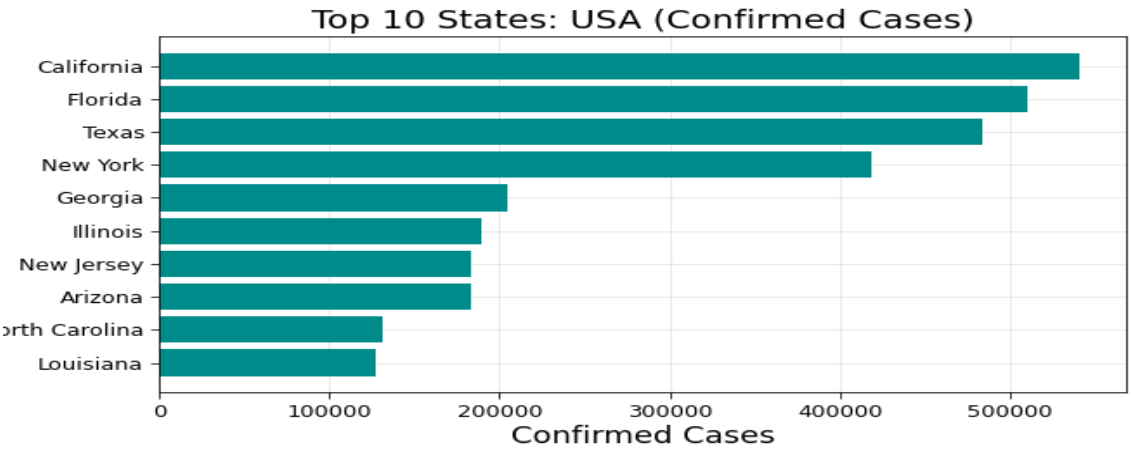

```
[ ]: 1 print("Avergae Mortality Rate of Cluster 0: ",df_covid19[df_covid19["Clusters"]==0]["Mortality Rate"].mean())
     2 print("Avergae Recovery Rate of Cluster 0: ",df_covid19[df_covid19["Clusters"]==0]["Recovery Rate"].mean())
     3 print("Avergae Mortality Rate of Cluster 1: ",df_covid19[df_covid19["Clusters"]==1]["Mortality Rate"].mean())
     4 print("Avergae Recovery Rate of Cluster 1: ",df_covid19[df_covid19["Clusters"]==1]["Recovery Rate"].mean())
     5 print("Avergae Mortality Rate of Cluster 2: ",df_covid19[df_covid19["Clusters"]==2]["Mortality Rate"].mean())
     6 print("Avergae Recovery Rate of Cluster 2: ",df_covid19[df_covid19["Clusters"]==2]["Recovery Rate"].mean())
```

```
Avergae Mortality Rate of Cluster 0:  1.1072205889325661
Avergae Recovery Rate of Cluster 0:  7.728973307509244
Avergae Mortality Rate of Cluster 1:  2.5687085347822194
Avergae Recovery Rate of Cluster 1:  72.50337751788645
Avergae Mortality Rate of Cluster 2:  14.562654682780716
Avergae Recovery Rate of Cluster 2:  28.86145303975767
```

Cluster 2 is a set of countries which have really High Mortality Rate and considerably Good Recovery Rate. Basically, few countries among these clusters have seen already the worst of this pandemic but are now recovering with healthy Recovery Rate.
Cluster 0 is set of countries which have Low Mortality Rate and really High Recovery Rate. These are the set of countries who has been able to control the COVID-19 by following pandemic controlling practices rigorously.
Cluster 1 is set of countries which have Low Mortality Rate and Low Recovery Rate. These countries need to pace up their Recovery Rate to get out it, some these, countries have really high number of Infected Cases, but Low Mortality is positive sign out of it.

Top 10 states in USA: confirmed and deaths cases.





Covid-19 Forecasting using Time Series Analysis

It is a series of observations taken at specified times basically at equal intervals. It is used to predict future values based on past observed values.

The components you might observe in the time-series analysis are Trend, Seasonal, Irregular, and Cyclicity.

In the case of such datasets where only one variable is observed at each time is called ‘Univariate Time Series’ and if two or more variables are observed at each time is called ‘Multivariate Time Series’.
In this project, we will focus on the univariate time series for forecasting with different model and choose best model comparing its accuracy.

Comparison of different models (Models Description)

COVID-19 coronavirus has been global threat of the disease and infected humans rapidly. Control of the pandemic is urgently essential, and science community have continued to research treatment agents. Support therapy and intensive care units in hospitals are also effective to overcome of COVID-19. Statistic forecasting models could aid to healthcare system in prevention of COVID-19. This study aimed to compose of forecasting model that could be practical to predict the spread of COVID-19 world. For this purpose, I will be using ARIMA, SARIMA, Exponential Smoothing and FACEBOOK PROPHET model.

Forecasting with different Time series models and comparing the accuracies to find the best model for covid19 dataset.

“Prediction is very difficult, especially about the future”. Forecasting is the process of making predictions of the future, based on past and present data. One of the most common methods for this is the ARIMA model, which stands for Autoregressive Integrated Moving Average. I will be using exponential smoothing, AUTO-ARIMA, SARIMA AND PROPHET.

Holt's Winter Time Series analysis

Exponential smoothing is a time series forecasting method for univariate data that can be extended to support data with a systematic trend or seasonal component.
It is a powerful forecasting method that may be used as an alternative to the popular Box-Jenkins ARIMA family of methods.
The implementations of Exponential Smoothing in Python are provided in the stats models Python library.

Double and Triple Exponential Smoothing
Single, Double and Triple Exponential Smoothing can be implemented in Python using the Exponential Smoothing Stats models class.
First, an instance of the Exponential Smoothing class must be instantiated, specifying both the training data and some configuration for the model.
Specifically, you must specify the following configuration parameters:

trend: The type of trend component, as either “add” for additive or “mul” for multiplicative. Modeling the trend can be disabled by setting it to None. damped: Whether or not the trend component should be damped, either True or False. seasonal: The type of seasonal component, as either “add” for additive or “mul” for multiplicative. Modeling the seasonal component can be disabled by setting it to None. seasonal periods: The number of time steps in a seasonal period, e.g. 12 for 12 months in a yearly seasonal structure (more here). The model can then be fit on the training data by calling the fit() function.
This function allows you to either specify the smoothing coefficients of the exponential smoothing model or have them optimized. By default, they are optimized (e.g. optimized=True). These coefficients include:
smoothing level (alpha): the smoothing coefficient for the level. smoothing slope (beta): the smoothing coefficient for the trend. smoothing seasonal (gamma): the smoothing coefficient for the seasonal component. Damping slope (phi): the coefficient for the damped trend.
Additionally, the fit function can perform basic data preparation prior to modeling; specifically: use box cox: Whether or not to perform a power transform of the series (True/False) or specify the lambda for the transform. The fit() function will return an instance of the Holt Winters Results class that contains the learned coefficients. The forecast() or the predict() function on the result object can be called to make a forecast.
In [7]:

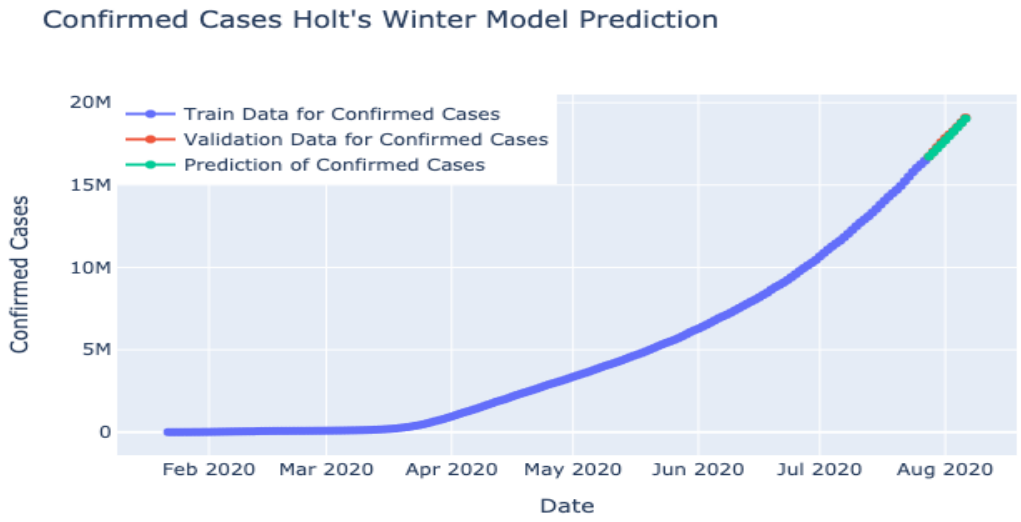
Steps:
1. train test split
2.using exponential model
3.predicting using. forecast and checking model scores

```
# train and test split
model_train=df_data.iloc[:int(df_data.shape[0]*0.95)]
valid=df_data.iloc[int(df_data.shape[0]*0.95):]
y_pred=valid.copy()
```

```

# using exponentialSmoothing model
es=ExponentialSmoothing(np.asarray(model_train['Confirmed']),seasonal_periods=11,trend='mul', seasonal='add').fit()
```

```
#predicting using .forecast and checking model scores

model_scores=[]
model_holts=pd.DataFrame(es.forecast(steps=10),index=valid.index)
y_pred["Holt's Winter Model"]=model_holts
model_scores.append(np.sqrt(mean_squared_error(y_pred["Confirmed"],y_pred["Holt's Winter Model"])))
print("Root Mean Square Error for Holt's Winter Model: ",np.sqrt(mean_squared_error(y_pred["Confirmed"],y_pred["Hol
```

```
Root Mean Square Error for Holt's Winter Model:  81065.9905908756
```



AUTO ARIMA MODEL

Usually, in the basic ARIMA model, we need to provide the p,d, and q values which are essential. We use statistical techniques to generate these values by performing the difference to eliminate the non-stationarity and plotting ACF and PACF graphs. In Auto ARIMA, the model itself will generate the optimal p, d, and q values which would be suitable for the data set to provide better forecasting.
from pmdarima.arima import auto_arima is imported.
Auto-Regressive (p) -> Number of autoregressive terms.
Integrated (d) -> Number of nonseasonal differences needed for stationarity.
Moving Average (q) -> Number of lagged forecast errors in the prediction equation.
In the Auto ARIMA model, note that small p, d, q values represent non-seasonal components, and capital P, D, Q represent seasonal components. It works similarly like hyper tuning techniques to find the optimal value of p, d, and q with different combinations and the final values would be determined with the lower AIC, BIC parameters taking into consideration. Here, we are trying with the p, d, q values ranging from 0 to 5 to get better optimal values from the model.

Steps:
1. train test split
2.using exponential model
3.predicting using. forecast and checking model scores

[13]:
```
# using auto-arima model
from pmdarima.arima import auto_arima
model_arima= auto_arima(model_train["Confirmed"],trace=True, error_action='ignore', start_p=1,start_q=1,max_p=3,max
                  suppress_warnings=True,stepwise=False,seasonal=False)
results=model_arima.fit(model_train["Confirmed"])
```

.4]:

SARIMAX Results

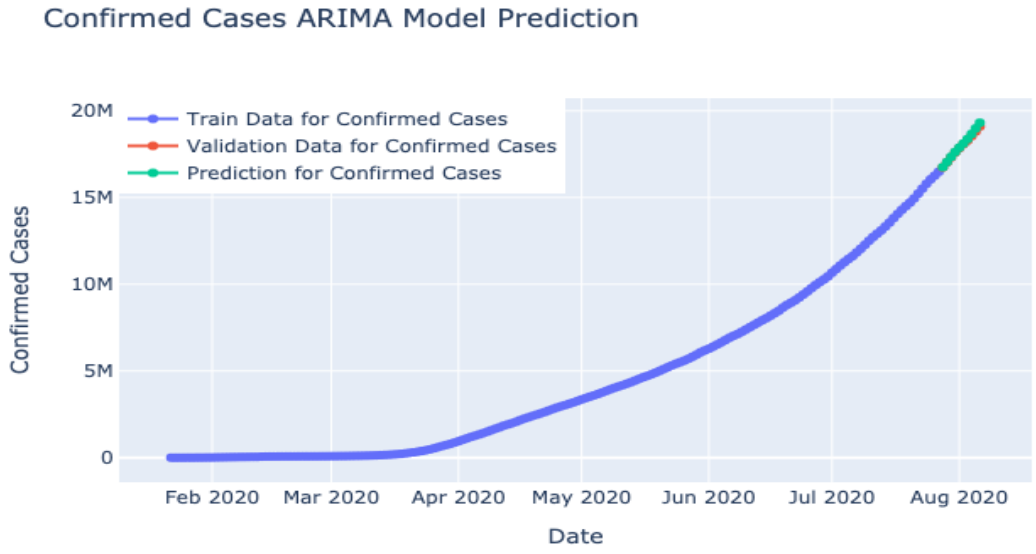
Dep. Variable:	y	**No. Observations:**	188
Model:	SARIMAX(2, 2, 3)	**Log Likelihood**	-1969.907
Date:	Thu, 27 Aug 2020	**AIC**	3953.813
Time:	14:39:17	**BIC**	3976.394
Sample:	0	**HQIC**	3962.964
	- 188		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	2780.7044	372.591	7.463	0.000	2050.439	3510.969
ar.L1	1.2043	0.016	74.627	0.000	1.173	1.236
ar.L2	-1.0000	0.010	-98.812	0.000	-1.020	-0.980
ma.L1	-1.4642	0.069	-21.083	0.000	-1.600	-1.328
ma.L2	1.1489	0.099	11.588	0.000	0.955	1.343
ma.L3	-0.1950	0.075	-2.609	0.009	-0.341	-0.049
sigma2	9.505e+07	0.001	8.72e+10	0.000	9.5e+07	9.5e+07

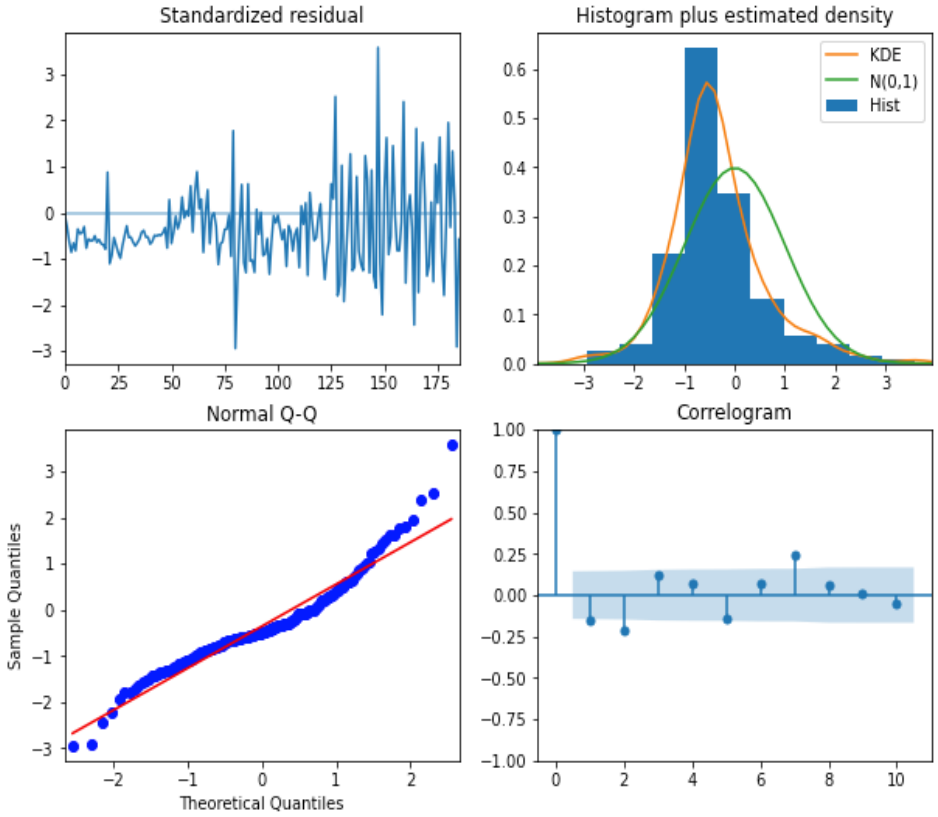
Ljung-Box (Q):	132.70	**Jarque-Bera (JB):**	68.88
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	5.44	**Skew:**	0.82
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.48

16]:
```
# checking accuracy of model by metrics rmse
model_scores.append(np.sqrt(mean_squared_error(valid["Confirmed"],arima_prediction)))
y_pred["ARIMA Model Prediction"]=arima_prediction
print("Root Mean Square Error for ARIMA Model: ",np.sqrt(mean_squared_error(valid["Confirmed"],arima_prediction)))
```

```
Root Mean Square Error for ARIMA Model:  102709.16411405608
```



]:
```
1 results.plot_diagnostics(figsize=(10,8))
2 plt.show()
```



SARIMA

In order to do this we will need to choose p,d,q values for the ARIMA, and P,D,Q values for the Seasonal component.
There are many ways to choose these values statistically, such as looking at auto-correlation plots, correlation plots, domain experience, etc.
One simple approach is to perform a grid search over multiple values of p,d,q,P,D,and Q using some sort of performance criteria. The Akaike information criterion (AIC) is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models.
The AIC value will allow us to compare how well a model fits the data and takes into account the complexity of a model, so models that have a better fit while using fewer features will receive a better (lower) AIC score than similar models that utilize more features.
The pyramid-arima library for Python allows us to quickly perform this grid search and even creates a model object that you can fit to the training data. It is similar alike arima but seasonality is true here.
Steps:
1.Test for stationary
2. train test split
3.using exponential model
4.predicting using. forecast and checking model scores

```
: 1 #Test for stationary
  2 from pmdarima.arima import ADFTest
  3 adf_test=ADFTest(alpha=0.05)
  4 adf_test.should_diff(df_data["Confirmed"])
```

```
: (0.99, True)
```

```
: 1 #train-test split
  2 model_train=df_data.iloc[:int(df_data.shape[0]*0.95)]
  3 valid=df_data.iloc[int(df_data.shape[0]*0.95):]
  4 y_pred=valid.copy()
```

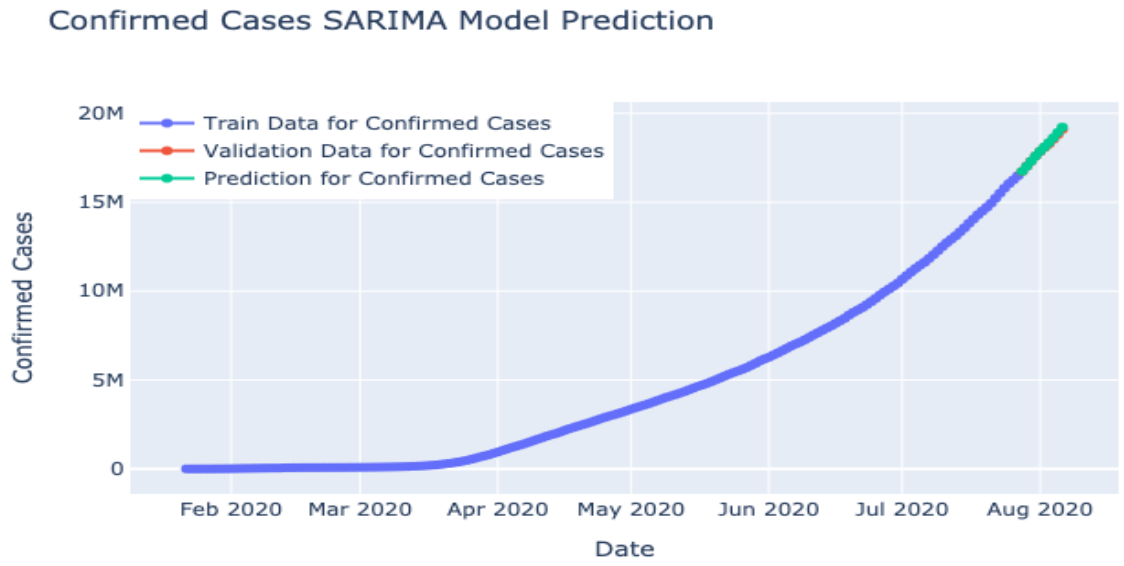
```
: 1 # sarima model.fit()
  2 model_sarima=auto_arima(model_train["Confirmed"],trace=True,error_action='ignore',
  3                         start_p=0,start_q=0,max_p=2,max_q=2,m=7,suppress_warnings=True,stepwise=True,seasonal=True)
  4 result=model_sarima.fit(model_train["Confirmed"])
```

SARIMAX Results

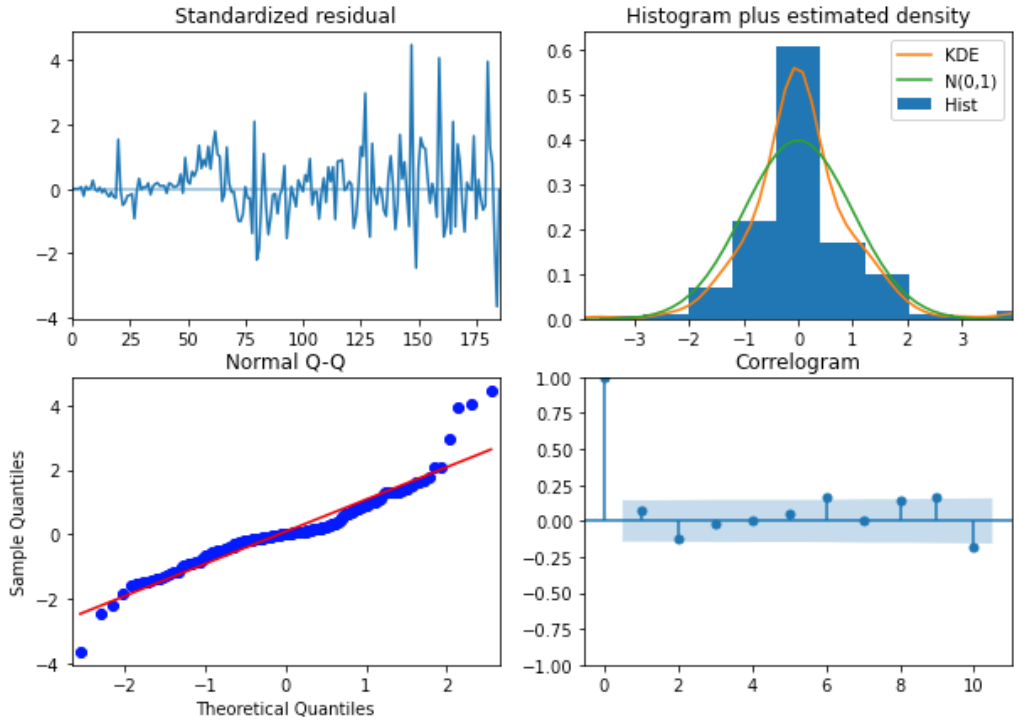
Dep. Variable:	y	**No. Observations:**	188
Model:	SARIMAX(0, 2, 1)x(2, 0, 1, 7)	**Log Likelihood**	-1956.394
Date:	Thu, 27 Aug 2020	**AIC**	3922.787
Time:	14:39:23	**BIC**	3938.916
Sample:	0	**HQIC**	3929.323
	- 188		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.6377	0.057	-11.152	0.000	-0.750	-0.526
ar.S.L7	1.1265	0.119	9.438	0.000	0.893	1.360
ar.S.L14	-0.1280	0.119	-1.077	0.281	-0.361	0.105
ma.S.L7	-0.6826	0.087	-7.852	0.000	-0.853	-0.512
sigma2	7.097e+07	3.13e-09	2.27e+16	0.000	7.1e+07	7.1e+07

Ljung-Box (Q):	109.94	**Jarque-Bera (JB):**	167.62
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	9.37	**Skew:**	0.83
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.34



```
result.plot_diagnostics(figsize=(10,7))
plt.show()
```



Prophet

We use Prophet, a procedure for forecasting time series data based on an additive model where nonlinear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.it works best with time series that have strong seasonal effects and several effects and several seasons of historical data. It is robust to missing data and shifts in the trend and typically handles outliers.

1.train test split
2.using exponential model
3.predicting using. forecast and checking model scores

```
df_confirmed=rename_func(confirmed)
df_confirmed.head()
```

	ds	y
0	2020-01-22	555
1	2020-01-23	654
2	2020-01-24	941
3	2020-01-25	1434
4	2020-01-26	2118

```
#traintest split
model_train=df_confirmed.iloc[:int(df_data.shape[0]*0.95)]
valid=df_confirmed[int(df_data.shape[0]*0.95):]
y_pred=valid.copy()
```

```
model_train.shape,valid.shape
```

```
((188, 2), (10, 2))
```

```
#prophet_model
model=Prophet(interval_width=0.95)
model.add_seasonality(name="Monthly",period=30.42,fourier_order=5)
```

```
<fbprophet.forecaster.Prophet at 0x102564c50>
```

```

# model_fit
model.fit(model_train)
```

```
#forecast
future_dates=model.make_future_dataframe(periods=10)
forecast_confirmed=future_dates.copy()
```

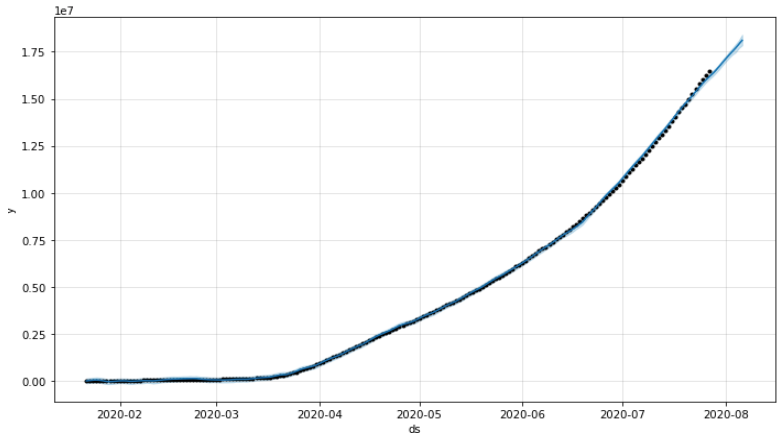
```
prophet_forecast=model.predict(future_dates)

```

```
prophet_forecast.yhat.tail()
```

```
193    1.731164e+07
194    1.750315e+07
195    1.768765e+07
196    1.789309e+07
197    1.810774e+07
Name: yhat, dtype: float64
```

```
## checking accuracy
model_scores.append(np.sqrt(mean_squared_error(valid["y"],prophet_forecast['yhat'].head(valid.shape[0]))))
print("Root Mean Squared Error for Prophet Model: ",np.sqrt(mean_squared_error(valid["y"],prophet_forecast['yhat'].
```

```
Root Mean Squared Error for Prophet Model:  17939649.30044411
```



Summarization of Forecasts using different Models

	Model Name	Root Mean Squared Error
2	SARIMA Model	6.429346e+04
0	Holt's Winter Model	8.106599e+04
1	ARIMA Model	1.027092e+05
3	Facebook's Prophet Model	1.793965e+07

RMSE of SARIMA Model has good accuracy with Rmse 64293.45 and AIC score 3922 so I will be using SARIMA model to forecast covid19 cases.

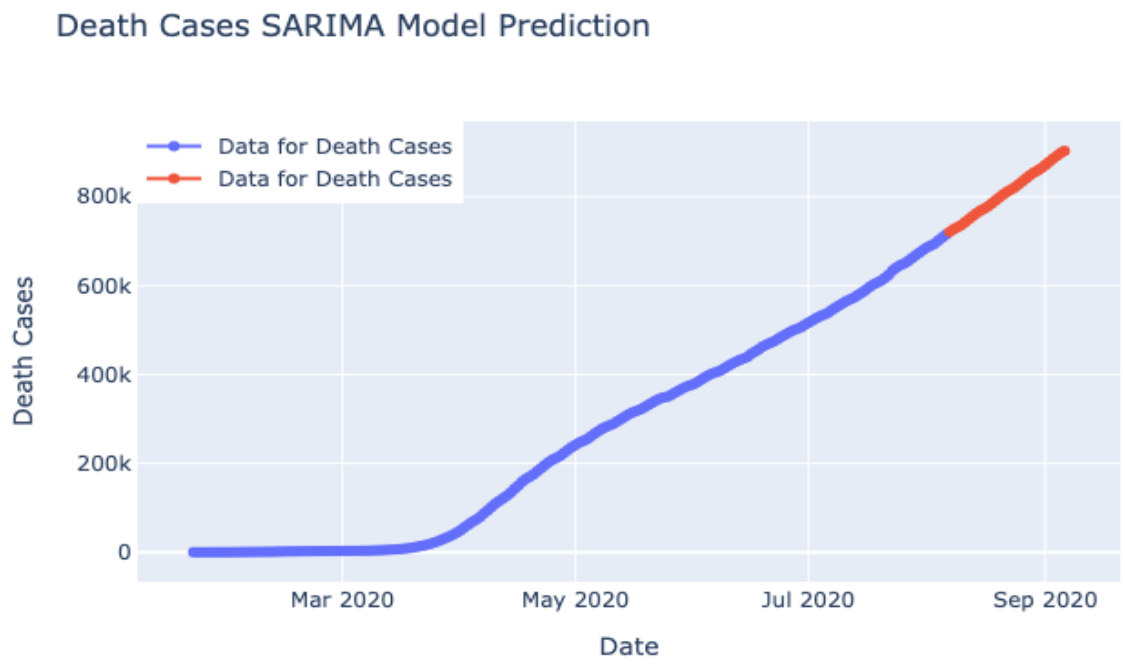
Forecasting 30 days deaths cases with SARIMA MODEL

```
#modelling
model_sarima_death=auto_arima(df_data["Deaths"],trace=True,error_action='ignore',
                      start_p=0,start_q=0,max_p=2,max_q=2,m=7,suppress_warnings=True,stepwise=True,seasonal=True)
model_sarima_death.fit(df_data["Deaths"])
```

```
]:
#forecasting 30 days deaths cases
future_dates=pd.date_range(start="2020-08-07",end="2020-09-06")
future_forecast_deaths=pd.DataFrame(model_sarima_death.predict(31),index=future_dates[0:],columns=["Deaths"]).reset
#future_forecast_deaths
future_forecast_deaths=future_forecast_deaths.rename(columns={"index":"Date"})
future_forecast_deaths.set_index('Date',inplace=True)
future_forecast_deaths.tail(5)
```

]:

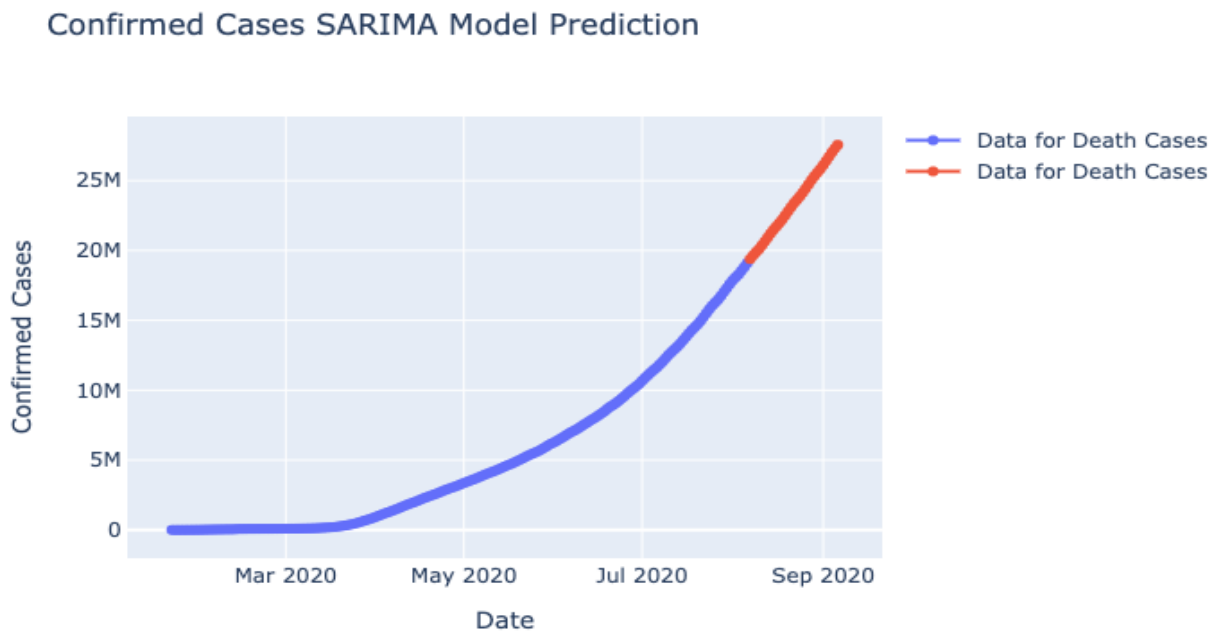
Date	Deaths
2020-09-02	878441.689249
2020-09-03	885467.787406
2020-09-04	892117.312420
2020-09-05	898239.602401
2020-09-06	903348.437187



It seems it decreases first and then increases with the number of days. However, in time series analysis can be done considering endogenous and exogenous variables but I have only considered endogenous variables.

Forecasting 30 days of Confirmed cases with SARIMA model

```
model_sarima=auto_arima(df_data["Confirmed"],trace=True,error_action='ignore',
                        start_p=0,start_q=0,max_p=2,max_q=2,m=7,suppress_warnings=True,stepwise=True,seasonal=True)
model_sarima.fit(df_data["Confirmed"])


future_forecast_confirmed=pd.DataFrame(model_sarima.predict(31),index=future_dates[0:],columns=["Confirmed"]).reset
future_forecast_confirmed=future_forecast_confirmed.rename(columns={"index":"Date"})
future_forecast_confirmed.set_index('Date',inplace=True)
```



Assumptions and Limitations

- Stationarity: The first assumption is that the series are stationary.
- This means that the series are normally distributed, and the mean and variance are constant over a long time period.
- No outliers: We assume that there is no outlier in the series.
- In Univariate Time Series analysis, exogeneous factors are not taken consideration due to which forecasting may differ if considered those factors.

More Ideas to improve model in future

- In this case, only one variable is observed at each time is called ‘Univariate Time Series’.
- If two or more variables are observed at each time is called ‘Multivariate Time Series’. In future I would consider exogenous factor to forecast using Multivariate Time series models.
- In this case, we will focus on the univariate time series for forecasting the cases with Auto SARIMA functionality in python.
- I will use Multivariate Time series models to forecast cases using LSTM RNN for better results with more data.

Conclusion:

- All sources of datasets help in forecasting of covid-19 cases.
- Out of 4 models, SARIMA model performs best with least RMSE and AIC scores.
- South Korea, Germany and New Zealand has managed pandemic better than rest of world.
- Model has forecasted increase of death cases to 903348 and confirmed cases to 27,564,467 by 2020-09-06 worldwide.
- With more ideas, model can be improved in future