Capstone Project

Analysing the trends of COVID-19
with Python

Dataset-
Covid_19_Clean_Complete.csv

Table of Contents

Problem Statement

Given data about COVID-19 patients, write code to visualize the impact and analyse the trend of rate of infection and recovery as well as make predictions about the number of cases expected a week in future based on the current trends.

Project Objective

The primary objective of this project is to analyse historical Covid-19 data and leverage predictive modelling techniques to forecast future trends in terms of confirmed cases, recoveries, deaths, and active cases across different regions. By identifying potential patterns and deriving actionable insights, this analysis aims to contribute to pandemic preparedness and public health planning.

Additionally, the project generates meaningful visualizations to communicate findings effectively. These visualizations will facilitate understanding of trends, support decision-making for policymakers and healthcare providers, and enhance public awareness of potential future scenarios. Through innovative data exploration and visualization techniques, the project intends to simplify complex data and present it in an engaging, intuitive manner.

Data Description

The dataset provides a detailed collection of information related to the Covid-19 pandemic from January 2020 to July 2020. It consists of 49,068 records and 10 columns, offering comprehensive insights into the global spread of the virus. Here's an overview of each column:

- Province/State: Identifies specific states, provinces, or regions within a country, not always available for all records as most of values are null.
- Country/Region: Specifies the country or region where the data was recorded.
- Lat: (Latitude) Geographical latitude of the location, aiding in spatial analysis and mapping.
- Long: (Longitude) Geographical longitude of the location, complementing latitude for mapping purposes.
- Date: Records the specific date when the data was captured, enabling time-series analysis.
- Confirmed: Shows the cumulative number of reported Covid-19 cases in the location up to that date.
- Deaths: Reflects the cumulative number of fatalities due to Covid-19.
- Recovered: Indicates the total number of recoveries, providing insights into the recovery rate.

- Active: Represents ongoing cases, calculated as the difference between confirmed cases and the sum of deaths and recoveries.
- WHO Region: Classifies the data by World Health Organization (WHO) regions, such as Africa, Europe, or the Americas.

The dataset's structure and breadth make it ideal for various analyses, including spatial and temporal patterns, predictive modelling, and impactful visualizations.

Data Preprocessing Steps And Visualisations

1. Basic information about the Dataset:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49068 entries, 0 to 49067
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Province/State  14664 non-null  object
 1   Country/Region  49068 non-null  object
 2   Lat             49068 non-null  float64
 3   Long            49068 non-null  float64
 4   Date            49068 non-null  object
 5   Confirmed       49068 non-null  int64
 6   Deaths          49068 non-null  int64
 7   Recovered       49068 non-null  int64
 8   Active          49068 non-null  int64
 9   WHO Region      49068 non-null  object
dtypes: float64(2), int64(4), object(4)
```

The datatypes for each column in the Covid-19 dataset:

- Province/State: This column typically uses an object datatype.

- Country/Region: Similar to the previous column, this is stored as a string datatype.
- Lat (Latitude): Latitude values are typically in the form of float datatype.
- Long (Longitude): Longitude values are also stored as a float datatype.
- Date: This column uses a object datatype to represent specific calendar dates.
- Confirmed: This numeric column uses an integer datatype.
- Deaths: Similar to Confirmed, this uses an integer datatype.
- Recovered: This numeric column is represented as an integer datatype.
- Active: Active cases are calculated and stored as an integer datatype.
- WHO Region: This column uses a object datatype to classify entries.

2. Renaming few columns:

```
# Renaming columns
df.rename(columns={'Province/State':'state','Country/Region':'country'},inplace=True)
df.head()
```

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
0	NaN	Afghanistan	33.93911	67.709953	2020-01-22	0	0	0	0	Eastern Mediterranean
1	NaN	Albania	41.15330	20.168300	2020-01-22	0	0	0	0	Europe
2	NaN	Algeria	28.03390	1.659600	2020-01-22	0	0	0	0	Africa
3	NaN	Andorra	42.50630	1.521800	2020-01-22	0	0	0	0	Europe
4	NaN	Angola	-11.20270	17.873900	2020-01-22	0	0	0	0	Africa

- The column names in the dataset, such as "Province/State" and "Country/Region," are being renamed to simpler terms like "state" and "country."
- This is often done to make column names easier to reference and consistent with coding conventions
- After renaming, the head() function is used to display the first five rows of the dataset.

3. Getting value count of Date field:

```
# getting value count of Date field
df['Date'].value_counts()
```

Date	count
2020-01-22	261
2020-01-23	261
2020-01-24	261
2020-01-25	261
2020-01-26	261

The code snippet counts occurrences of each unique date in the 'Date' column of our DataFrame.

- ‘Date’: This lists the distinct dates from the dataset, ranging from January 22, 2020, to July 27, 2020.
- 'value_counts()': This shows the frequency of each date, which is consistently 261 for the dates in the output, indicating that each of these dates appears 261 times in the DataFrame.

4. Checking max date and populating rows with max date in top:

```
# check max date
df['Date'].max()
```

```
'2020-07-27'
```

- It determines the latest date present in the dataset by applying the max() function to the ‘Date’ column.
- The output ‘2020-07-27’ indicates that the most recent date in the dataset is July 27, 2020.

```
# poulating rows with max date in top
top = df[df['Date'] == '2020-07-27']
top
```

Filters a DataFrame to extract rows where the 'Date' column equals '2020-07-27'—the maximum date previously identified.

- The condition df['Date']=='2020-07-27'selects rows where the 'Date' matches.
- The resulting subset is assigned to a dataframe called top.

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
48807	NaN	Afghanistan	33.939110	67.709953	2020-07-27	36263	1269	25198	9796	Eastern Mediterranean
48808	NaN	Albania	41.153300	20.168300	2020-07-27	4880	144	2745	1991	Europe
48809	NaN	Algeria	28.033900	1.659600	2020-07-27	27973	1163	18837	7973	Africa
48810	NaN	Andorra	42.506300	1.521800	2020-07-27	907	52	803	52	Europe
48811	NaN	Angola	-11.202700	17.873900	2020-07-27	950	41	242	667	Africa
...	...	...	...	...	...	...	...	...	...	...
49063	NaN	Sao Tome and Principe	0.186400	6.613100	2020-07-27	865	14	734	117	Africa
49064	NaN	Yemen	15.552727	48.516388	2020-07-27	1691	483	833	375	Eastern Mediterranean
49065	NaN	Comoros	-11.645500	43.333300	2020-07-27	354	7	328	19	Africa
49066	NaN	Tajikistan	38.861000	71.276100	2020-07-27	7235	60	6028	1147	Europe
49067	NaN	Lesotho	-29.610000	28.233600	2020-07-27	505	12	128	365	Africa

261 rows × 10 columns

5. Grouping all rows based on country

```
# Grouping all rows based on country
top.groupby(by='country')[['Confirmed','Deaths','Recovered', 'Active']].sum().reset_index()
```

- Grouping by 'Country': The groupby() function is used on the 'country' column to group rows by each unique country name. Grouping allows aggregation of data for each country.
- Aggregating Specific Columns: The columns ‘Confirmed’, ‘Deaths’, ‘Recovered’, and ‘Active’ are selected for summation within each group i.e. for each country. This sums up the Covid-19 metrics for each country.
- Resetting Index: The reset_index() function converts the grouped data back into a standard DataFrame, with the ‘country’ column re-integrated as a regular column instead of an index.
- Purpose: This operation aggregates Covid-19 statistics on basis of each country, making it easier to analyse total cases, deaths, recoveries, and active cases for each country.

	country	Confirmed	Deaths	Recovered	Active
0	Afghanistan	36263	1269	25198	9796
1	Albania	4880	144	2745	1991
2	Algeria	27973	1163	18837	7973
3	Andorra	907	52	803	52
4	Angola	950	41	242	667
...	...	...	...	...	...
182	West Bank and Gaza	10621	78	3752	6791
183	Western Sahara	10	1	8	1
184	Yemen	1691	483	833	375
185	Zambia	4552	140	2815	1597
186	Zimbabwe	2704	36	542	2126

187 rows × 5 columns

6. Populating all confirmed case in confirmed dataset:

```
# Populating all confirmed case in confirmed dataset
confirmed = df.groupby(by='Date').sum()['Confirmed'].reset_index()
confirmed
```

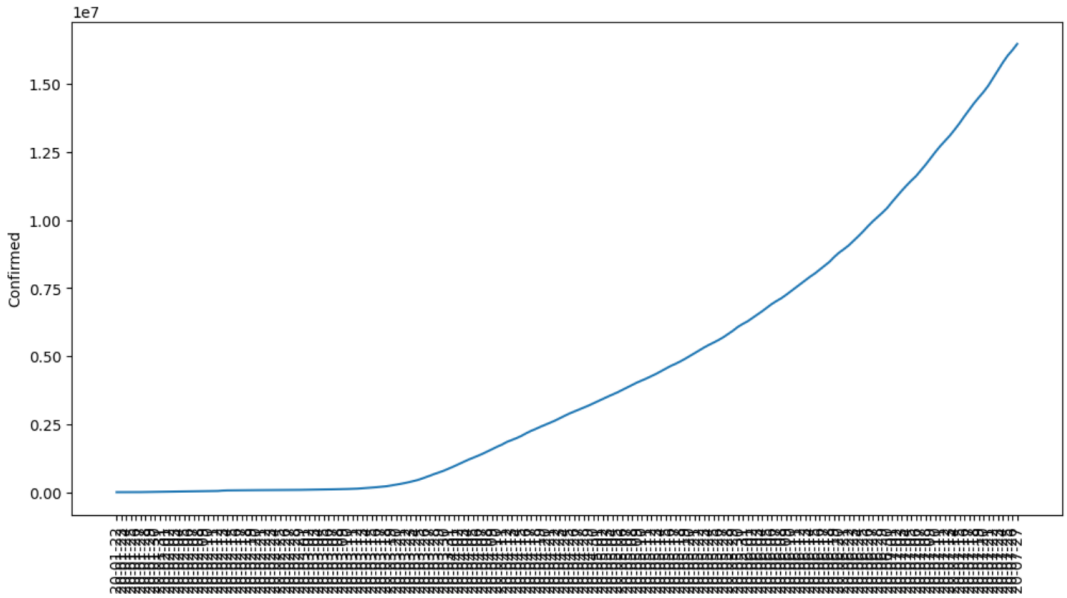
Processes a dataset to calculate the total number of confirmed Covid-19 cases grouped by the ‘Date’ column.

- df.groupby(by='Date') groups the dataset by unique dates in the 'Date' column, essentially clustering all entries recorded on the same date.
- sum()['Confirmed'] within each group (i.e., for each date), the sum() function computes the total number of confirmed cases. This results in a series where dates serve as the index, and the corresponding values represent the daily totals.
- Using reset_index() the dataset is converted back into a standard DataFrame with the date as a column rather than an index. This structure is ideal for further analysis or visualization.
- The dataframe confirmed stores the resulting DataFrame, which contains two columns: 'Date' and the total 'Confirmed' cases for each date. This information is essential for analysing and visualizing the temporal progression of Covid-19 cases.

	Date	Confirmed
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
...	...	...
183	2020-07-23	15510481
184	2020-07-24	15791645
185	2020-07-25	16047190
186	2020-07-26	16251796
187	2020-07-27	16480485

Plotting confirmed cases:

```
# Visualizing confirmed case
plt.figure(figsize=(12,6))
sns.lineplot(x='Date',y='Confirmed',data=confirmed)
plt.xticks(rotation=90)
plt.show()
```



- Trend: The graph shows a steep exponential growth in confirmed cases, starting from lower values and peaking sharply towards the end of the timeline.
- Scale: The confirmed cases range from 0 to approximately 15 million, reflecting a massive surge as the pandemic progresses.
- Timeline: The graph represents a continuous time series from the beginning to the end of the dataset (22 January 2020 to 27 July 2020).

7. Top 10 countries with the most recovered cases

```
top_10_recovered = df.groupby(by='country')['Recovered'].sum().sort_values(ascending=False).head(10).reset_index()
top_10_recovered
```

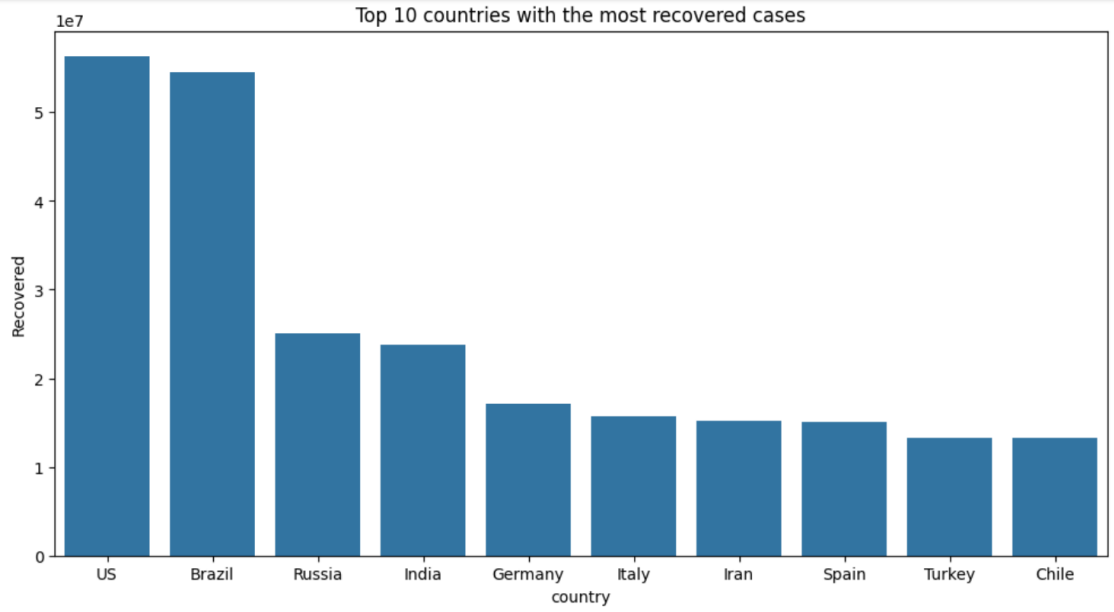
	country	Recovered
0	US	56353416
1	Brazil	54492873
2	Russia	25120448
3	India	23783720
4	Germany	17107839
5	Italy	15673910
6	Iran	15200895
7	Spain	15093583
8	Turkey	13345389
9	Chile	13292593

Identifying the top 10 countries with the highest number of recovered Covid-19 cases.

- US (56,353,416 recoveries): Leading the list with the highest number of recoveries.
- Brazil with 54,492,873 recoveries stands second in the list.
- Russia ranks 3rd with 25,120,448 recovered patients.
- India is in the fourth position having 23,783,720 recoveries.

Plotting Recovered patient’s graph

```
# Visualizing recovered case
plt.figure(figsize=(12,6))
sns.barplot(x='country',y='Recovered',data=top_10_recovered)
plt.title('Top 10 countries with the most recovered cases')
plt.show()
```



8. Top 10 Countries with deaths

```
top_10_deaths = df.groupby(by='country')['Deaths'].sum().sort_values(ascending=False).head(10).reset_index()
top_10_deaths
```

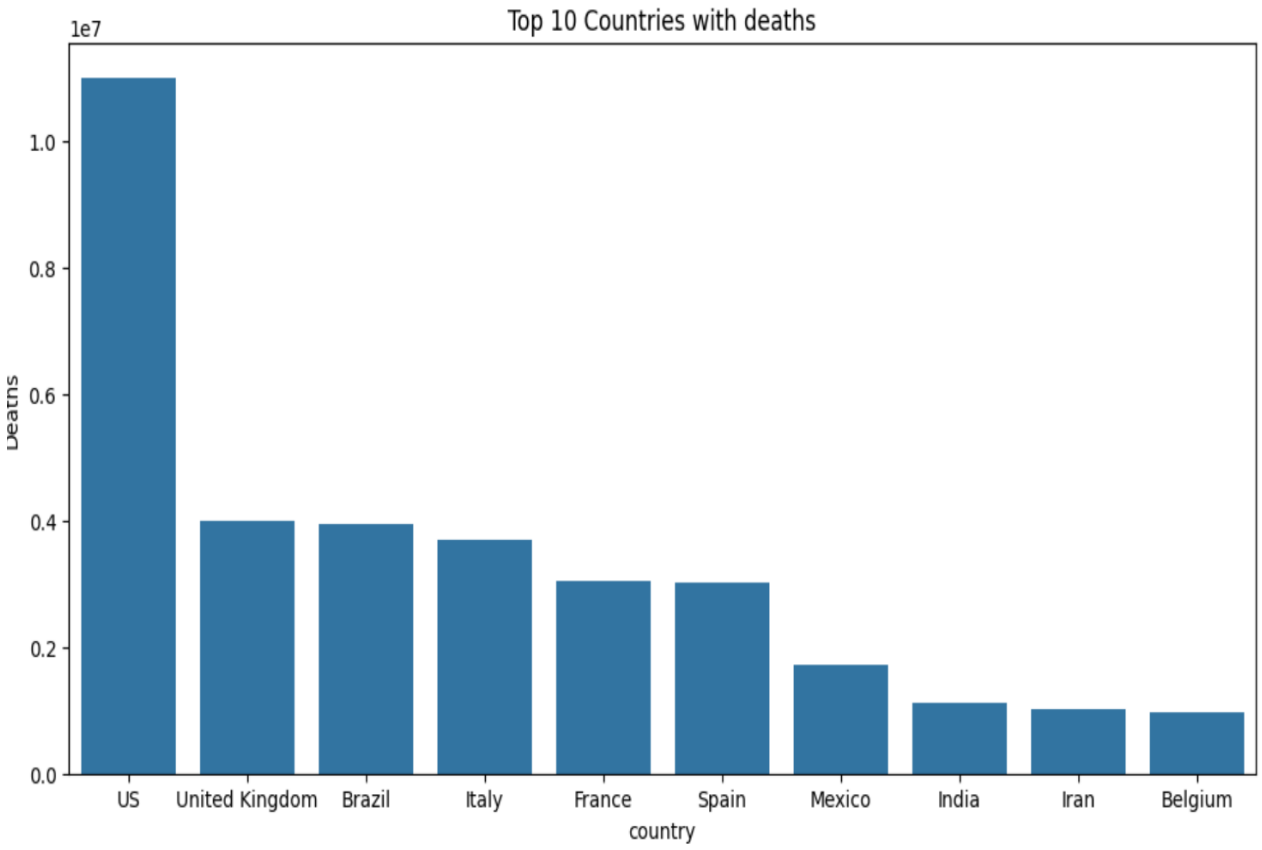
	country	Deaths
0	US	11011411
1	United Kingdom	3997775
2	Brazil	3938034
3	Italy	3707717
4	France	3048524
5	Spain	3033030
6	Mexico	1728277
7	India	1111831
8	Iran	1024136
9	Belgium	963679

Identify the top 10 countries with the highest recorded deaths due to Covid-19.

- US leading the list with 11,011,411 deaths.
- United Kingdom ranks second highest with 3,997,775 deaths.
- Brazil ranks 3rd with 3,938,034 deaths.
- Other countries include Italy, France, Spain, Mexico, India, Iran, and Belgium.

Plotting Death cases:

```
# visualising death cases
plt.figure(figsize=(16,10))
sns.barplot(x='country',y='Deaths',data=top_10_deaths)
plt.title('Top 10 Countries with deaths')
plt.show()
```



9. Top 10 countries with most active cases

```
top_10_active = df.groupby(by='country')['Active'].sum().sort_values(ascending=False).head(10).reset_index()
top_10_active
```

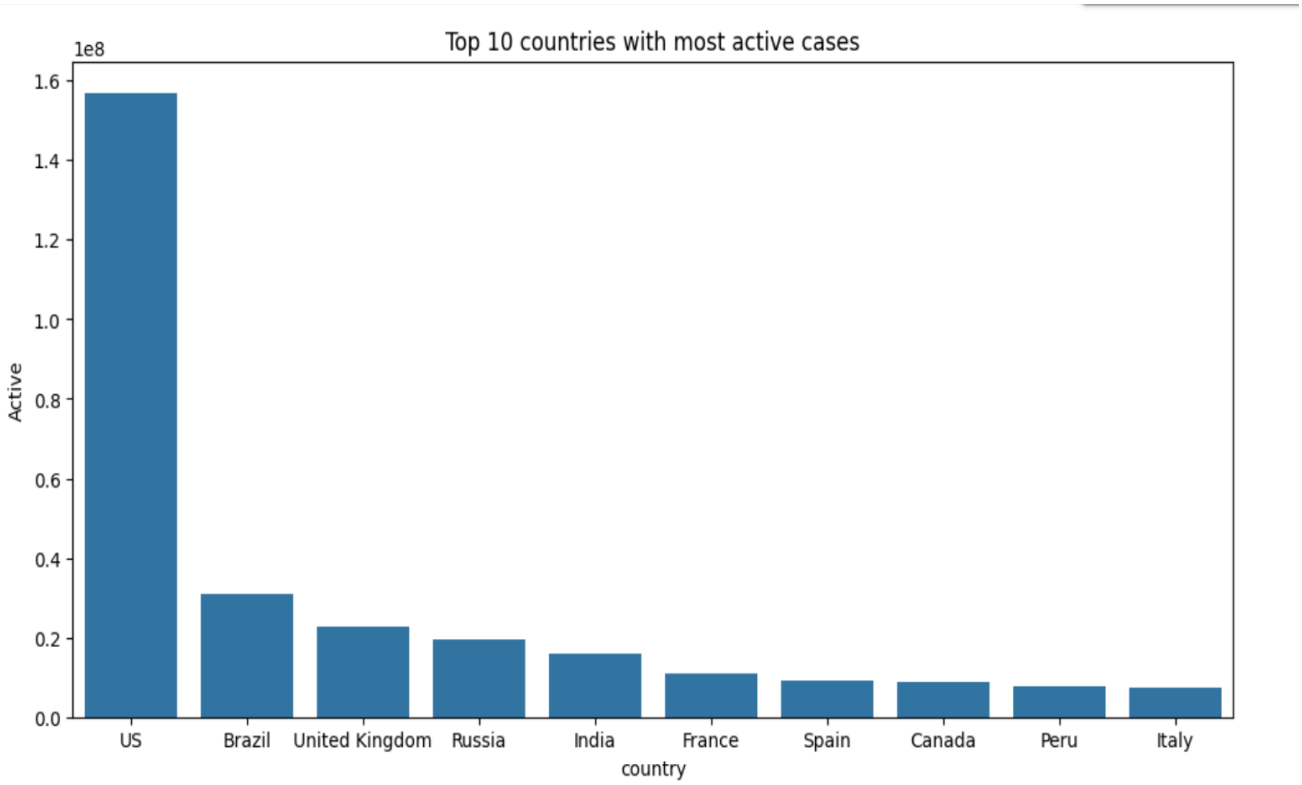
	country	Active
0	US	156981121
1	Brazil	31094060
2	United Kingdom	22624595
3	Russia	19668578
4	India	15987913
5	France	10980287
6	Spain	9277432
7	Canada	8656985
8	Peru	7748957
9	Italy	7363518

The resulting table lists the top 10 countries with the highest total active cases

- US: Leads the list with 156,981,121 active cases.
- Brazil: Comes second with 31,094,060 active cases.
- UK: ranked 3rd with 22,624,595 active cases.

Plotting active cases data:

```
# Visualizing active case
plt.figure(figsize=(12,6))
sns.barplot(x='country',y='Active',data=top_10_active)
plt.title('Top 10 countries with most active cases')
plt.show()
```



Other countries include the Russia, India, France, Spain, Canada, Peru, & Italy belong to the group of top 10 nations having most active cases.

This analysis focuses on identifying countries facing the most active Covid-19 cases, which is vital for understanding the distribution of the pandemic's ongoing burden. It provides critical insights for resource allocation, healthcare strategies, and international collaboration.

10. Country based analysis of US, China and India

Creating US dataset which have data related to covid-19 in USA

```
# Creating US dataset which have country as US
US = df[df['country'] == 'US']
US.head()
```

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
223	NaN	US	40.0	-100.0	2020-01-22	1	0	0	1	Americas
484	NaN	US	40.0	-100.0	2020-01-23	1	0	0	1	Americas
745	NaN	US	40.0	-100.0	2020-01-24	2	0	0	2	Americas
1006	NaN	US	40.0	-100.0	2020-01-25	2	0	0	2	Americas
1267	NaN	US	40.0	-100.0	2020-01-26	5	0	0	5	Americas

Filters a dataset to create a subset of records where the 'country' column matches ‘US’.

```
US = US.groupby(by='Date')[['Confirmed', 'Deaths', 'Recovered', 'Active']].sum().reset_index()
US
```

Above code snippet performs grouping and aggregation operations to summarize Covid-19 data for the United States by date.

- US = US.groupby(by='Date'): Groups the dataset by the 'Date' column, clustering all rows with the same date.

- [['Confirmed', 'Deaths', 'Recovered', 'Active']]: Specifies that only these four columns (Confirmed, Deaths, Recovered, Active) will be included in the aggregation.
- sum(): sums up the values for each of the selected columns within each group for each unique date. This calculates the daily totals for these metrics.
- reset_index(): Converts the grouped data back into a standard DataFrame, reintroducing the 'Date' column as a regular column rather than an index.

The resulting DataFrame, containing the aggregated data for each date, is stored in the new dataframe named US.

	Date	Confirmed	Deaths	Recovered	Active
0	2020-01-22	1	0	0	1
1	2020-01-23	1	0	0	1
2	2020-01-24	2	0	0	2
3	2020-01-25	2	0	0	2
4	2020-01-26	5	0	0	5
...	...	...	...	...	...
183	2020-07-23	4038816	144430	1233269	2661117
184	2020-07-24	4112531	145560	1261624	2705347
185	2020-07-25	4178970	146465	1279414	2753091
186	2020-07-26	4233923	146935	1297863	2789125
187	2020-07-27	4290259	148011	1325804	2816444

188 rows × 5 columns

Creating China dataset which have data related to covid-19 in China.

```
# Creating China dataset which have country as China
China = df[df['country'] == 'China']
China.head()
```

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
48	Anhui	China	31.8257	117.2264	2020-01-22	1	0	0	1	Western Pacific
49	Beijing	China	40.1824	116.4142	2020-01-22	14	0	0	14	Western Pacific
50	Chongqing	China	30.0572	107.8740	2020-01-22	6	0	0	6	Western Pacific
51	Fujian	China	26.0789	117.9874	2020-01-22	1	0	0	1	Western Pacific
52	Gansu	China	35.7518	104.2861	2020-01-22	0	0	0	0	Western Pacific

Filters a dataset to create a subset of records where the 'country' column matches ‘China.

```
China = China.groupby(by='Date')[['Confirmed', 'Deaths', 'Recovered', 'Active']].sum().reset_index()
China
```

Above code snippet performs grouping and aggregation operations to summarize Covid-19 data for the China by date.

- China = China.groupby(by='Date'): Groups the dataset by the 'Date' column, clustering all rows with the same date.

- [['Confirmed', 'Deaths', 'Recovered', 'Active']]: Specifies that only these four columns (Confirmed, Deaths, Recovered, Active) will be included in the aggregation.
- sum(): sums up the values for each of the selected columns within each group for each unique date. This calculates the daily totals for these metrics.
- reset_index(): Converts the grouped data back into a standard DataFrame, reintroducing the 'Date' column as a regular column rather than an index.

The resulting DataFrame, containing the aggregated data for each date, is stored in the new dataframe named China.

	Date	Confirmed	Deaths	Recovered	Active
0	2020-01-22	548	17	28	503
1	2020-01-23	643	18	30	595
2	2020-01-24	920	26	36	858
3	2020-01-25	1406	42	39	1325
4	2020-01-26	2075	56	49	1970
...	...	...	...	...	...
183	2020-07-23	86045	4649	78701	2695
184	2020-07-24	86202	4650	78745	2807
185	2020-07-25	86381	4652	78813	2916
186	2020-07-26	86570	4652	78862	3056
187	2020-07-27	86783	4656	78869	3258

188 rows × 5 columns

Creating India dataset which have data related to covid-19 in India.

```
# Creating India dataset
India = df[df['country'] == 'India']
India.head()
```

	state	country	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
129	NaN	India	20.593684	78.96288	2020-01-22	0	0	0	0	South-East Asia
390	NaN	India	20.593684	78.96288	2020-01-23	0	0	0	0	South-East Asia
651	NaN	India	20.593684	78.96288	2020-01-24	0	0	0	0	South-East Asia
912	NaN	India	20.593684	78.96288	2020-01-25	0	0	0	0	South-East Asia
1173	NaN	India	20.593684	78.96288	2020-01-26	0	0	0	0	South-East Asia

Filters a dataset to create a subset of records where the 'country' column matches ‘China.

```
India = India.groupby(by='Date')[['Confirmed', 'Deaths', 'Recovered', 'Active']].sum().reset_index()
India
```

Above code snippet performs grouping and aggregation operations to summarize Covid-19 data for the China by date.

- India = India.groupby(by='Date'): Groups the dataset by the 'Date' column, clustering all rows with the same date.
- [['Confirmed', 'Deaths', 'Recovered', 'Active']]: Specifies that only these four columns (Confirmed, Deaths, Recovered, Active) will be included in the aggregation.

- sum(): sums up the values for each of the selected columns within each group for each unique date. This calculates the daily totals for these metrics.
- reset_index(): Converts the grouped data back into a standard DataFrame, reintroducing the 'Date' column as a regular column rather than an index.

The resulting DataFrame, containing the aggregated data for each date, is stored in the new dataframe named India.

	Date	Confirmed	Deaths	Recovered	Active
0	2020-01-22	0	0	0	0
1	2020-01-23	0	0	0	0
2	2020-01-24	0	0	0	0
3	2020-01-25	0	0	0	0
4	2020-01-26	0	0	0	0
...	...	...	...	...	...
183	2020-07-23	1288108	30601	817209	440298
184	2020-07-24	1337024	31358	849432	456234
185	2020-07-25	1385635	32060	885573	468002
186	2020-07-26	1435616	32771	917568	485277
187	2020-07-27	1480073	33408	951166	495499

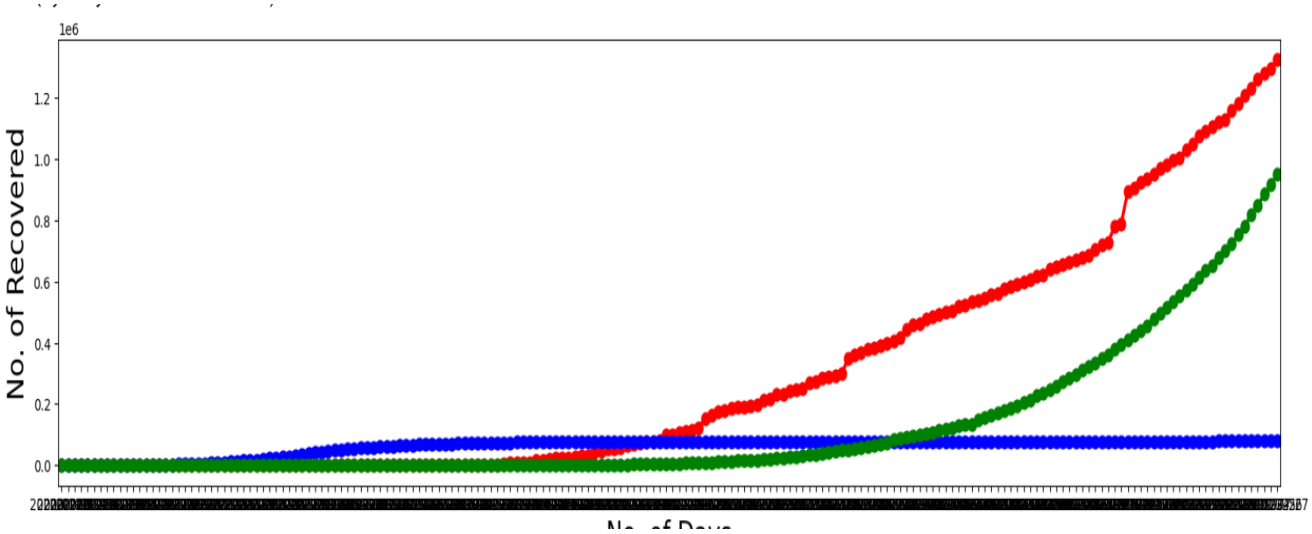
188 rows × 5 columns

11. Compare USA, China, India countries on the basis of Recovered cases

```
# Visualizing using point plot
plt.figure(figsize=(20,5))

sns.pointplot(x='Date',y='Recovered',data=US, color='Red')
sns.pointplot(x='Date',y='Recovered',data=China, color='Blue')
sns.pointplot(x='Date',y='Recovered',data=India,color='Green')
plt.xlabel('No. of Days', fontsize = 20)
plt.ylabel('No. of Recovered', fontsize = 20)
```

Visualised Plot



The graph displays trends in the number of Covid-19 recoveries over a period of time.

- US: Shows a sharp and steep increase in recoveries over time, indicating a rapid improvement or recovery trend.
- India: Reflects a moderate upward trend in recoveries, suggesting steady progress.

- China: Displays a relatively flat trend, showing little change in recoveries compared to the other datasets.

Insights:

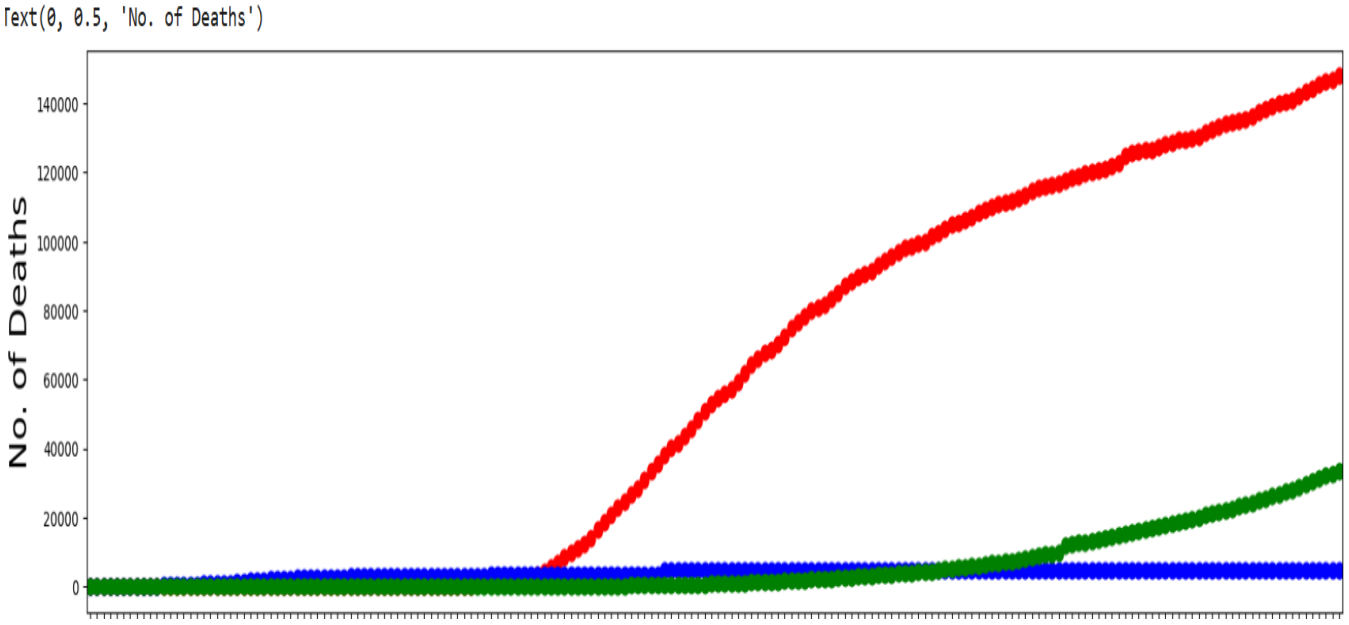
- Cases in USA indicates significant improvements in recovery rates, possibly due to effective medical interventions or containment measures in specific contexts.
- The other two lines (China & India) highlight slower or more stable recovery trends, which could be attributed to varying regional or systemic factors.

12. Compare USA, China, India countries on the basis of Death cases

```
# Visualizing using point plot
plt.figure(figsize=(20,5))

sns.pointplot(x='Date',y='Deaths',data=US, color='Red')
sns.pointplot(x='Date',y='Deaths',data=China, color='Blue')
sns.pointplot(x='Date',y='Deaths',data=India,color='Green')
plt.xlabel('No. of Days', fontsize = 20)
plt.ylabel('No. of Deaths', fontsize = 20)
```

Plotting the Graph:



The graph contains three distinct lines, each corresponding to one country:

- US: Represents the United States and shows a significantly steep upward trend, indicating a high number of deaths over time.
- China: Represents China, where the line remains relatively flat, suggesting a far lower death toll compared to the other countries.
- India: Represents India, showing a moderate upward trend, reflecting a growing number of fatalities but less dramatic compared to the US.

Insights:

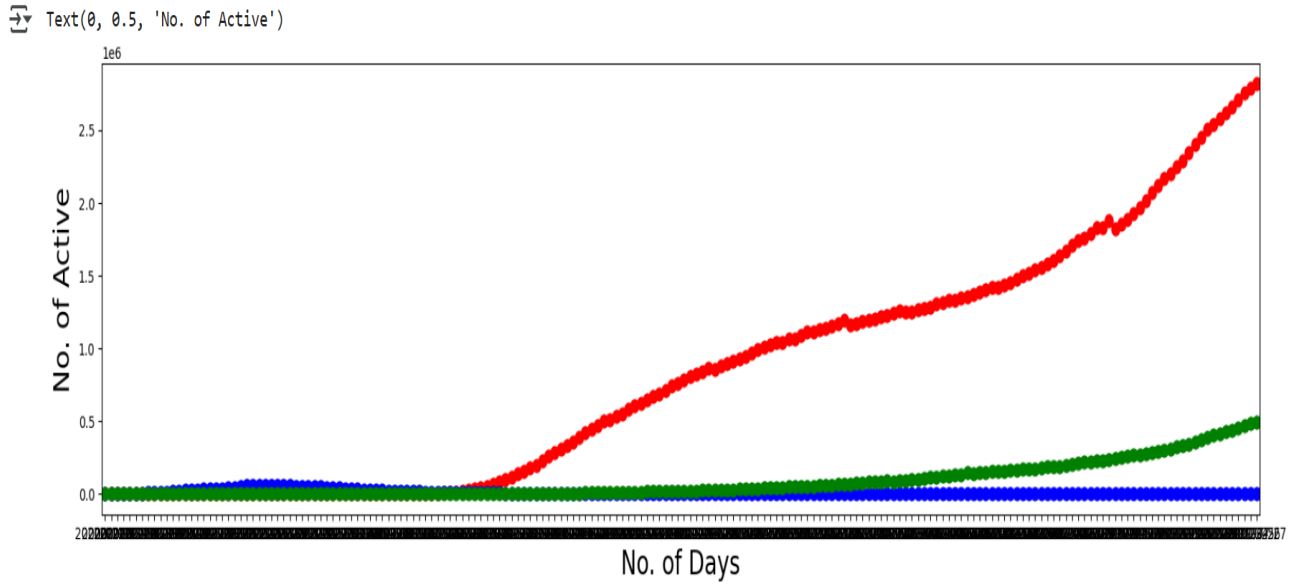
- United States: The steep curve indicates that the US faced the most severe impact in terms of fatalities.
- China: The flat line implies effective containment measures or early control of the pandemic, resulting in relatively fewer deaths.
- India: The green line reflects a consistent increase in deaths, though not as steep as the US, highlighting the challenges faced by the healthcare system over time.

13. Compare USA, China, India countries on the basis of Active cases:

```
# visualize using point plot
plt.figure(figsize=(20,5))

sns.pointplot(x='Date',y='Active',data=US, color='Red')
sns.pointplot(x='Date',y='Active',data=China, color='Blue')
sns.pointplot(x='Date',y='Active',data=India,color='Green')
plt.xlabel('No. of Days', fontsize = 20)
plt.ylabel('No. of Active', fontsize = 20)
```

Plotting the Graph:



The graph visualizes the trends in active Covid-19 cases over time for three countries- USA, China, and India-using a point plot.

- US: Displays a sharply increasing trend, indicating a substantial and sustained rise in active Covid-19 cases, highlighting the severe impact of the pandemic on the US.
- China: Remains relatively stable, suggesting early containment measures or successful control over the virus's spread.
- India: Demonstrates moderate growth in active cases over time, indicating a consistent but less dramatic trend compared to the US.

Choosing the Algorithm for the Project

The Prophet library of python is a powerful tool for time-series forecasting, developed by Facebook's Core Data Science team. It is specifically designed to handle datasets with strong seasonal patterns and trends.

About Prophet Library:

1. Additive Model: Prophet uses an additive model to fit non-linear trends with components like yearly, weekly, and daily seasonality. It can also incorporate holiday effects, making it versatile for real-world applications.
2. Robustness: It is robust to missing data, outliers, and sudden changes in trends, which makes it ideal for handling noisy datasets.
3. Ease of Use: Prophet follows a simple workflow to fit the model to historical data, generate future predictions using the make_future_dataframe() method.
4. Flexibility: The library supports custom seasonality and changepoints, allowing users to tailor forecasts to specific scenarios.

I have chosen the Prophet Library for this project for the following reasons:

- Seasonality and Trends: Covid-19 data often exhibits seasonal patterns (e.g., waves of infections). Prophet's ability to model these trends makes it suitable for forecasting future cases, recoveries, and deaths.
- Handling Missing Data: Covid-19 datasets may have gaps due to reporting inconsistencies. Prophet's robustness ensures reliable predictions despite missing entries.
- Ease of Implementation: With its straightforward API, Prophet allows quick setup and forecasting, making it accessible for projects with tight timelines.
- Visualization: Prophet provides intuitive visualizations of forecasts and their components, helping communicate insights effectively.

Assumptions

Assumptions are essential for defining the scope and ensuring clarity in any project. For your Covid-19 data prediction and visualization project, here are possible assumptions that are considered:

1. Data-Related Assumptions:

- The dataset is assumed to be accurate, complete, and represent real-world scenario, despite potential inconsistencies in reporting across regions.
- Future data trends are assumed to follow similar reporting standards as those observed in the dataset's timeframe.

2. Model Building Assumptions:

- The patterns observed in the historical data like growth rates, recovery trends, seasonal effects will continue in a similar manner during the forecast period specially with no rapid exponential rate of increase in infections.
- External factors like unexpected medical breakthroughs, drastic policy changes, or new virus variants are not accounted for in the prediction model.

3. Technical Assumptions:

- The Prophet library is assumed to handle seasonality, changepoints, and other temporal patterns efficiently for generating reliable predictions.
- The data is assumed to remain in the same structure, with no missing or invalid entries during preprocessing and modelling.

4. Visualization Assumptions:

- Visualizations are assumed to effectively communicate trends and insights aiding decision-making.
- The scales used in graphs will appropriately represent the dataset, avoiding misinterpretation.

Model Evaluation and Technique

Model evaluation and techniques refer to the process of assessing how well a predictive model performs on unseen data. This is critical to ensure the model's accuracy, reliability, and generalization ability. This involves using metrics and methods to measure the performance of a trained model. Key elements include:

- Evaluation Metrics: Mean Absolute Error (MAE) measures the average magnitude of errors, Mean Squared Error (MSE) captures the average squared difference between predictions and actual values.
- Classification Tasks: Accuracy defines percentage of correctly predicted outcomes, Confusion Matrix showing the breakdown of true/false positive and negative predictions.
- Cross-Validation: The dataset is divided into multiple parts (folds). The model is trained on some folds and tested on the remaining ones. This process ensures the model performs well across all data segments and avoids overfitting.
- Rolling Forecast Origin: A rolling window approach ensures that new data points are continuously added to the training set to predict future values.

- Visual Validation: Forecasts are compared visually with actual values over time to evaluate their accuracy.

By combining the right metrics and evaluation techniques, we can assess how well your model predicts future Covid-19 trends and refine it for better performance.

1. Creating new dataset of Confirmed case.

```
# creating new dataset of Confirmed case
confirmed = df1.groupby(by='Date')['Confirmed'].sum().reset_index()
confirmed
```

	Date	Confirmed
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
...	...	...
183	2020-07-23	15510481
184	2020-07-24	15791645
185	2020-07-25	16047190
186	2020-07-26	16251796
187	2020-07-27	16480485

188 rows × 2 columns

Insights:

- The dataset starts from January 22, 2020, with 555 confirmed cases, and spans through July 27, 2020, ending with a cumulative total of 16,480,485 confirmed cases.
- A sample of the dataset showcases a continuous increase in confirmed cases, reflecting the global progression of the pandemic.

2. Creating new dataset of Deaths cases

```
# creating new dataset of Deaths cases
deaths = df1.groupby(by='Date')['Deaths'].sum().reset_index()
deaths
```

	Date	Deaths
0	2020-01-22	17
1	2020-01-23	18
2	2020-01-24	26
3	2020-01-25	42
4	2020-01-26	56
...	...	...
183	2020-07-23	633506
184	2020-07-24	639650
185	2020-07-25	644517
186	2020-07-26	648621
187	2020-07-27	654036

- The dataset starts from January 22, 2020, with 17 deaths, and spans through July 27, 2020, ending with a cumulative total of 654,036 deaths.

3. Creating new dataset of Recovered cases

```
recovered = df1.groupby(by='Date')['Recovered'].sum().reset_index()
recovered
```

	Date	Recovered
0	2020-01-22	28
1	2020-01-23	30
2	2020-01-24	36
3	2020-01-25	39
4	2020-01-26	52
...	...	...
183	2020-07-23	8710969
184	2020-07-24	8939705
185	2020-07-25	9158743
186	2020-07-26	9293464
187	2020-07-27	9468087

188 rows × 2 columns

- The dataset starts from January 22, 2020, with 28 recovered cases, and spans through July 27, 2020, ending with a cumulative total of 9,468,087 recovered cases.

4. Creating new dataset of Active cases

```
# creating new dataset of Active cases
active = df1.groupby(by='Date')['Active'].sum().reset_index()
active
```

	Date	Active
0	2020-01-22	510
1	2020-01-23	606
2	2020-01-24	879
3	2020-01-25	1353
4	2020-01-26	2010
...	...	...
183	2020-07-23	6166006
184	2020-07-24	6212290
185	2020-07-25	6243930
186	2020-07-26	6309711
187	2020-07-27	6358362

- The dataset starts from January 22, 2020, with 510 active cases, and spans through July 27, 2020, ending with a cumulative total of 6,358,362 active cases.

5. Renaming columns:

```
# renaming confirmed columns to ds and y
confirmed.columns = ['ds','y']
confirmed
```

- This line changes the column names of the confirmed DataFrame.
- The first column, likely the 'Date' column, is renamed to 'ds' (short for "date stamp").
- The second column, likely the 'Confirmed' cases column, is renamed to 'y' (notation for the dependent variable or target column).

	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
...	...	...
183	2020-07-23	15510481
184	2020-07-24	15791645
185	2020-07-25	16047190
186	2020-07-26	16251796
187	2020-07-27	16480485

188 rows × 2 columns

5.Importing Libraries:

```
# importing prophet library
from prophet import Prophet
```

- Imports the Prophet library in Python, enabling you to use its powerful functionalities for time-series forecasting.

```
m = Prophet()
m.fit(confirmed)
```

- A new Prophet object is created and stored in the variable m.
- The fit() method trains the Prophet model on the dataset confirmed.

6. Making prediction for next 7 dates

```
# Making prediction for next 7 dates
future = m.make_future_dataframe(periods=7)

forecast = m.predict(future)
forecast
```

- The above code generates a future dataframe with 7 additional periods (days) beyond the existing dataset.
- The period=7 argument specifies the number of future days to be predicted.
- The future variable stores this dataframe, which includes both historical dates and the new dates for forecasting.
- The predict() method processes the future dataframe using the previously trained Prophet model (m).
- It generates forecasts for all dates in the dataframe, including the 7 future dates, and stores the results in the forecast variable.

```
forecast[['ds','yhat','yhat_upper','yhat_lower']]
```

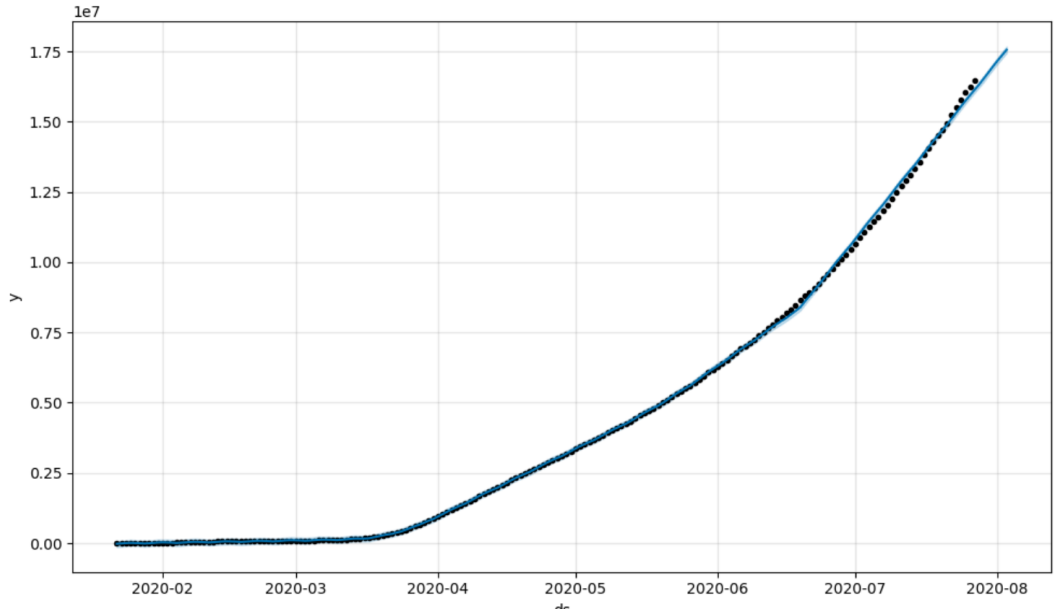
	ds	yhat	yhat_upper	yhat_lower
0	2020-01-22	-2.067684e+04	8.368616e+04	-1.307194e+05
1	2020-01-23	-8.050948e+03	9.181782e+04	-1.195123e+05
2	2020-01-24	5.827455e+03	1.030481e+05	-1.020836e+05
3	2020-01-25	1.217668e+04	1.210755e+05	-9.376257e+04
4	2020-01-26	8.405020e+03	1.070187e+05	-9.382363e+04
...	...	...	...	...
190	2020-07-30	1.674391e+07	1.685384e+07	1.664133e+07
191	2020-07-31	1.695910e+07	1.706681e+07	1.684956e+07
192	2020-08-01	1.716676e+07	1.727705e+07	1.705751e+07
193	2020-08-02	1.736430e+07	1.748551e+07	1.725391e+07
194	2020-08-03	1.755889e+07	1.767870e+07	1.744431e+07

195 rows × 4 columns

- ds: This column contains the dates for which the predictions have been made. It stands for “date stamp”.
- yhat: This is the predicted value (forecast) for the given date in the ds column. It represents the model's best estimate for that point in time.
- yhat_upper: This column represents the upper bound of the predicted value, which indicates the higher end of the confidence interval. It suggests that the actual value is unlikely to exceed this amount, giving you an idea of the prediction's uncertainty.
- yhat_lower: This column represents the lower bound of the predicted value, indicating the lower end of the confidence interval. It suggests that the actual value is unlikely to be lower than this amount.

Visualisation:

```
# Visualizing
m.plot(forecast)
plt.show()
```



The graph illustrates a time-series analysis, showcasing data trends between February 2020 and August 2020.

- The x-axis represents the time frame, spanning from February 2020 to August 2020.
- The y-axis, labelled "y," displays numerical values ranging from 0 to 17.5 million. This represents the quantity of confirmed cases, recoveries, or other relevant metrics.
- The blue line smoothly fits the black data points, representing a model's prediction or summarization of the trend.
- The curve depicts exponential growth, with the values rising sharply over time. This suggests that the variable being analysed (confirmed cases) is increasing at an accelerating rate.
- The graph serves as a powerful visual tool for analyzing and predicting the progression of the data metric.

Inferences from the Project

The project, focusing on Covid-19 data analysis and forecasting using the Prophet library, can provide several inferences. These inferences could be categorized into the following:

1. Data Trends and Patterns:

- Exponential Growth: Analysis of confirmed cases might reveal exponential growth trends over time, highlighting critical phases of the pandemic.
- Peak Points: Identification of peak periods for cases, recoveries, or fatalities.
- Seasonality: If data shows seasonal trends, it may indicate specific periods of higher transmission rates.

2. Country-Specific Insights:

- Impact Comparison: By analyzing top countries with the most confirmed, recovered, or death cases, the project might infer how different regions have been impacted.
- Recovery Rates: Insights into which countries exhibited higher recovery rates and possible contributing factors like healthcare infrastructure or policies.
- Trend Differences: Different countries may show varying trends due to diverse interventions or reporting practices.

3. Predictive Insights:

- Forecasting Future Cases: The Prophet-based model can predict future confirmed, active, and recovery cases, aiding in planning resource allocation.
- Trend Detection: Detection of points where trends change significantly, such as after implementing strict policies or vaccinations.

4. Healthcare System Insights:

- Burden on Healthcare: Active case trends can indicate periods of stress on healthcare systems and preparedness levels.
- Long-Term Recovery: Recovery trends may show how long it takes for a region to stabilize after a surge.

5. Limitations and Challenges:

- Reporting Bias: The project might infer inconsistencies in reporting among different countries, affecting the validity of conclusions.
- Unexpected Events: Insights may indicate that predictions are affected by unplanned events like new variants or interventions.

6. Policy and Strategic Implications:

- Effectiveness of Interventions: Policy changes' impacts, such as lockdowns or vaccinations, might be inferred from trend shifts.
- Preparedness for Future Pandemics: The analysis provides insights into how trends evolve and the importance of timely actions.

Future Possibilities

The future possibilities of a Covid-19 data analysis and forecasting project are vast and can pave the way for meaningful applications and research.

1. Long-Term Predictions: Expanding forecasts beyond a short timeframe to provide insights into pandemic trends years into the future.
2. Variant Analysis: Including data on Covid-19 variants to predict their spread and impact more accurately.
3. Worldwide Comparisons: Analyzing and forecasting data for more countries to identify global trends and how different regions manage the pandemic.
4. Healthcare Systems Insights: Providing data-driven insights into healthcare capacity and system preparedness in different areas.
5. Live Data Integration: Updating the model in real time to reflect the latest Covid-19 data, enabling dynamic predictions and immediate decision-making.
6. Early Warning Systems: Detecting potential surges in cases or new outbreaks through rapid analysis.
7. Policy Impact Assessment: Simulating how various interventions (e.g., lockdowns,

vaccinations) might alter the trajectory of cases, recoveries, and deaths.

8. Resource Allocation: Using forecasts to optimize allocation of medical resources, staffing, and infrastructure in response to predicted trends.
9. Cross-Pandemic Applications: Adapting the techniques developed in this project to model other diseases, seasonal flu outbreaks, or future pandemics.
10. Public Awareness: Informing citizens through user-friendly dashboards or infographics on Covid-19 trends and predictions.

Conclusion

In conclusion, this project leverages data analysis and time-series forecasting with tools like the Prophet library to understand and predict Covid-19 trends. By exploring historical patterns of confirmed cases, recoveries, and deaths, it provides critical insights into the progression of the pandemic. The project aids in identifying key trends, peak periods, and the effectiveness of interventions while forecasting future scenarios. Such insights have applications in healthcare preparedness, resource allocation, and public health policy. This study underscores the importance of data-driven decision-making in addressing global health crises and lays a foundation for further research and applications.