**COVID 19 CASE ANALYSIS USING COGNOS**

**BATCH MEMBER**

**611721205021 : RATHIKASRI.K**

**Phase 4 submission document**

**Project Title: Covid 19 case Analysis**

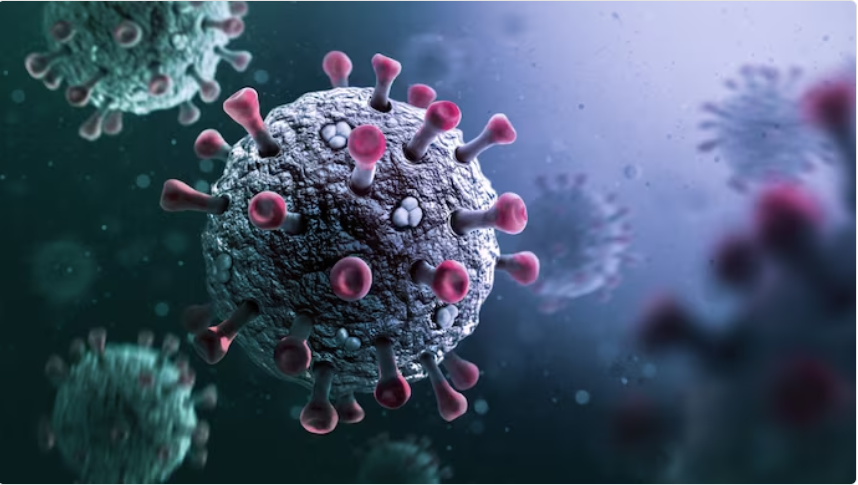
**Phase 3: Development part 2**

**Topic:**

**In this part I will continue building**

**project.**

* **Continue building the analysis by creating visualizations using IBM Cognos and deriving insights from the data.**
* **Create charts and graphs in IBM Cognos to visualize and compare the mean values and standard deviations of COVID-19 cases and associated deaths.**
* **Analyze the visualizations to identify trends, variations, and potential correlations between cases and deaths.**

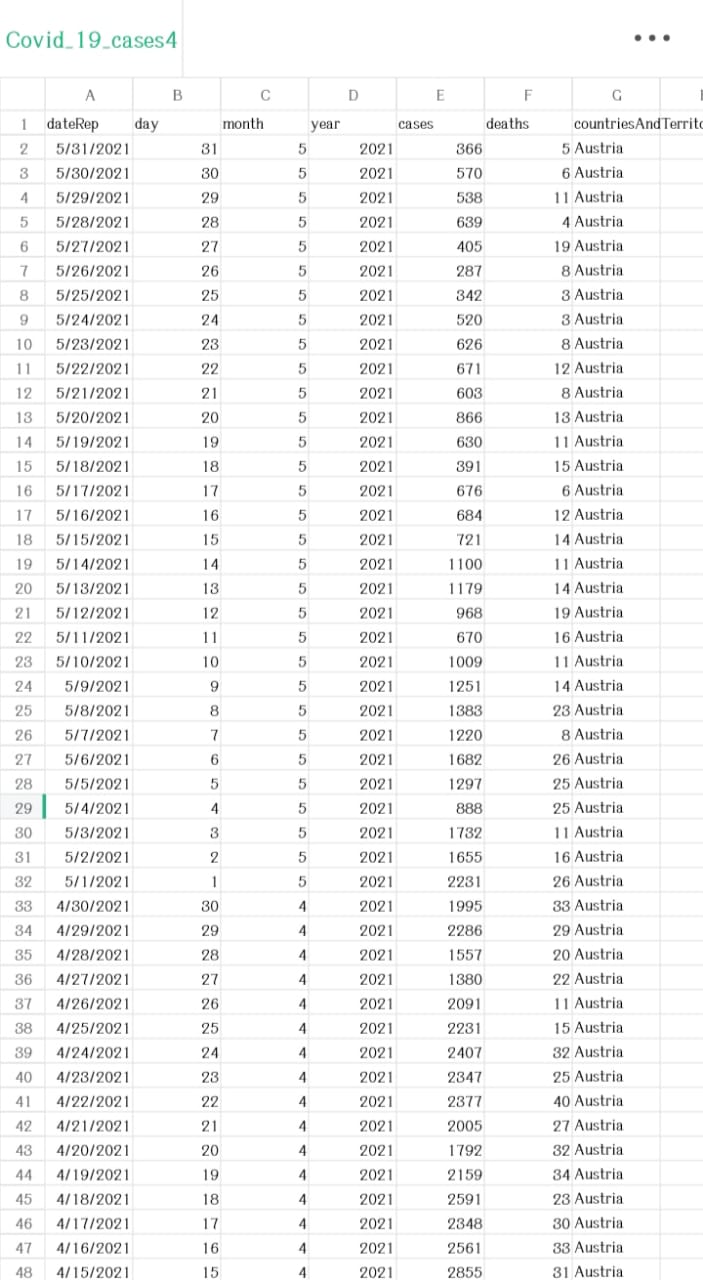
****

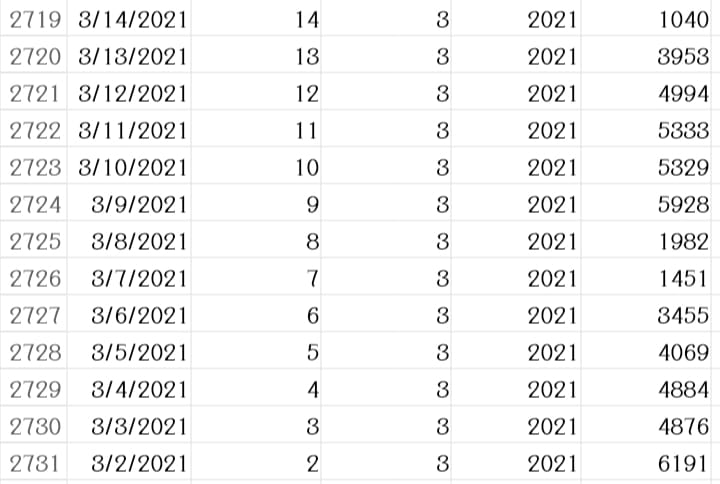
**COVID 19 CASE ANALYSIS**

**INTRODUCTION:**

In the face of the unprecedented global health crisis brought about by the COVID-19 pandemic, understanding and analyzing the data related to the spread and impact of the virus have become paramount. The abundance of data available, ranging from infection rates to recovery statistics, presents an invaluable opportunity for researchers and analysts to derive meaningfulinsights. This project, titled "COVID-19 Case Analysis," delves into the intricacies of the pandemic, employing advanced data analysis techniques to unravel patterns and trends.

**GIVEN DATASET:**





2731 rows x 5 columns

**NECESSARY STEPS TO FOLLOW:**

**1)Import Libraries:**

Begin by importing the necessary libraries. In most cases,need Pandas, a powerful data manipulation library in Python.

**CODE:**

import pandas as pd

**2) Choose the Data Source:**

Determine the source of your dataset. It can be a CSV file, Excel file, SQL database, or any other structured data format.

**3)Load the Dataset:**

Use the appropriate Pandas function to load the dataset into a Pandas DataFrame. For example, if your data is in a CSV file:

# Assuming your file is named "data.csv"

df = pd.read\_csv("data.csv")

For Excel files:

# Assuming your file is named "data.xlsx" and the sheet name is "Sheet1"

df = pd.read\_excel("data.xlsx", sheet\_name="Sheet1")

For SQL databases:

# Assuming you have a SQL connection and your query is stored in the variable "sql\_query"

df = pd.read\_sql\_query(sql\_query, sql\_connection)

**4)Explore the Loaded Data:**

After loading, it's crucial to explore the data briefly to understand its structure and ensure it was loaded correctly. Use functions like head(), info(), and describe()

**CODE:**

print(df.head()) # Displays the first few rows of the DataFrame

print(df.info()) # Provides information about the DataFrame, including data types and null values

print(df.describe()) # Generates summary statistics of the numerical columns.

**5)Handle Missing Data:**

Check for missing data and handle it appropriately. You can remove rows with missing values or fill them with suitable values using dropna() or fillna() functions.

**CODE:**

df.dropna(inplace=True) # Drops rows with missing values

# OR

df.fillna(value, inplace=True) # Fills missing values with a specific value

**6)Data Cleaning and Transformation:**

Depending on your analysis goals, you might need to clean the data by removing outliers, standardizing formats, or transforming variables.

**CODE:**

# Example: Converting a column to datetime format

df['date\_column'] = pd.to\_

datetime(df['date\_column'])

**7)Save Processed Data:**

If you make significant changes to the dataset, consider saving the processed data for future use.

**CODE:**

df.to\_csv("processed\_data.csv", index=False) # Saves the DataFrame to a CSV file without index column

**PREPROCESSING THE DATASET:**

Preprocessing a dataset is a crucial step in data analysis and machine learning. It involves cleaning, transforming, and organizing the data to make it suitable for analysis or modeling. Here are the essential steps for preprocessing a dataset.

**1)Handling Missing Data:**

Identify and handle missing data points. Options include removing rows with missing values or filling missing values with the mean, median, or a specific value.

**CODE:**

df.dropna(inplace=True) # Drops rows with missing values

# OR

df.fillna(value, inplace=True) # Fills missing values with a specific value

**2)Handling Categorical Data:**

Convert categorical variables into numerical representations. This can be done using techniques like one-hot encoding, where each category becomes a binary column.

**CODE:**

df = pd.get\_dummies(df, columns=['categorical\_column']) # One-hot encoding

**3)Data Standardization:**

Standardize or normalize features if they are on different scales. Standardization ensures that the features have the same scale, which is important for many machine learning algorithms.

**CODE:**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['feature1', 'feature2']] = scaler.fit\_transform(df[['feature1', 'feature2']])

**4)Data Transformation:**

Transform variables if necessary. Common transformations include log transformations for skewed data to make it more normally distributed.

**CODE:**

import numpy as np

df['transformed\_column'] = np.log(df['original\_column'])

**5)Feature Engineering:**

Create new features from existing ones if they can provide valuable information to the analysis or model.

**CODE:**

df['new\_feature'] = df['feature1'] \* df['feature2']

**6)Outlier Detection and Removal:**

Identify and handle outliers if they exist in the dataset. Outliers can significantly impact the results of data analysis and machine learning models.

**CODE:**

from scipy import stats

z\_scores = np.abs(stats.zscore(df['numeric\_column']))

df\_no\_outliers = df[(z\_scores < 3)] # Keeps only the data points within 3 standard deviations

**7)Data Splitting (for Machine Learning):**

If preparing the dataset for machine learning, split it into features (X) and target variable (y). This is necessary for supervised learning tasks.

**CODE:**

X = df.drop('target\_column', axis=1) # Features

y = df['target\_column'] # Target variable

**8) Feature Scaling (for Machine Learning):**

Scale the features to ensure that all features contribute equally to the analysis or model. Common techniques include Min-Max scaling or Standardization.

**CODE:**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

**IMPORTANCE OF LOADING AND PREPROCESSING THE DATASET:**

**1. Data Quality Assurance:**

Loading a dataset allows you to assess its quality. By inspecting the loaded data, you can identify missing values, inconsistencies, and errors. Preprocessing steps such as handling missing data and outliers enhance the quality and reliability of the dataset.

**2. Data Understanding:**

Loading the dataset provides an initial understanding of its structure, variables, and format. This understanding is essential for making informed decisions during preprocessing. It helps you identify the types of transformations and cleaning operations required for meaningful analysis.

**3. Ensuring Compatibility:**

Preprocessing ensures that the data is in a format compatible with the analysis or machine learning algorithms you plan to use. Different algorithms have different requirements; preprocessing helps in transforming data to meet these requirements, improving the performance and accuracy of the models.

**4. Enhancing Model Performance:**

For machine learning tasks, preprocessing steps like feature scaling and handling categorical data are vital. Scaling ensures that features are on a similar scale, preventing certain features from dominating the model. Handling categorical data appropriately ensures that machine learning algorithms can interpret these variables correctly, leading to more accurate predictions.

**5. Handling Missing Data:**

Real-world datasets often have missing values. Preprocessing techniques such as imputation (filling missing values) or removing incomplete records ensure that analyses and models are based on complete information, preventing biased or skewed results.

**6. Improving Interpretability:**

Well-preprocessed data is easier to interpret. Clear, consistent, and transformed data allows for better visualization and understanding of patterns and trends, leading to more meaningful insights.

**7. Reducing Computational Costs:**

Preprocessing, including techniques like dimensionality reduction, can significantly reduce the computational resources required for analysis and modeling. This is particularly important when dealing with large datasets, making the process more efficient.

**8. Facilitating Collaboration:**

When datasets are properly loaded and preprocessed, they become easier to share and collaborate on. Clean, well-documented data ensures that other team members or researchers can understand and work with the data effectively.

**CHALLENGES INVOLVED IN LOADING AND PREPROCESSING DATASET:**

Loading and preprocessing datasets, especially large and real-world datasets, come with several challenges. Here are some common challenges faced in these processes:

**1)Data Inconsistency:**

Datasets often come from multiple sources, leading to inconsistencies in data formats, units, and naming conventions. Handling these inconsistencies is challenging, as standardizing the data is crucial for meaningful analysis.

**2)Missing Data:**

Real-world data frequently contains missing values, which need to be appropriately handled. Deciding whether to remove incomplete records or fill in missing values without introducing bias is a challenge in preprocessing.

**3)Noisy Data:**

Noisy data includes irrelevant or erroneous information. Identifying and filtering out noise is essential, as it can significantly impact the quality of analysis and modeling.

**4)Large Volume of Data:**

Big data challenges include processing and managing vast volumes of data efficiently. Loading and preprocessing large datasets require powerful computing resources and specialized algorithms to handle the volume.

**5)Data Security and Privacy:**

Ensuring data privacy and security is crucial, especially when working with sensitive data. Compliance with regulations such as GDPR adds complexity to handling and preprocessing datasets, requiring careful handling of personally identifiable information (PII).

**6)Handling Categorical Data:**

Categorical variables need to be transformed into numerical representations for analysis. Deciding on the appropriate encoding technique, especially for variables with many categories, can be challenging.

**7)Feature Engineering:**

Creating new meaningful features from existing ones requires domain knowledge. Deciding which features to engineer and how to combine them effectively can be a challenge.

**8)Scalability:**

Scalability challenges arise when processing datasets that grow over time. Preprocessing methods need to be scalable to handle the increasing volume of data efficiently.

**9)Time Complexity:**

Real-time data processing requires algorithms and techniques that can handle data streams efficiently. Traditional preprocessing methods might not be suitable for real-time applications.

**10)Data Imbalance:**

In classification tasks, datasets may be imbalanced, where some classes have significantly fewer instances than others. Handling this imbalance during preprocessing is crucial to prevent biases in the analysis or modeling results.

**11)Versioning and Reproducibility:**

Ensuring version control of the preprocessing steps is vital for reproducibility. Changes in preprocessing techniques or parameters can significantly impact the results, making it essential to document and version preprocessing procedures.

**LOADING THE DATASET:**

**import pandas as pd**

**# Replace 'your\_dataset.csv' with the actual file path of your CSV dataset**

**file\_path = 'https://www.kaggle.com/datasets/chakradharmattapalli/covid-19-cases'**

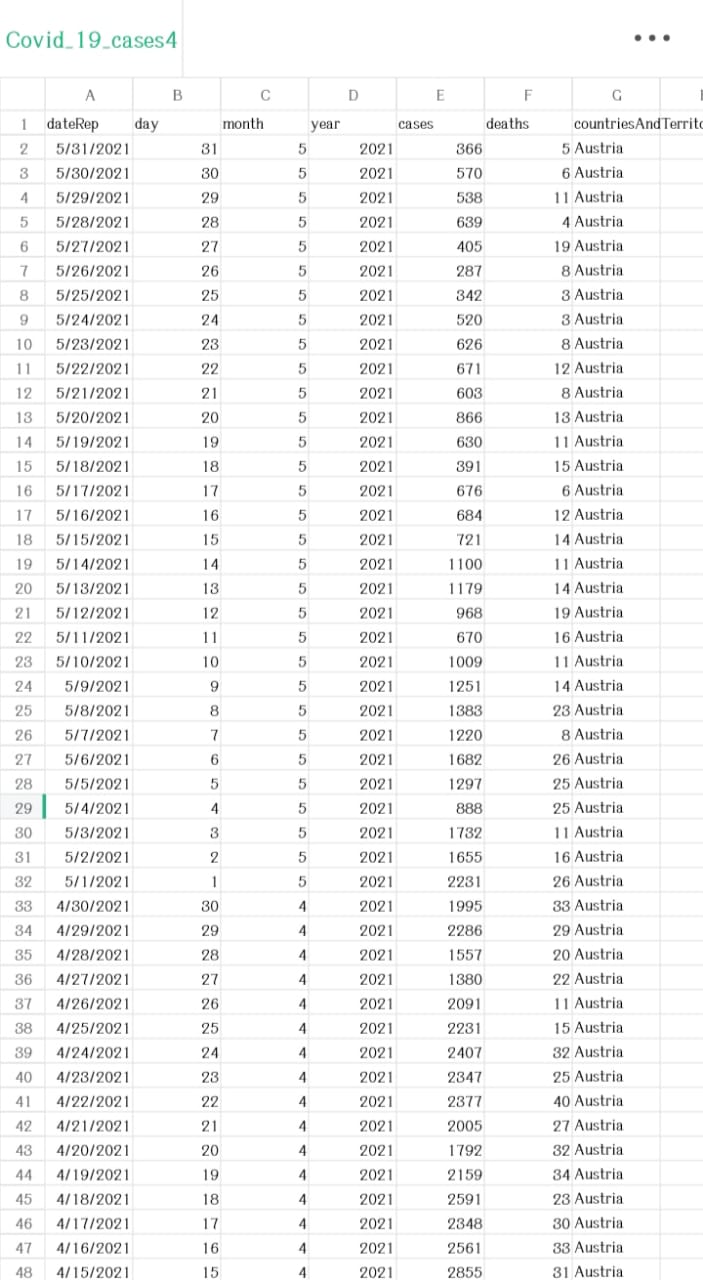
**# Load the dataset into a Pandas DataFrame**

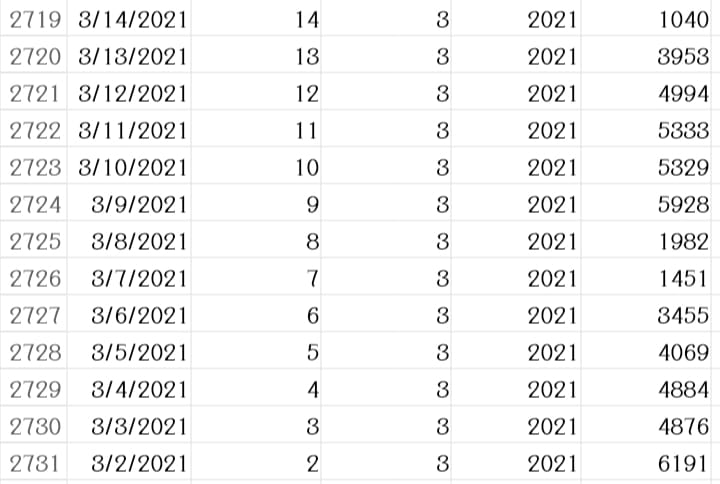
**df = pd.read\_csv(https://www.kaggle.com/datasets/chakradharmattapalli/covid-19-cases)**

**# Now 'df' contains your dataset and you can perform various operations on it**

**print(df.head()) # Display the first few rows of the loaded dataset**

**OUTPUT:**

****

****

**PREPROCESSING THE DATASET:**

* Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.
* This may involve removing error and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable rows and columns.
* #Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load your COVID-19 dataset into a Pandas DataFrame

data = pd.read\_csv(‘https://www.kaggle.com/datasets/chakradharmattapalli/covid-19-cases')

# Handle missing values (if any)

data = data.dropna()

# Encode categorical variables (if any)

label\_encoders = {}

categorical\_columns = ['column1', 'column2'] # Specify the categorical columns in your dataset

for column in categorical\_columns:

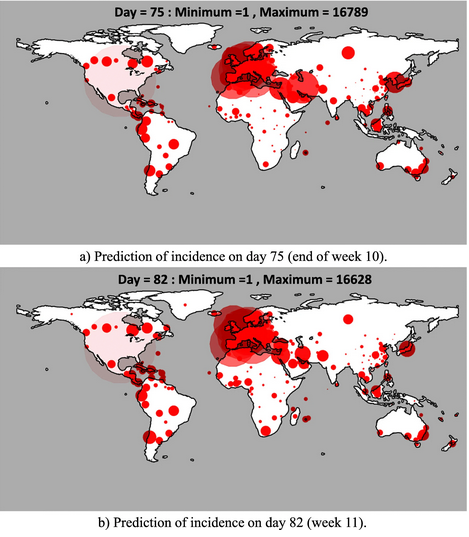
label\_encoders[column] = LabelEncoder()

data[column] = label\_encoders[column].fit\_transform(data[column])

# Split the data into features (X) and target variable (y)

X = data.drop(columns=['target\_column']) # Replace 'target\_column' with the name of your target variable

y = data['target\_column']

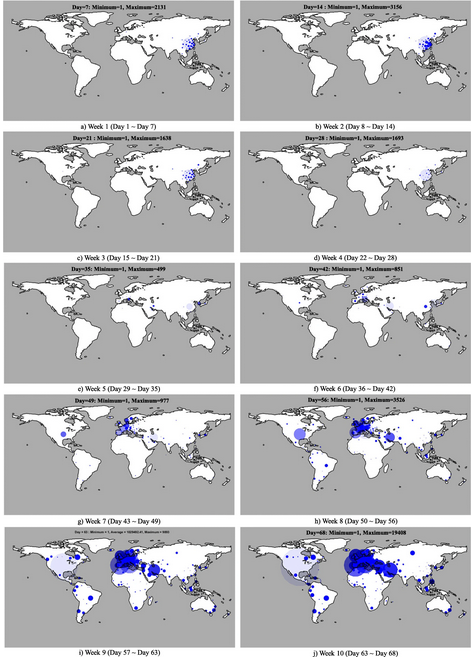


# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

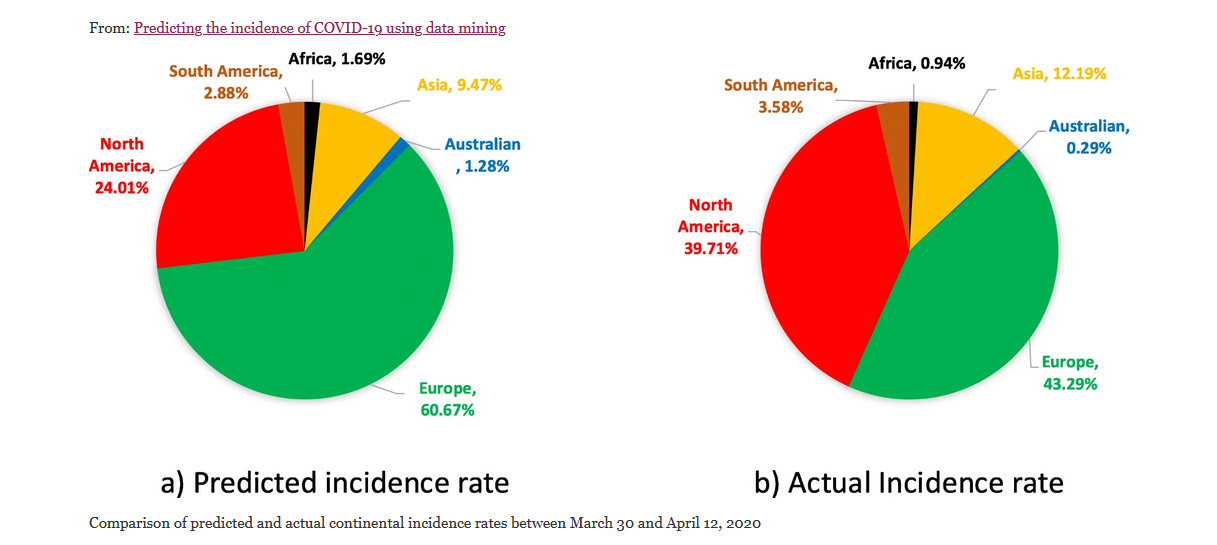
# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()



X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)



import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load your preprocessed COVID-19 dataset into a Pandas DataFrame

# (Assuming you already have a preprocessed DataFrame named 'data')

# data = pd.read\_csv(‘https://www.kaggle.com/datasets/chakradharmattapalli/covid-19-cases’)

sns.lineplot(x='Date', y='Cases', data=data)

plt.xlabel('Date')

plt.ylabel('Number of Cases')

plt.title('Daily COVID-19 Cases Over Time')

plt.xtic')

Bar plot - COVID-19 cases by country

plt.figure(figsize=(12, 6))

sns.barplot(x='Country', y='Cases', data=data)

plt.xticks(rotation=90)

plt.xlabel('Country')

plt.ylabel('Number of Cases')

plt.title('COVID-19 Cases by Country')

plt.show()

Line plot - Daily cases over time

plt.figure(figsize=(12, 6))

sns.lineplot(x='Date', y='Cases', data=data)

plt.xlabel('Date')

plt.ylabel('Number of Cases')

plt.title('Daily COVID-19 Cases Over Time')

plt.xticks(rotation=45)

plt.show()

Box plot - Distribution of cases by continent

plt.figure(figsize=(10, 6))

sns.boxplot(x='Continent', y='Cases', data=data)

plt.xlabel('Continent')

plt.ylabel('Number of Cases')

plt.title('Distribution of COVID-19 Cases by Continent')

plt.show()

Heatmap - Correlation between numerical variables

correlation\_matrix = data.corr()

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

Pairplot - Relationship between multiple variables

sns.pairplot(data[['Cases', 'Deaths', 'Recovered', 'Population']])

plt.suptitle('Pairplot of COVID-19 Data', y=1.02)

plt.show()

**VISUALIZATIONS:**

PLOTTING SIMPLE PLOT:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv('case\_time\_series.csv')

Y = data.iloc[61:,1].values

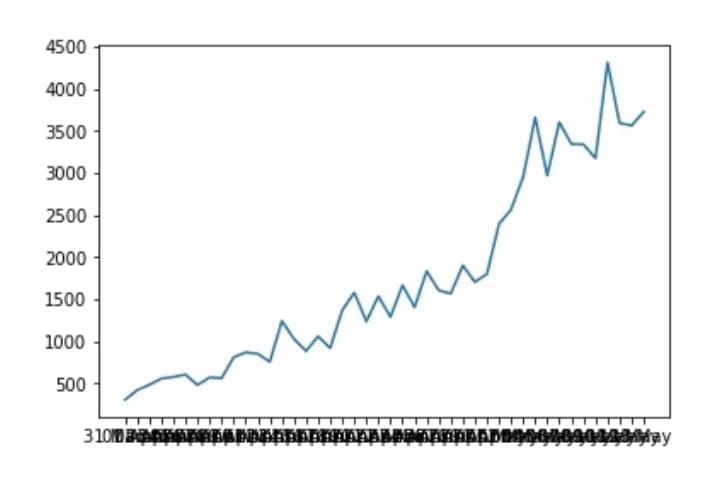
R = data.iloc[61:,3].values

D = data.iloc[61:,5].values

X = data.iloc[61:,0]

plt.plot(X,Y)

**OUTPUT:**

****

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**data = pd.read\_csv('case\_time\_series.csv')**

**Y = data.iloc[61:,1].values**

**R = data.iloc[61:,3].values**

**D = data.iloc[61:,5].values**

**X = data.iloc[61:,0]**

**plt.figure(figsize=(25,8))**

**ax = plt.axes()**

**ax.grid(linewidth=0.4, color='#8f8f8f')**

**ax.set\_facecolor("black")**

**ax.set\_xlabel('\nDate',size=25,color='#4bb4f2')**

**ax.set\_ylabel('Number of Confirmed Cases\n',**

**size=25,color='#4bb4f2')**

**ax.plot(X,Y,**

**color='#1F77B4',**

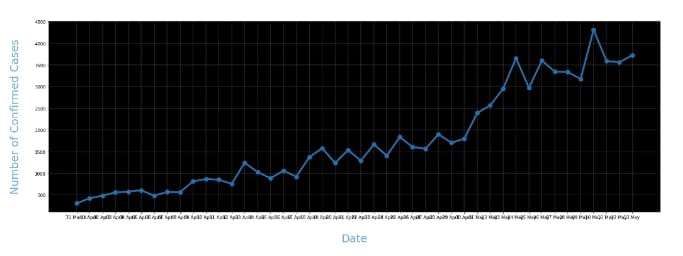
**marker='o',**

**linewidth=4,**

**markersize=15,**

**markeredgecolor='#035E9B**

**OUTPUT:**

****

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**data = pd.read\_csv('case\_time\_series.csv')**

**Y = data.iloc[61:,1].values**

**R = data.iloc[61:,3].values**

**D = data.iloc[61:,5].values**

**X = data.iloc[61:,0]**

**plt.figure(figsize=(25,8))**

**ax = plt.axes()**

**ax.grid(linewidth=0.4, color='#8f8f8f')**

**ax.set\_facecolor("black")**

**ax.set\_xlabel('\nDate',size=25,color='#4bb4f2')**

**ax.set\_ylabel('Number of Confirmed Cases\n',**

**size=25,color='#4bb4f2')**

**plt.xticks(rotation='vertical',size='20',color='white')**

**plt.yticks(size=20,color='white')**

**plt.tick\_params(size=20,color='white')**

**for i,j in zip(X,Y):**

**ax.annotate(str(j),xy=(i,j+100),color='white',size='13')**

**ax.annotate('Second Lockdown 15th April',**

**xy=(15.2, 860),**

**xytext=(19.9,500),**

**color='white',**

**size='25',**

**arrowprops=dict(color='white',**

**linewidth=0.025))**

**plt.title("COVID-19 IN : Daily Confirmed\n",**

**size=50,color='#28a9ff')**

**ax.plot(X,Y,**

**color='#1F77B4',**

**marker='o',**

**linewidth=4,**

**markersize=15,**

**markeredgecolor='#035E9B**

**OUTPUT:**

****

**data = pd.read\_csv('district.csv')**

**data.head()**

**re=data.iloc[:30,5].values**

**de=data.iloc[:30,4].values**

**co=data.iloc[:30,3].values**

**x=list(data.iloc[:30,0])**

**plt.figure(figsize=(25,10))**

**ax=plt.axes()**

**ax.set\_facecolor('black')**

**ax.grid(linewidth=0.4, color='#8f8f8f')**

**plt.xticks(rotation='vertical',**

**size='20',**

**color='white')#ticks of X**

**plt.yticks(size='20',color='white')**

**ax.set\_xlabel('\nDistrict',size=25,**

**color='#4bb4f2')**

**ax.set\_ylabel('No. of cases\n',size=25,**

**color='#4bb4f2')**

**plt.tick\_params(size=20,color='white')**

**ax.set\_title('Maharashtra District wise breakdown\n',**

**size=50,color='#28a9ff')**

**plt.bar(x,co,label='re')**

**plt.bar(x,re,label='re',color='green')**

**plt.bar(x,de,label='re',color='red')**

**for i,j in zip(x,co):**

**ax.annotate(str(int(j)),**

**xy=(i,j+3),**

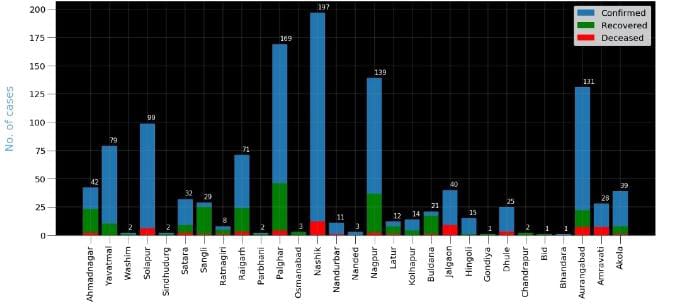
**color='white',**

**size='15')**

**plt.legend(['Confirmed','Recovered','Deceased'],**

**fontsize=20)**

**OUTPUT:**

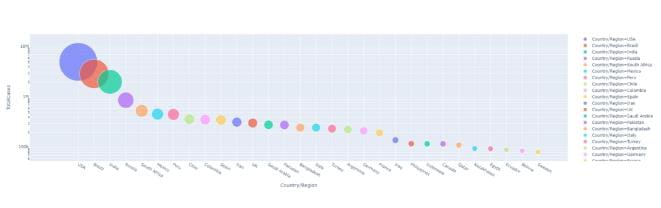
****

**px.scatter(dataset1.head(30), x='Country/Region', y='TotalCases',**

**hover\_data=['Country/Region', 'Continent'],**

**color='Country/Region**

**OUTPUT:**

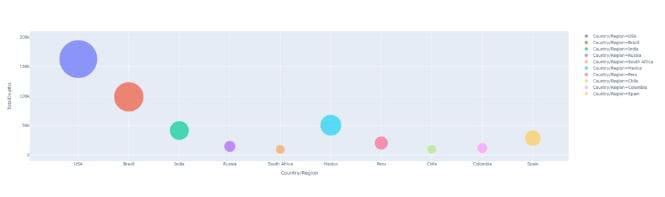
****

**px.scatter(dataset1.head(10), x='Country/Region', y= 'TotalDeaths',**

**hover\_data=['Country/Region', 'Continent'],**

**color='Country/Region', size= 'TotalDeaths', size\_max=80)**

**OUTPUT:**

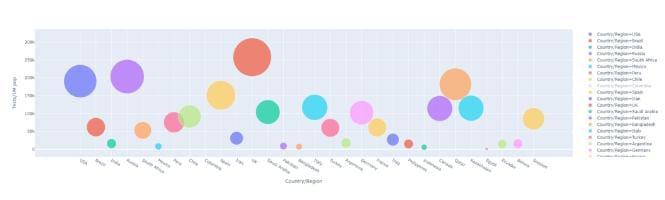
****

**px.scatter(dataset1.head(30), x='Country/Region', y= 'Tests/1M pop',**

**hover\_data=['Country/Region', 'Continent'],**

**color='Country/Region', size= 'Tests/1M pop', size\_max=80)**

**OUTPUT:**

****

**CONCLUSION OF PHASE 4 PROJECT:**

In the phase 4 conclusion of development part 2, we performed the covid 19 cases analysis and create visualizations, death cases , recovered, decreased, maximum causes. then we created some visualizations using data visualization libraries(e.g., matplotlib, seaborn) for **covid 19 cases analysis** by using the given dataset.