# DATA ANALYSIS USING PYTHON

## Data PreProcessing

Data preprocessing is a crucial step in data analysis and machine learning. It refers to the process of cleaning, transforming, and organizing raw data before it can be used for further analysis or modeling. The goal of data preprocessing is to ensure that the data is in a format that is suitable for analysis and to minimize errors or inaccuracies that could affect the results of the analysis.

Data preprocessing involves several steps, including:

Data cleaning: This step involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, or incorrect data types.

Data transformation: This step involves converting the data into a suitable format for analysis. This may include scaling or normalizing the data, converting categorical variables into numerical values, or reducing the dimensionality of the data.

Data integration: This step involves combining data from multiple sources into a single dataset.

Data reduction: This step involves reducing the amount of data to be analyzed by selecting relevant features or samples.

Overall, data preprocessing is an essential step in data analysis and machine learning, as it helps to ensure that the data is accurate, consistent, and suitable for analysis.

**CODE:**

```
import pandas as pd

import csv

def handle_Null_Value(df,string):

    n = len(df[string])

    for i in range(n):

        if(df[string][i]=='' or int(df[string][i])<0):

            df[string][i]=0



with open("WHO-COVID-19-global-data.csv") as myfile:

    raw_data = list(csv.reader(myfile))


newData = dict()

headers = list(raw_data[0])

for i in range(0,len(headers)):

    k=list()

    for j in range(1,len(raw_data)):

        k.append(raw_data[j][i])

    newData[headers[i]]=list(k)
```

```
df = pd.DataFrame(newData)
handle_Null_Value(df,'New_cases')
handle_Null_Value(df,'Cumulative_cases')
handle_Null_Value(df,'New_deaths')
handle_Null_Value(df,'Cumulative_deaths')
df.to_csv("Processed_Covid_file.csv",index=False)
```

## MODEL TRAINING AND TESTING

Model training and testing are two important stages in the process of developing a machine learning model. The goal of model training is to build a predictive model that can accurately predict the outcome of a given task based on a set of input features. The goal of model testing is to evaluate the performance of the trained model on a separate set of data, called the test set, to ensure that it can generalize well to new, unseen data.

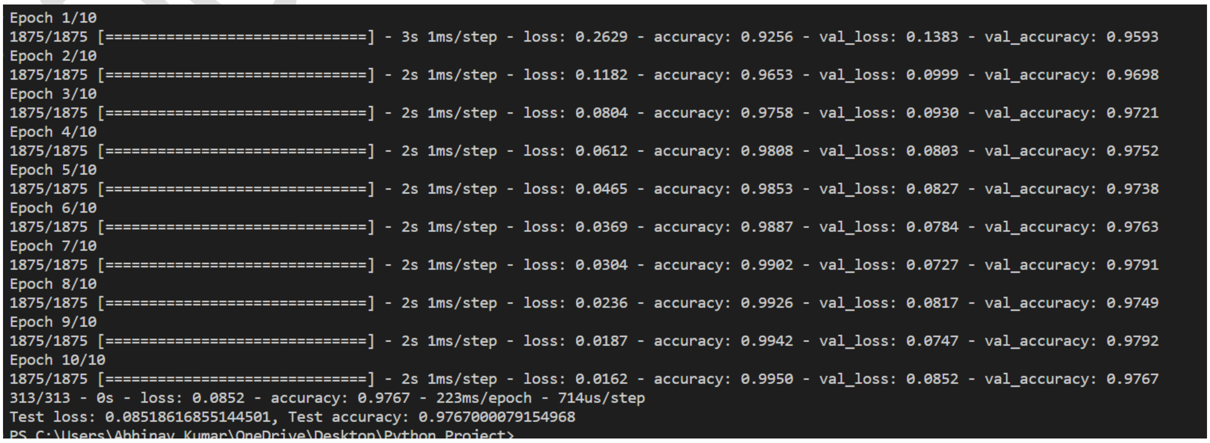
During the model training stage, the machine learning algorithm is trained using a set of labeled data, called the training set. The algorithm uses this data to learn the relationship between the input features and the output variable. The model is adjusted and refined based on the performance on the training set until it can accurately predict the outcome of the task.

Once the model is trained, it is tested using a separate set of data, called the test set. The test set is used to evaluate the performance of the model on new, unseen data. The performance of the model is evaluated based on various metrics, such as accuracy, precision, recall, or F1 score. The goal is to ensure that the model can generalize well to new data and that it is not overfitting to the training data.

Model training and testing are iterative processes, and the model may need to be adjusted and retrained multiple times before achieving satisfactory performance on the test set. Once the model has been tested and validated, it can be used to make predictions on new, unseen data.

**CODE:**

```
import tensorflow as tf
from tensorflow.keras import layers

# Load data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Preprocess data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Define model architecture
model = tf.keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Set up optimizer and loss function
optimizer = tf.keras.optimizers.Adam()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()

# Compile the model
model.compile(optimizer=optimizer, loss=loss_fn, metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Test loss: {test_loss}, Test accuracy: {test_acc}')
```



```
Epoch 1/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2629 - accuracy: 0.9256 - val_loss: 0.1383 - val_accuracy: 0.9593
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1182 - accuracy: 0.9653 - val_loss: 0.0999 - val_accuracy: 0.9698
Epoch 3/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0804 - accuracy: 0.9758 - val_loss: 0.0930 - val_accuracy: 0.9721
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0612 - accuracy: 0.9808 - val_loss: 0.0803 - val_accuracy: 0.9752
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0465 - accuracy: 0.9853 - val_loss: 0.0827 - val_accuracy: 0.9738
Epoch 6/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0369 - accuracy: 0.9887 - val_loss: 0.0784 - val_accuracy: 0.9763
Epoch 7/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0304 - accuracy: 0.9902 - val_loss: 0.0727 - val_accuracy: 0.9791
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0236 - accuracy: 0.9926 - val_loss: 0.0817 - val_accuracy: 0.9749
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0187 - accuracy: 0.9942 - val_loss: 0.0747 - val_accuracy: 0.9792
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0162 - accuracy: 0.9950 - val_loss: 0.0852 - val_accuracy: 0.9767
313/313 - 0s - loss: 0.0852 - accuracy: 0.9767 - 223ms/epoch - 714us/step
Test loss: 0.08518616855144501, Test accuracy: 0.9767000079154968
PS C:\Users\Abhinav Kumar\OneDrive\Desktop\Python Project>
```

# ANALYSIS FUNCTION

```
import pandas as pd


all_cases=0
all_deaths=0
maxi_cases=0
maxi_deaths=0
maxi_cases_date=""
maxi_deaths_date=""
avg_cases=0
avg_deaths=0


def average_cases(c_code):
    main(c_code)
    return avg_cases

def average_deaths(c_code):
    main(c_code)
    return avg_deaths



def maximum_deaths(c_code):
    main(c_code)
    return (maxi_deaths,maxi_deaths_date)



def maximum_cases(c_code):
    main(c_code)
    return (maxi_cases,maxi_cases_date)



def total_cases(c_code):
    main(c_code)
    return all_cases



def total_deaths(c_code):
    main(c_code)
    return all_deaths



def main(c_code):
    global all_cases,all_deaths,maxi_cases,maxi_deaths,avg_cases,avg_deaths,maxi_cases_date,maxi_deaths_date
    with open("Processed_Covid_file.csv") as myfile:
            raw_data=pd.read_csv(myfile)
    dates=list(raw_data['Date_reported'])
    daily_cases=list(raw_data['New_cases'])
    daily_deaths=list(raw_data['New_deaths'])
    country_codes=list(raw_data['Country_code'])
    days=0

    for i in range(1,len(raw_data['Country_code'])):
        if(country_codes[i]==c_code):
            all_cases+=int(daily_cases[i])
            days+=1
            all_deaths+=int(daily_deaths[i])
            if(maxi_cases<int(daily_cases[i])):
                maxi_cases=daily_cases[i]
                maxi_cases_date=dates[i]

            if(maxi_deaths<int(daily_deaths[i])):
                maxi_deaths=daily_deaths[i]
                maxi_deaths_date=dates[i]

    avg_cases=all_cases/days
    avg_deaths=all_deaths/days
```

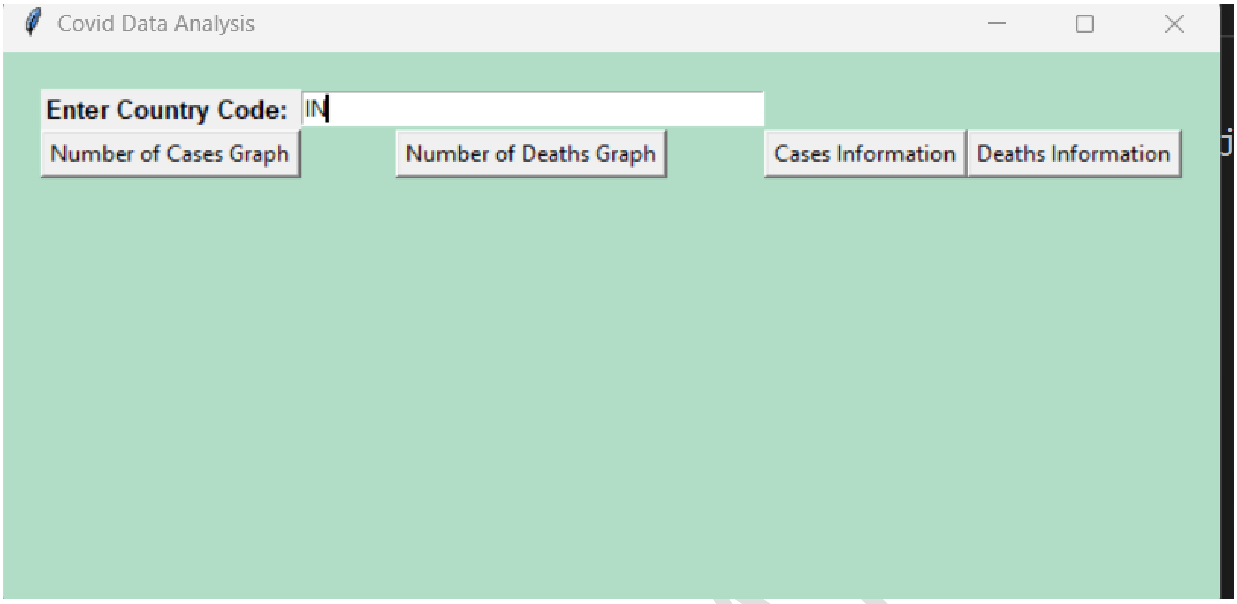
# GRAPHS

```
from matplotlib import pyplot as plt
import pandas as pd

def cases_graph(c_code):
    with open("Processed_Covid_file.csv") as myfile:
        raw_data=pd.read_csv(myfile)

    dates=list(raw_data['Date_reported'])
    daily_cases=list(raw_data['New_cases'])
    country_codes=list(raw_data['Country_code'])
    country=list(raw_data['Country'])

    input_code=c_code
    country_name=""

    x=list()
    y=list()
    for i in range(1,len(raw_data['Country_code'])):
        if(country_codes[i]==input_code):
            x.append(dates[i])
            y.append(daily_cases[i])
            country_name=country[i]
    plt.plot(x,y)

    plt.title(f"Covid Cases In {country_name}({input_code})")
    plt.ylabel('Number of Cases')
    plt.xlabel(f'Dates({dates[0]} to {dates[-1]})')
    plt.xticks([])
    plt.show()


def death_graph(c_code):
    with open("Processed_Covid_file.csv") as myfile:
        raw_data=pd.read_csv(myfile)

    dates=list(raw_data['Date_reported'])
    daily_deaths=list(raw_data['New_deaths'])
    country_codes=list(raw_data['Country_code'])
    country=list(raw_data['Country'])

    input_code=c_code
    country_name=""

    x=list()
    y=list()
    for i in range(1,len(raw_data['Country_code'])):
        if(country_codes[i]==input_code):
            x.append(dates[i])
            y.append(daily_deaths[i])
            country_name=country[i]



    plt.plot(x,y)


    plt.title(f"Covid Deaths In {country_name}({input_code})")
    plt.ylabel('Number of Deaths')
    plt.xlabel(f'Dates({dates[0]} to {dates[-1]})')
    plt.xticks([])
    plt.show()
```

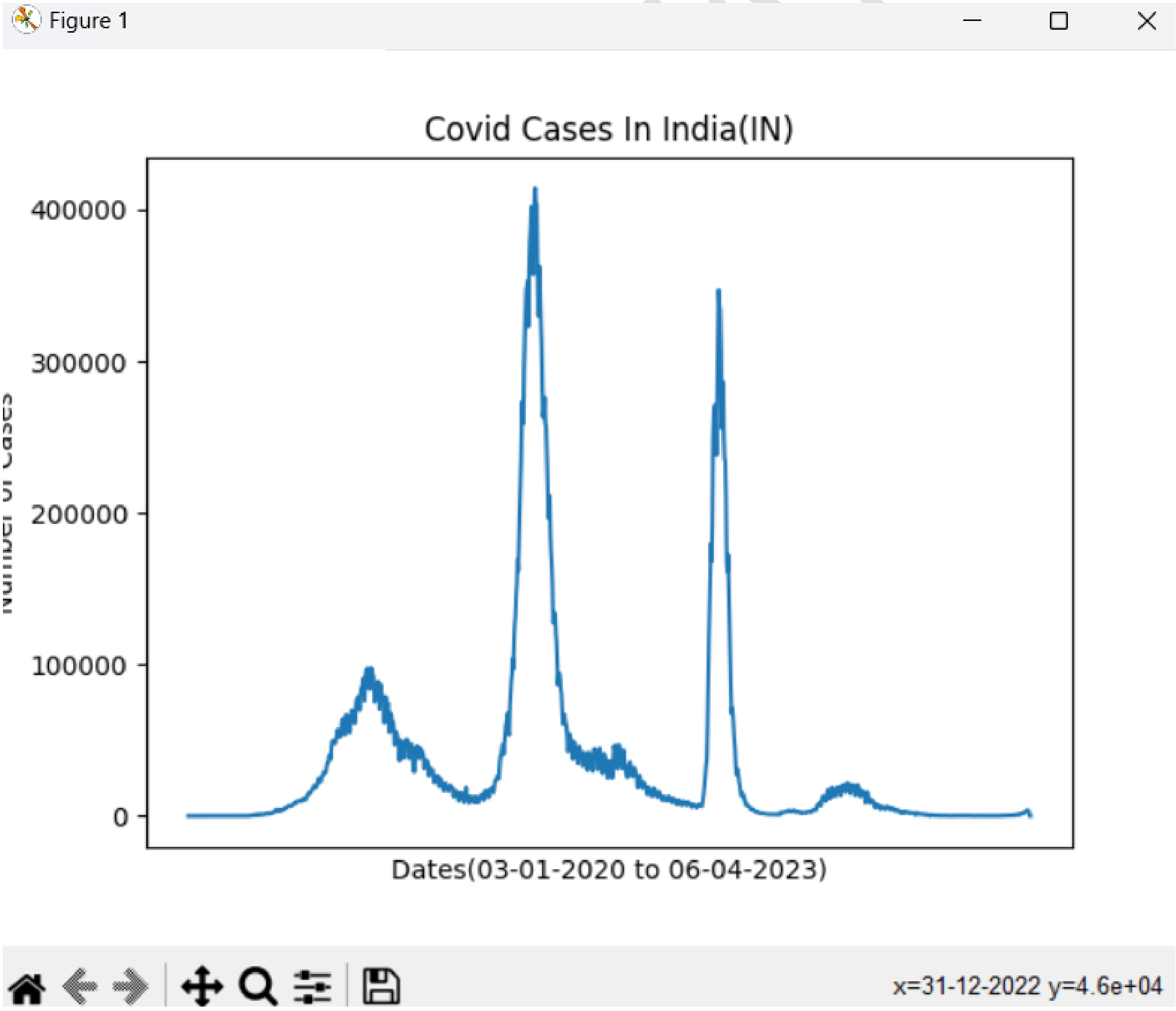
# MAIN FILE

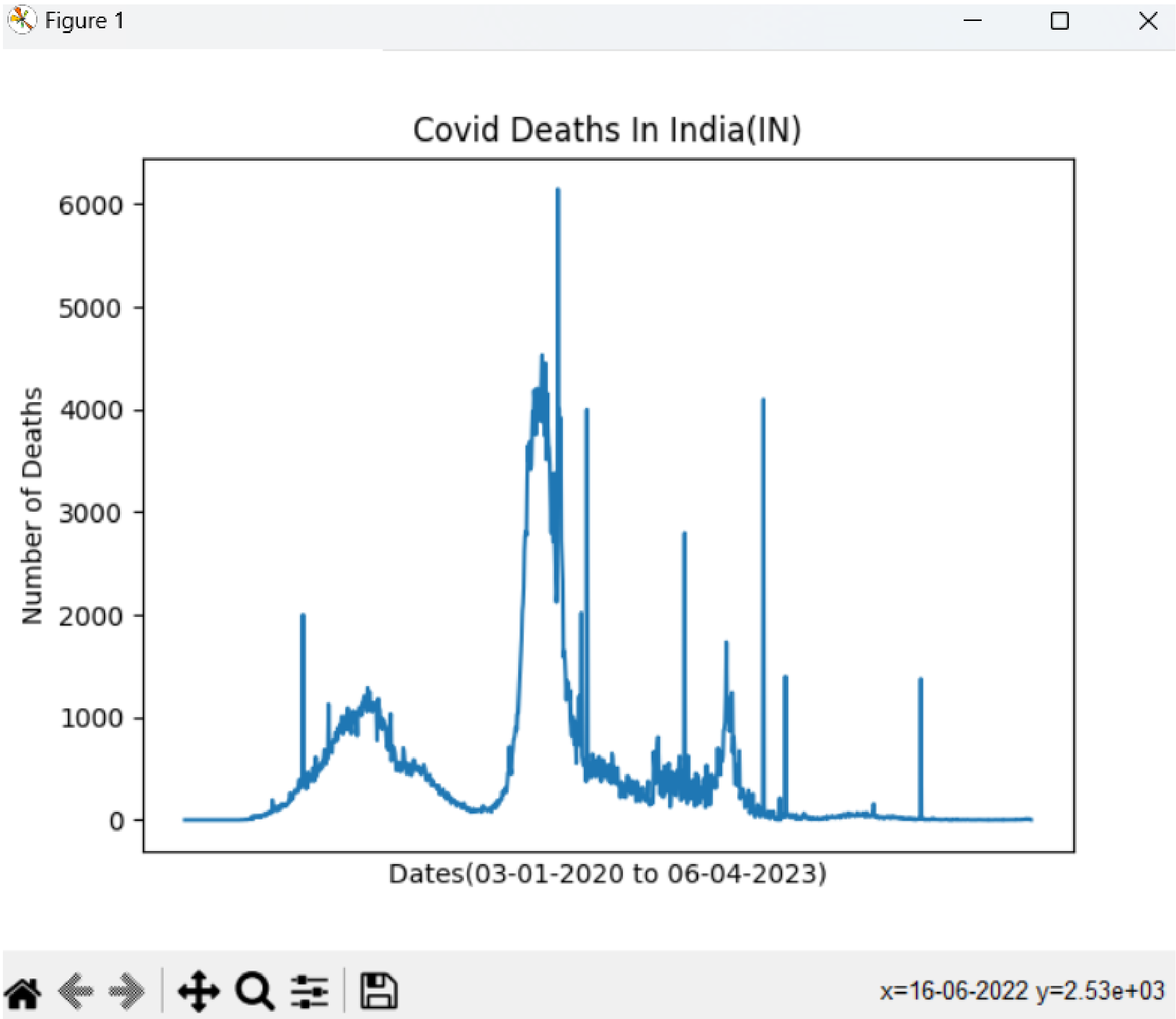
```
from tkinter import *
from covid_graphs import cases_graph
from covid_graphs import death_graph
import Analysis_functions as af

BACKGROUND_COLOR = "#B1DDC6"
def bt1_clicked():
    cases_graph(my_input.get())
    response_label=Label(text="Button Got Clicked")
    response_label.grid(column=0,row=2)


def bt2_clicked():
    death_graph(my_input.get())
    response_label=Label(text="Button Got Clicked")
    response_label.grid(column=1,row=2)


def bt3_clicked():
    avg_cases=af.average_cases(my_input.get())
    total_cases=af.total_cases(my_input.get())
    max_cases=af.maximum_cases(my_input.get())
    newLabel=Label(text=f'Average Cases : {avg_cases}\nTotal Cases : {total_cases}\nMax Cases : {max_cases[0]} On {max_cases[1]}')
    newLabel.grid(column=3,row=3)


def bt4_clicked():
    avg_deaths=af.average_deaths(my_input.get())
    total_deaths=af.total_deaths(my_input.get())
    max_deaths=af.maximum_deaths(my_input.get())
    newLabel=Label(text=f'Average Deaths : {avg_deaths}\nTotal Death : {total_deaths}\nMax Deaths : {max_deaths[0]} On {max_deaths[1]}')
    newLabel.grid(column=3,row=4)


my_window=Tk()
my_window.minsize(width=600,height=300)
my_window.title("Covid Data Analysis")
my_window.config(padx=20,pady=20)

my_label=Label(text="Enter Country Code: ",font=("Arial",10,"bold"))
my_label.grid(column=0,row=0)

my_input=Entry(width=40)
my_input.grid(column=1,row=0)



bt1=Button(text="Number of Cases Graph",command=bt1_clicked)
bt1.grid(column=0,row=1)

bt2=Button(text="Number of Deaths Graph",command=bt2_clicked)
bt2.grid(column=1,row=1)

bt3=Button(text="Cases Information",command=bt3_clicked)
bt3.grid(column=2,row=1)

bt4=Button(text="Deaths Information",command=bt4_clicked)
bt4.grid(column=3,row=1)

my_window.config(bg=BACKGROUND_COLOR)

my_window.mainloop()
```

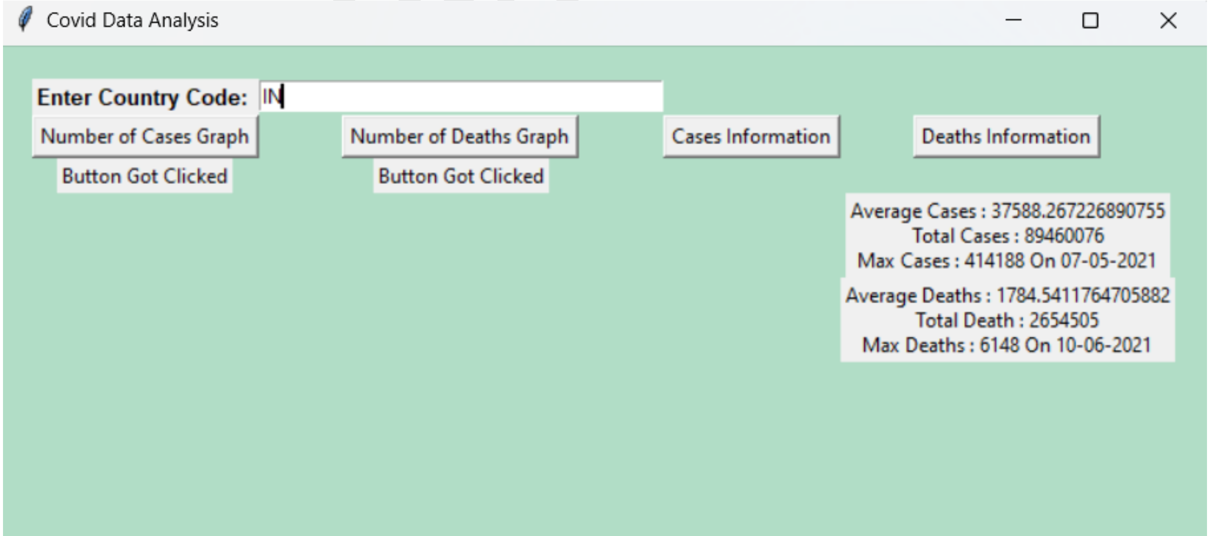
# EXAMPLE:



Enter Country Code: IN







Enter Country Code: IN

Button Got Clicked

Button Got Clicked

```
Average Cases : 37588.267226890755
Total Cases : 89460076
Max Cases : 414188 On 07-05-2021
Average Deaths : 1784.5411764705882
Total Death : 2654505
Max Deaths : 6148 On 10-06-2021
```