**Problem 4: Real-Time COVID-19 Statistics Tracker**

**4Scenario:**

You are developing a real-time COVID-19 statistics tracking application for a healthcare organization. The application should provide up-to-date information on COVID-19 cases, recoveries, and deaths for a specified region.

**Tasks:**

1. Model the data flow for fetching COVID-19 statistics from an external API and displaying it to the user.
2. Implement a Python application that integrates with a COVID-19 statistics API (e.g., disease.sh) to fetch real-time data.
3. Display the current number of cases, recoveries, and deaths for a specified region.
4. Allow users to input a region (country, state, or city) and display the corresponding COVID-19 statistics.

**Deliverables:**

* Data flow diagram illustrating the interaction between the application and the API.
* Pseudocode and implementation of the COVID-19 statistics tracking application.
* Documentation of the API integration and the methods used to fetch and display COVID-19 data.
* Explanation of any assumptions made and potential improvements.

**Approach:**

**+---------------------+**

**| User Input |**

**| (Region Query) |**

**+----------+----------+**

**|**

**v**

**+---------------------+**

**| Application Logic |**

**| (Fetch Data from API)|**

**+----------+----------+**

**|**

**v**

**+---------------------+**

**| External COVID-19 |**

**| Statistics API |**

**| (e.g., disease.sh) |**

**+----------+----------+**

**|**

**v**

**+---------------------+**

**| Display Results |**

**| (Cases, Recoveries, |**

**| Deaths for Region) |**

**+---------------------+**

**The key steps are:**

1. The user inputs a region (country, state, or city) into the application.
2. The application sends a request to the COVID-19 API to fetch the real-time statistics for the specified region.
3. The COVID-19 API processes the request and returns the current case, recovery, and death data.
4. The application receives the COVID-19 statistics and displays the information to the user.

**Pseudocode:**

Function fetch\_covid\_data(endpoint)

Try

response = Send GET request to endpoint

If response is successful

Return response in JSON format

Else

Print error message

Return None

Catch error

Print error message

Return None

Function display\_global\_covid\_data(data)

If data is not None

Extract cases, deaths, and recoveries from data

Print cases

Print deaths

Print recoveries

Else

Print "No data available."

Function display\_historical\_covid\_data(data)

If data is not None

Extract cases, deaths, and recoveries data

Convert data to DataFrame

Convert 'Date' to datetime format

Plot cases over time

Plot deaths over time

Plot recoveries over time

Display all plots

Else

Print "No data available."

Function main()

Print "1. View Global COVID-19 Statistics"

Print "2. View Historical COVID-19 Data"

choice = Get user input

If choice is '1'

url = "https://disease.sh/v3/covid-19/all"

data = Call fetch\_covid\_data(url)

Call display\_global\_covid\_data(data)

Else If choice is '2'

url = "https://disease.sh/v3/covid-19/historical/all?lastdays=all"

data = Call fetch\_covid\_data(url)

Call display\_historical\_covid\_data(data)

Else

Print "Invalid choice. Please enter 1 or 2."

Call main()

**Detailed explanation of the actual code:**

* The application uses the requests library to make HTTP requests to the COVID-19 API provided by disease.sh. The get\_covid\_stats function takes a region (country, state, or city) as input and returns the current number of cases, recoveries, and deaths for that region.
* The display\_covid\_stats function is responsible for formatting and printing the COVID-19 statistics in a user-friendly way. It takes the cases, recoveries, and deaths data as input and displays them with appropriate formatting (e.g., adding commas to large numbers).
* The main function is the entry point of the application. It prompts the user to enter a region, calls the get\_covid\_stats function to fetch the data, and then passes the results to the display\_covid\_stats function to display the information.

**Assumptions made (if any):**

* The application assumes that the disease.sh API is available and providing accurate real-time COVID-19 data.
* The application assumes that the user will input a valid region (country, state, or city) that the API can recognize.
* Potential Improvements:
* Add error handling to the application to gracefully handle API errors or invalid user input.
* Provide additional features, such as the ability to display historical COVID-19 data, trends, or visualizations.
* Integrate the application with a user interface (e.g., a web application or a mobile app) to improve the user experience.
* Allow users to select multiple regions and compare the COVID-19 statistics side-by-side.
* Provide the ability to set alerts or notifications for significant changes in COVID-19 statistics.

**Limitations:**

1. The API may have rate limits that restrict the number of requests.
2. The data may not always be up-to-date due to delays in reporting.
3. The application currently only handles countries; state and city-level queries may require additional endpoints.

**Code:**

import requests

import matplotlib.pyplot as plt

import pandas as pd

# Function to fetch COVID-19 data from an API endpoint

def fetch\_covid\_data(endpoint):

try:

response = requests.get(endpoint)

response.raise\_for\_status()

return response.json()

except requests.RequestException as e:

print(f"Error fetching data: {e}")

return None

# Function to display global COVID-19 statistics

def display\_global\_covid\_data(data):

if data:

cases = data.get('cases', 'N/A')

deaths = data.get('deaths', 'N/A')

recovered = data.get('recovered', 'N/A')

print(f"Global COVID-19 Statistics:")

print(f" Cases: {cases}")

print(f" Deaths: {deaths}")

print(f" Recoveries: {recovered}")

else:

print("No data available.")

# Function to display historical COVID-19 data with graphs

def display\_historical\_covid\_data(data):

if data:

cases = data.get('cases', {})

deaths = data.get('deaths', {})

recovered = data.get('recovered', {})

# Convert data to DataFrame for plotting

df\_cases = pd.DataFrame(list(cases.items()), columns=['Date', 'Cases'])

df\_deaths = pd.DataFrame(list(deaths.items()), columns=['Date', 'Deaths'])

df\_recovered = pd.DataFrame(list(recovered.items()), columns=['Date', 'Recovered'])

# Convert 'Date' column to datetime

df\_cases['Date'] = pd.to\_datetime(df\_cases['Date'])

df\_deaths['Date'] = pd.to\_datetime(df\_deaths['Date'])

df\_recovered['Date'] = pd.to\_datetime(df\_recovered['Date'])

# Plotting

plt.figure(figsize=(14, 7))

plt.subplot(3, 1, 1)

plt.plot(df\_cases['Date'], df\_cases['Cases'], label='Cases', color='blue')

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

plt.xlabel('Date')

plt.ylabel('Number of Cases')

plt.legend()

plt.subplot(3, 1, 2)

plt.plot(df\_deaths['Date'], df\_deaths['Deaths'], label='Deaths', color='red')

plt.title('COVID-19 Deaths Over Time')

plt.xlabel('Date')

plt.ylabel('Number of Deaths')

plt.legend()

plt.subplot(3, 1, 3)

plt.plot(df\_recovered['Date'], df\_recovered['Recovered'], label='Recovered', color='green')

plt.title('COVID-19 Recoveries Over Time')

plt.xlabel('Date')

plt.ylabel('Number of Recoveries')

plt.legend()

plt.tight\_layout()

plt.show()

else:

print("No data available.")

# Main function to interact with the user

def main():

print("1. View Global COVID-19 Statistics")

print("2. View Historical COVID-19 Data")

choice = input("Enter your choice (1 or 2): ")

if choice == '1':

url = "https://disease.sh/v3/covid-19/all"

data = fetch\_covid\_data(url)

display\_global\_covid\_data(data)

elif choice == '2':

url = "https://disease.sh/v3/covid-19/historical/all?lastdays=all"

data = fetch\_covid\_data(url)

display\_historical\_covid\_data(data)

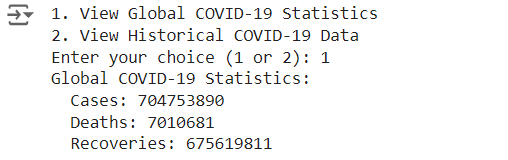
else:

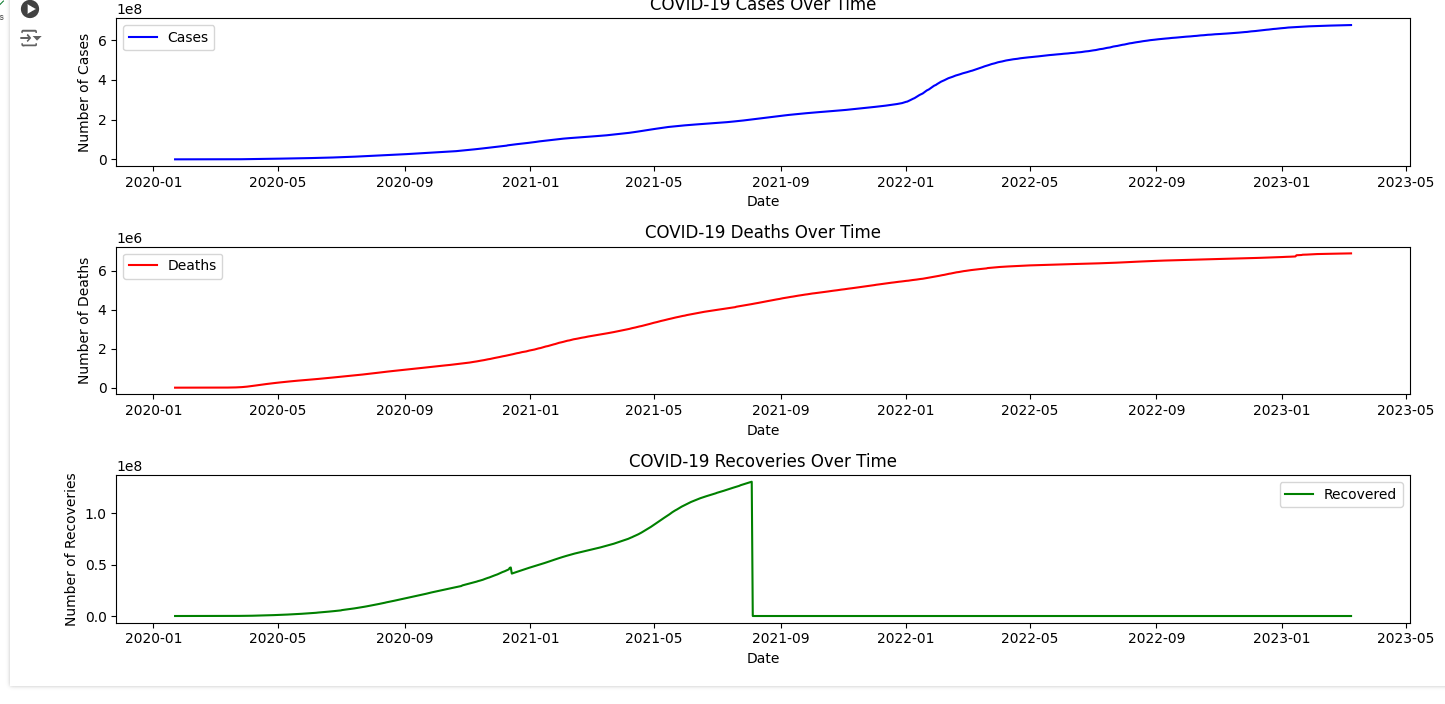
print("Invalid choice. Please enter 1 or 2.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sample Output / Screen Shots**

****

****