

Dhaka University of Engineering & Technology, Gazipur

Project Name: COVID-19 Data Dashboard

Student Name: Tanvir Ahmed

ID: 2204078

Description: Students can create a dashboard that collects and visualizes COVID-19 data using an API. The project could include visualizations of cases, recoveries, and vaccinations over time in different countries.

Skills Used: API, Pandas, Matplotlib/Seaborn, File Handling, Data Structures.

Step-1: Importing Libraries

import requests	requests: Used for making HTTP requests to fetch data from an API.
import pandas as pd	pandas: A library for data manipulation and analysis, particularly useful for handling and analyzing structured data
import tkinter as tk	tkinter: A built-in Python library for creating graphical user interfaces (GUIs).
from tkinter import messagebox	messagebox: Part of tkinter, used to show pop-up messages to the user.
import matplotlib.pyplot as plt	matplotlib.pyplot: A library used to create visualizations and plots (graphs).

Step-2: Global Variable for Dataframe

df = None	df: This is a global variable that will hold the COVID-19 data in the form of a pandas DataFrame. The data will either be fetched from an API or loaded from a CSV file.
-----------	---

Step-3: Fetching COVID-19 Data from API

def fetch_country_data(country):	fetch_country_data: This function accepts a country name, makes an API request to get COVID-19 historical data for that country, and returns the data if the request is successful.
url = f"https://disease.sh/v3/covid19/historical/{country}?lastdays=all"	The API URL is https://disease.sh/v3/covid-19/historical/{country}?lastdays=all, which fetches all available historical data for a country.
response = requests.get(url)	
if response.status_code == 200: data = response.json() return data['timeline'] else:	The function checks if the API request was successful by checking response.status_code. If the status is 200, it extracts the data using response.json() and

<pre>print (f"Error fetching data for {country}. Status code: {response.status_code}") return None</pre>	returns the "timeline" part (which contains the cases, deaths, recoveries, and vaccinations).
--	---

Step-4: Saving Data to a CSV File

<pre>def save_data_to_csv(country, data): cases = pd.DataFrame.from_dict(data['cases'], orient='index', columns=['cases']) deaths = pd.DataFrame.from_dict(data['deaths'], orient='index', columns=['deaths']) recovered = pd.DataFrame.from_dict(data['recovered'], orient='index', columns=['recovered']) vaccinations = pd.DataFrame.from_dict(data.get('vaccinated', {}), orient='index', columns=['vaccinations']) if vaccinations.empty: vaccinations = pd.DataFrame(index=cases.index, columns=['vaccinations']) vaccinations['vaccinations'] = 0 df = pd.concat([cases, deaths, recovered, vaccinations], axis=1) df.index = pd.to_datetime(df.index) df = df.reset_index().rename(columns={"index": "date"}) file_name = f"{country}_covid_data.csv" df.to_csv(file_name, index=False) print(f"Data for {country} has been saved to {file_name}.")</pre>	<p>save_data_to_csv: This function saves the fetched data to a CSV file.</p> <ul style="list-style-type: none"> • It creates individual DataFrames for cases, deaths, recoveries, and vaccinations. • If vaccination data is not available, it creates an empty vaccinations column and fills it with 0 values. • All data is combined into one DataFrame df, with the "date" as the index. • The DataFrame is saved to a CSV file named {country}_covid_data.csv
---	--

Step-5: Loading Data from CSV File

<pre>def load_country_data_from_csv(country): global df try: df = pd.read_csv(f"{country}_covid_data.csv") df['date'] = pd.to_datetime(df['date']) print(f"Data for {country} loaded from CSV.") return df except FileNotFoundError: messagebox.showerror("Error", f"CSV file for {country} not found.") return None</pre>	<p>load_country_data_from_csv: This function attempts to load the data from a CSV file into the global df variable.</p> <ul style="list-style-type: none"> • It reads the CSV file corresponding to the country and ensures the "date" column is in datetime format. • If the file does not exist, it shows an error message using a tkinter pop-up.
--	---

Step-6: Filtering Data by Date Range

<pre>def filter_by_date_range(df, start_date, end_date): start_date = pd.to_datetime(start_date) end_date = pd.to_datetime(end_date) return df[(df['date'] >= start_date) & (df['date'] <= end_date)]</pre>	<p>filter_by_date_range: This function filters the DataFrame by a specific date range.</p>
---	---

	<ul style="list-style-type: none"> • The start_date and end_date are converted to datetime objects. • It returns only the rows where the "date" falls within the specified range.
--	---

Step-7: Filtering Data by Specific Date

<pre>def filter_by_specific_date(df, specific_date): specific_date = pd.to_datetime(specific_date) return df[df['date'] == specific_date]</pre>	<p>filter_by_specific_date: This function filters the DataFrame for data on a specific date.</p> <ul style="list-style-type: none"> • It converts the specific_date to a datetime object and returns the rows where the "date" matches the provided date.
---	---

Step-8: Filtering Data by Year

<pre>def filter_by_year(df, year): df['year'] = df['date'].dt.year return df[df['year'] == year]</pre>	<p>filter_by_year: This function filters the DataFrame for data from a specific year.</p> <ul style="list-style-type: none"> • It extracts the year from the "date" column and filters the DataFrame for rows where the year matches the given year.
--	--

Step-9: Plotting the COVID-19 Trends

<pre>def plot_covid_trends(df, title, country): if df.empty: messagebox.showinfo("No Data", "No data available to plot.") return plt.figure(figsize=(12, 6)) plt.plot(df['date'], df['cases'], label='Cases', color='blue', linestyle='-', linewidth=2) plt.plot(df['date'], df['recovered'], label='Recoveries', color='green', linestyle='-', linewidth=2) plt.plot(df['date'], df['deaths'], label='Deaths', color='red', linestyle='-', linewidth=2) plt.plot(df['date'], df['vaccinations'], label='Vaccinations', color='purple', linestyle='-', linewidth=2) plt.title(f"{title} for {country.title()} ", fontsize=16) plt.xlabel("Date", fontsize=12) plt.ylabel("Counts", fontsize=12) plt.xticks(rotation=45) plt.grid(True, linestyle='--', alpha=0.6) plt.legend()</pre>	<p>plot_covid_trends: This function creates a plot of COVID-19 trends (cases, recoveries, deaths, vaccinations) over time.</p> <ul style="list-style-type: none"> • It plots each trend on the same graph, using different colors and labels. • It sets the title and labels for the x-axis and y-axis. • The x-axis labels (dates) are rotated for better visibility. • A grid is added for clarity, and a legend is included to identify each trend.
---	---

plt.tight_layout() plt.show()	
----------------------------------	--

Step-10: GUI for User Interaction

<pre>def run_dashboard(): def search_country(): global df country = country_entry.get().strip().lower() if not country: messagebox.showwarning("Input Error", "Please enter a country name.") return # Logic for searching or fetching country data (from CSV or API) ... # Other GUI components and logic for filtering and plotting the data ...</pre>	<p>run_dashboard: This function is the main entry point for the graphical user interface (GUI). It uses tkinter to create a window where the user can:</p> <ul style="list-style-type: none"> • Enter a country name. • Choose a filter option (by date range, specific date, or year). • Plot the corresponding COVID-19 data.
--	---

Explanation of Key Steps in GUI:

1. **search_country():** Handles the user's country input. If the country data is already saved in a CSV file, it loads it. If not, it fetches the data from the API and saves it.
2. **Option Buttons:** After a country is searched, the user can choose between filtering by date range, specific date, or year.
3. **Date Range / Date / Year Input:** Based on the chosen option, the user will input start and end dates or a specific year.
4. **Plotting:** Once the filter is applied, the corresponding data is plotted on a graph.

How to Use the Code:

1. **Run the Script:** Execute the script. A window will appear where you can input a country (e.g., "India", "USA").
2. **Country Search:** When you enter a country and press "Search", it checks if the data is available in a CSV file. If not, it fetches the data from the API and saves it.
3. **Choose Filter:** After the country data is loaded, you can select a filter option (date range, specific date, or year).
4. **View Plot:** Once the filter is applied, the graph will be displayed showing COVID-19 trends (cases, recoveries, deaths, and vaccinations).

Conclusion:

This code allows you to search for COVID-19 data by country, filter it by specific criteria (date range, specific date, or year), and visualize the trends using a graph. It also saves the fetched data in CSV files for future use, making the process faster when you search for the same country multiple times.