**Covid-19 Data Analysis Project**

**Fetching Data from Rapid API**

The code starts by importing the necessary Python modules: requests for making HTTP requests, json for working with JSON data, and csv for working with CSV files. It then defines the API endpoint URL and the required headers, including the API key and host information. Next, it sends a GET request to the API endpoint using requests.get() and retrieves the JSON data from the API response. If the request is successful, the retrieved JSON data is saved to a file named data.json using json.dump(). The code then extracts the relevant data from the JSON response, which is stored in the stats.breakdowns section. It opens a new CSV file named data.csv in write mode and creates a csv.DictWriter object, allowing it to write dictionaries (rows) to the CSV file. The field names (column headers) are taken from the keys of the first dictionary in the breakdowns list. The code writes the column headers to the CSV file using writer.writeheader() and then iterates through each dictionary (row) in the breakdowns list, writing it to the CSV file using writer.writerow(row). If any exception occurs during the API request, the code prints the error message. In summary, this Python code fetches global COVID-19 statistics data from an API, saves the raw JSON data, and then extracts specific information to write it to a CSV file for further analysis or processing.

**Data Transformation and Cleaning**

The code begins by importing the csv and ast modules from Python's standard library. It then defines the input and output CSV file paths. An empty list called country\_names is created to store the country names extracted from the data. The input CSV file is opened in read mode, and a csv.DictReader object is created to iterate over the rows as dictionaries. For each row, the 'location' column is parsed from a string to a dictionary using ast.literal\_eval(). The 'countryOrRegion' value is extracted from this dictionary and appended to the country\_names list. Next, the output CSV file is opened in write mode, and a csv.DictWriter object is created with the same fieldnames as the input file. The code writes the header row to the output file. Then, it reopens the input file and iterates over the rows and country\_names in parallel using zip(). For each row, the 'location' column is directly updated with the corresponding country name from the country\_names list. Finally, the modified row is written to the output CSV file. After iterating through all rows, a message is printed indicating that the modified data has been successfully written to the output CSV file.

**Data Visualization and Analysis**

The provided code is a Python script that performs data analysis and visualization on COVID-19 case data. Here's a multi-paragraph summary:

The script begins by importing the necessary libraries, such as pandas for data manipulation, matplotlib and plotly for data visualization. It then reads a CSV file named 'modified\_data.csv' into a pandas DataFrame. The data contains information about total confirmed cases, total deaths, and total recovered cases for various locations (countries/regions).

Next, the code creates several bar charts using plotly to visualize the COVID-19 case distribution across different locations. The first chart shows the total confirmed cases, total deaths, and total recovered cases side by side for each location. The second chart filters the data to only include locations with more than 4 million confirmed cases and stacks the bars for each metric. These visualizations provide an overview of the COVID-19 situation in different parts of the world.

The script then generates several bar charts using matplotlib to show the top 20 countries with the most and least COVID-19 cases, deaths, and recoveries. These charts help identify the countries that were most and least affected by the pandemic in terms of confirmed cases, casualties, and recoveries.

Additionally, the code creates a world map using plotly's choropleth visualization. The map displays the total confirmed cases for each country, with colors representing the case count. The map provides an interactive way to explore the global distribution of COVID-19 cases. Hover tooltips display additional information such as new cases, deaths, and recoveries for each country.

Finally, the script saves some of the visualizations as image files (JPEG) in the 'images' directory and generates an HTML file ('COVID-19 Cases Worldwide.html') to display the interactive world map. This code can be useful for analyzing and understanding the impact of the COVID-19 pandemic across different regions and countries.

**Connecting to MySQL Database**

This Python script is designed to transfer data from a CSV file named 'modified\_data.csv' into a MySQL database called 'covid'. It starts by importing the required Python modules to interact with MySQL databases and read CSV files. Then, it attempts to establish a connection to the MySQL database using predefined credentials like the username, password, host, port, and database name. If the connection is successful, it reads the contents of the CSV file line by line and stores the data in a list. Next, it creates a table called 'covid\_info' in the 'covid' database and inserts each row from the CSV file data into this table using SQL queries. The script carefully constructs the SQL statements to match the table structure and substitutes the actual data values from the CSV file. After inserting all the data, it commits the changes to make them permanent in the database. Finally, it closes the database connection and prints a message indicating the successful completion of the data transfer process.

**Interactive Covid-19 Dashboard**

Here we set up a Flask web application that serves as a dashboard for visualizing COVID-19 data. The application uses MySQL as the database backend to store user login information. The program starts by importing the necessary modules, including Flask, MySQL, and Werkzeug for password hashing. It then configures the Flask app and the MySQL connection details. The application has several routes defined, including the root route for the dashboard, the '/world\_map' route for displaying an interactive world map, '/register' for user registration, '/login' for user authentication, and '/logout' for logging out the user. The dashboard route retrieves the COVID-19 visualization images from the 'images' folder, encodes them as base64, and passes them to the 'dashboard.html' template for rendering. The '/world\_map' route reads the 'COVID-19 Cases Worldwide.html' file and serves its content.

The '/register' route handles user registration by accepting a name, email, and password from the form. It hashes the password using the Werkzeug library and stores the user information in the 'users' table of the MySQL database. The '/login' route authenticates users by checking the provided email and password against the database records. If the credentials are valid, it creates a session for the user and redirects them to the dashboard. The '/logout' route removes the user session and redirects to the login page. The code also includes a secret key for session management. Finally, the application is run in debug mode with the reloader disabled.

I hereby declare that the report I have written is correct.