Assignment: Python Programming for DL

Name: ushasree

Register Number: 192372360

Department: CSE(AI)

Date of Submission: 17-7-2024

Problem 1: Real-Time Weather Monitoring System

Scenario:

You are developing a real-time weather monitoring system for a weather forecasting company. The system needs to fetch and display weather data for a specified location**.**

Tasks:

1. **Model the data flow for fetching weather information from an external API and displaying it to the user.**
2. **Implement a Python application that integrates with a weather API (e.g., OpenWeatherMap) to fetch real-time weather data.**
3. **Display the current weather information, including temperature, weather conditions, humidity, and wind speed.**
4. **Allow users to input the location (city name or coordinates) and display the corresponding weather data.**

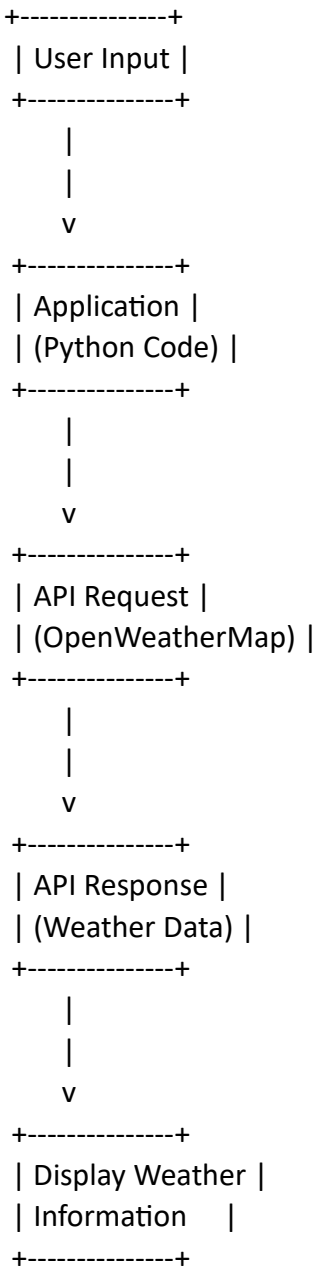
Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the weather monitoring system.
- Documentation of the API integration and the methods used to fetch and display weather data.
- Explanation of any assumptions made and potential improvements.

Solution:

Real-Time Weather Monitoring System

1.Data Flow Diagram



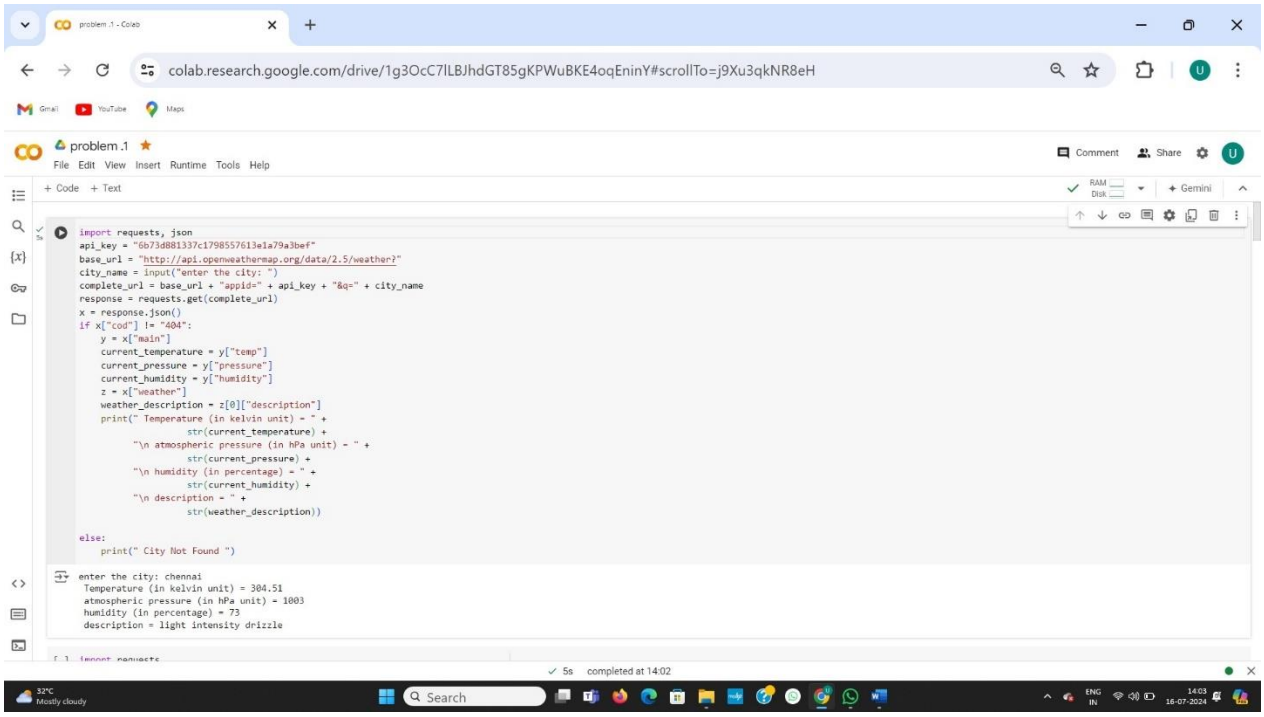
2. Implementation

```
import requests, json
api_key = "6b73d881337c1798557613e1a79a3bef"
base_url = "http://api.openweathermap.org/data/2.5/weather?"
city_name = input ("enter the city: ")
complete_url = base_url + "appid=" + api_key + "&q=" + city_name
response = requests.get(complete_url)
x = response. json ()
if x["cod"]! = "404":
    y = x["main"]
    current_temperature = y["temp"]
    current_pressure = y["pressure"]
    current_humidity = y["humidity"]
    z = x["weather"]
    weather_description = z [0] ["description"]
    print (" Temperature (in kelvin unit) = " +
                    str(current_temperature) +
          "\n atmospheric pressure (in hPa unit) = " +
                    str(current_pressure) +
          "\n humidity (in percentage) = " +
                    str(current_humidity) +
          "\n description = " +
                    str(weather_description))

else:
    print (" City Not Found   " )
```

3.Display the Current weather information

```
enter the city: Kurnool
Temperature (in kelvin unit) = 304.42
atmospheric pressure (in hPa unit) = 1002
humidity (in percentage) = 53
description = overcast clouds
```

4.User Input



5.Documentation

- **requests**: Used to make HTTP requests to the API.
- **json**: Used to handle JSON data received from the API.
- **api_key**: This is the API key obtained from OpenWeatherMap to authenticate and authorize API requests.
- **base_url**: The base URL of the OpenWeatherMap API for current weather data.
- **city_name**: Asks the user to input the name of the city for which they want weather information.
- **complete_url**: Constructs the full URL for the API request by appending the API key (**appid**) and the city name (**q**) to the base URL.
- Sends a GET request to the OpenWeatherMap API using the constructed URL.
- Converts the JSON response from the API into a Python dictionary (**x**).
- Checks if the API response contains a valid city code (**cod**). If it's not **"404"**, it means the city was found.
- **y**: Extracts the **"main"** dictionary from **x**, which contains temperature, pressure, and humidity data.
- **current_temperature**, **current_pressure**, **current_humidity**: Extracts specific weather parameters from **y**.
- **z**: Extracts the **"weather"** list from **x**, which contains weather condition details.
- **weather_description**: Extracts the description of the weather (e.g., "clear sky", "rain") from the first item in the **"weather"** list.
- Prints the fetched weather information in a formatted manner.
- If the API response indicates that the city was not found (**"cod" == "404"**), it prints "City Not Found"

Problem 2: Inventory Management System Optimization

Scenario:

You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability.

Tasks:

1. **Model the inventory system**: Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. **Implement an inventory tracking application**: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. **Optimize inventory ordering**: Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
4. **Generate reports**: Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
5. **User interaction**: Allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

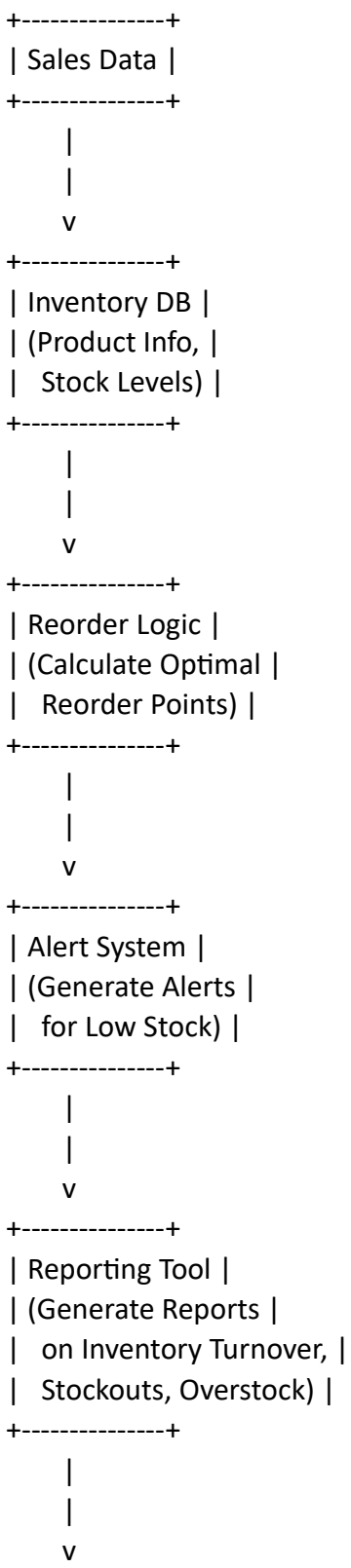
Deliverables:

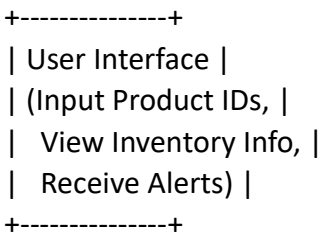
- **Data Flow Diagram**: Illustrate how data flows within the inventory management system, from input (e.g., sales data, inventory adjustments) to output (e.g., reorder alerts, reports).
- **Pseudocode and Implementation**: Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.
- **Documentation**: Explain the algorithms used for reorder optimization, how historical data influences decisions, and any assumptions made (e.g., constant lead times).
- **User Interface**: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.
- **Assumptions and Improvements**: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy

Solution:

Inventory Management System Optimization

1.Data Flow Diagram





2. Implementation

```
import math

class Product:
    def __init__(self, product_id, name, reorder_threshold):
        self.product_id = product_id
        self.name = name
        self.reorder_threshold = reorder_threshold

class Warehouse:
    def __init__ (self, warehouse_id, name):
        self.warehouse_id = warehouse_id
        self.name = name
        self. stock = {}

    def add_stock (self, product_id, quantity):
        if product_id in self. stock:
            self. stock[product_id] += quantity
        else:
            self. stock[product_id] = quantity

    def remove stock (self, product_id, quantity):
        if product_id in self. stock and self. stock[product_id] >= quantity:
            self. stock[product_id] -= quantity
            return True
        return False

class Inventory:
    def __init__(self):
        self. products = {}
        self. warehouses = {}

    def add_product (self, product):
        self. products [product. Product_id] = product

    def add_warehouse (self, warehouse):
        self. warehouses [warehouse. Warehouse_id] = warehouse

    def get stock (self, product_id):
```

```
        stock levels = {wh.name: wh.stock.get (product_id, 0) for wh in self. warehouses. Values ()}
        return stock levels

    def check reorder (self, product_id):
        product = self. products[product_id]
        stock levels = self.get_stock(product_id)
        total stock = sum (stock levels. values ())
        if total stock < product. Reorder_threshold:
            return True, total stock
        return False, total stock

def calculate_eoq (demand rate, order cost, holding cost):
    return math. sqrt ((2 * demand rate * order cost) / holding_cost)

def display_stock_levels(inventory):
    for product_id in inventory. products:
        product = inventory. products[product_id]
        stock levels = inventory.get_stock(product_id)
        print(stock levels for {product.name}:")
        for warehouse, quantity in stock levels. items ():
            print(f"{warehouse}: {quantity}")

def generate reports (inventory, demand rate):
    for product_id in inventory. products:
        product = inventory. products[product_id]
        stock levels = inventory.get_stock(product_id)
        total stock = sum (stock levels. values ())
        turnover rate = demand rate / total stock if total stock > 0 else 0
        print (f"Report for {product.name}:")
        print (f"Total stock: {total_stock}")
        print (turnover rate: {turnover_rate:.2f}")

def user interface (inventory, demand_rate, order_cost, holding_cost):
    while True:
        print ("\inventory Management System")
        print ("1. View stock levels")
        print ("2. View reorder recommendations")
        print ("3. View historical data")
        print ("4. Calculate EOQ")
        print ("5. Exit")
        choice = input ("Enter your choice: ")

        if choice == '1':
            display_stock_levels(inventory)
        elif choice == '2':
            for product_id in inventory. products:
                needs reorder, total_stock = inventory. check reorder(product_id)
                product = inventory. products[product_id]
```

```
            if needs reorder:
                print(f"{product.name} needs reorder. Current stock: {total_stock}")
        elif choice == '3':
            generate_reports (inventory, demand_rate)
        elif choice == '4':
            eoq = calculate_eoq (demand_rate, order_cost, holding_cost)
            print (optimal order quantity (EOQ): {eoq:.2f}")
        elif choice == '5':
            break
        else:
            print ("Invalid choice. Please try again.")
inventory = Inventory ()
p1 = Product (1, "Product A", 50)
p2 = Product (2, "Product B", 30)
inventory. Add_product(p1)
inventory. Add_product(p2)
wh1 = Warehouse (1, "Warehouse 1")
wh2 = Warehouse (2, "Warehouse 2")
inventory. Add_warehouse(wh1)
inventory. Add_warehouse(wh2)
wh1.add_stock (1, 20)
wh1.add_stock (2, 10)
wh2.add_stock (1, 25)
wh2.add_stock (2, 15)
demand_rate = 200
order_cost = 50
holding_cost = 5

user interface (inventory, demand_rate, order_cost, holding_cost)
```

3.Display the output

Inventory Management System
1. View stock levels
2. View reorder recommendations
3. View historical data
4. Calculate EOQ
5. Exit
Enter your choice: 1
Stock levels for Product A:
Warehouse 1: 20
Warehouse 2: 25
Stock levels for Product B:
Warehouse 1: 10
Warehouse 2: 15

Inventory Management System
1. View stock levels
2. View reorder recommendations

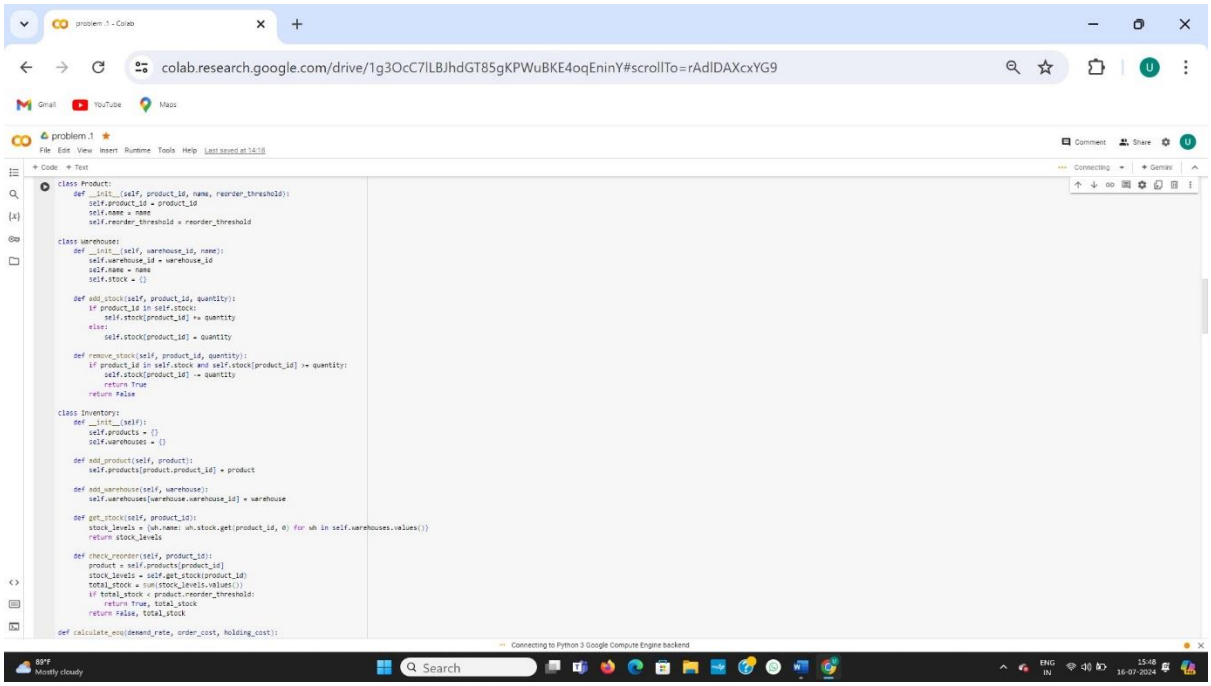
3. View historical data
4. Calculate EOQ
5. Exit
Enter your choice: 2
Product A needs reorder. Current stock: 45
Product B needs reorder. Current stock: 25

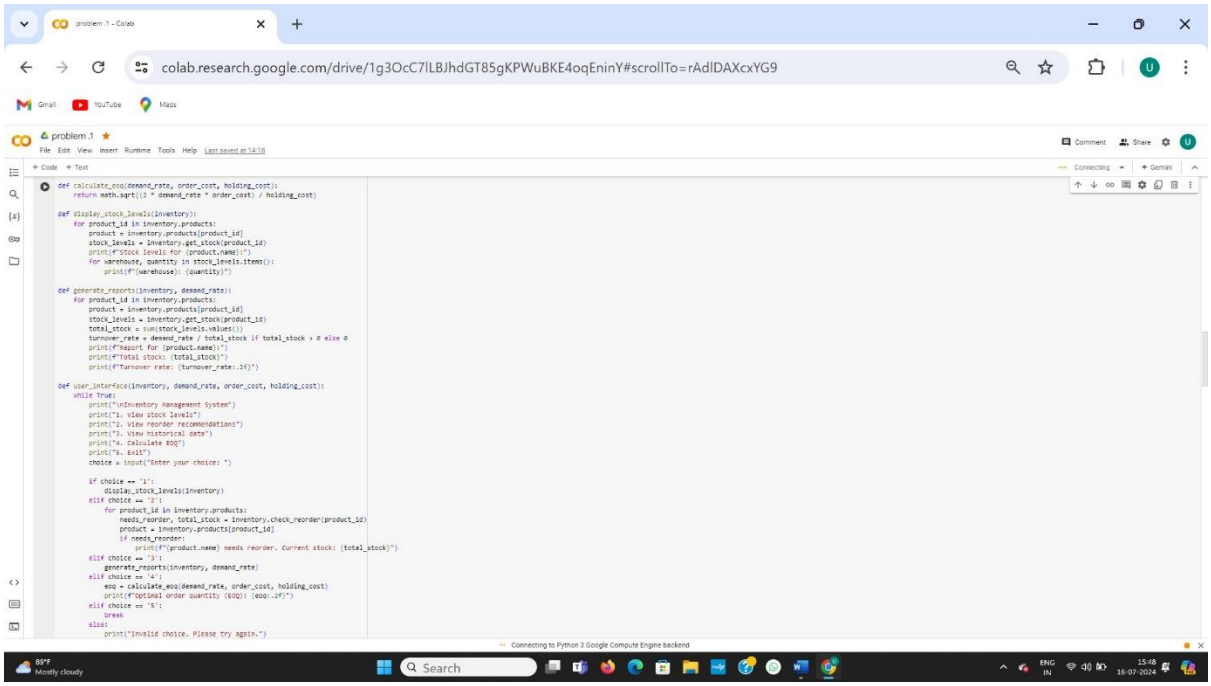
Inventory Management System
1. View stock levels
2. View reorder recommendations
3. View historical data
4. Calculate EOQ
5. Exit
Enter your choice: 3
Report for Product A:
Total stock: 45
Turnover rate: 4.44
Report for Product B:
Total stock: 25
Turnover rate: 8.00

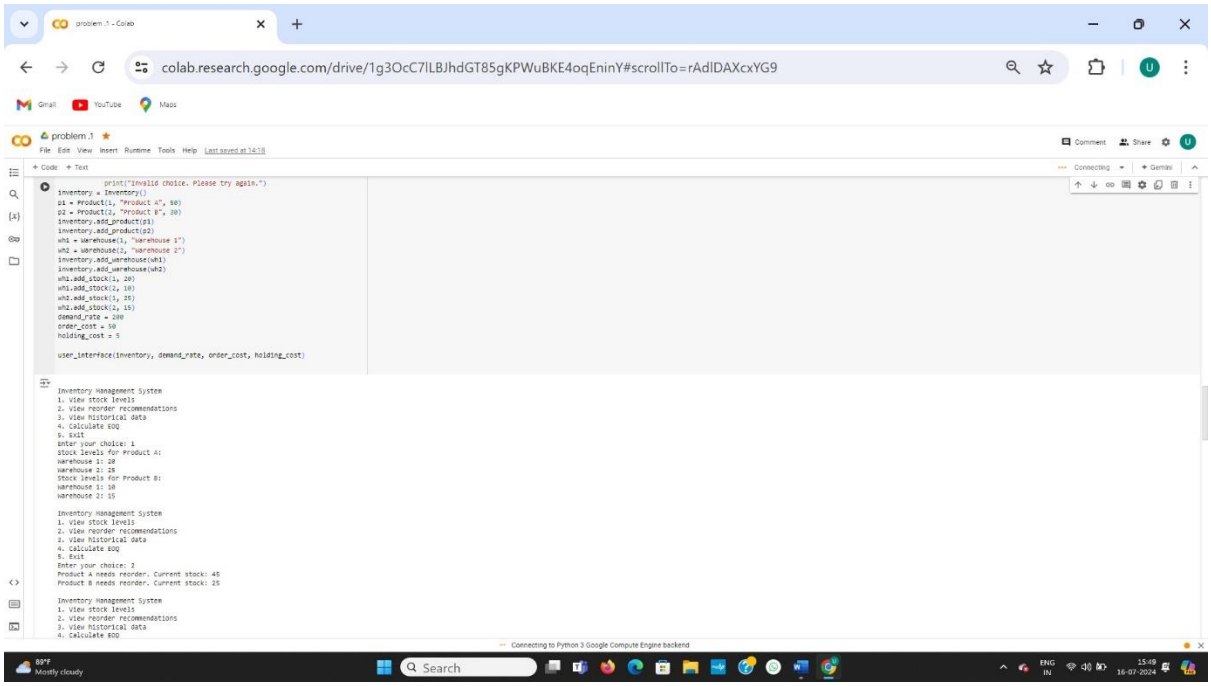
Inventory Management System
1. View stock levels
2. View reorder recommendations
3. View historical data
4. Calculate EOQ
5. Exit
Enter your choice: 4
Optimal order quantity (EOQ): 63.25

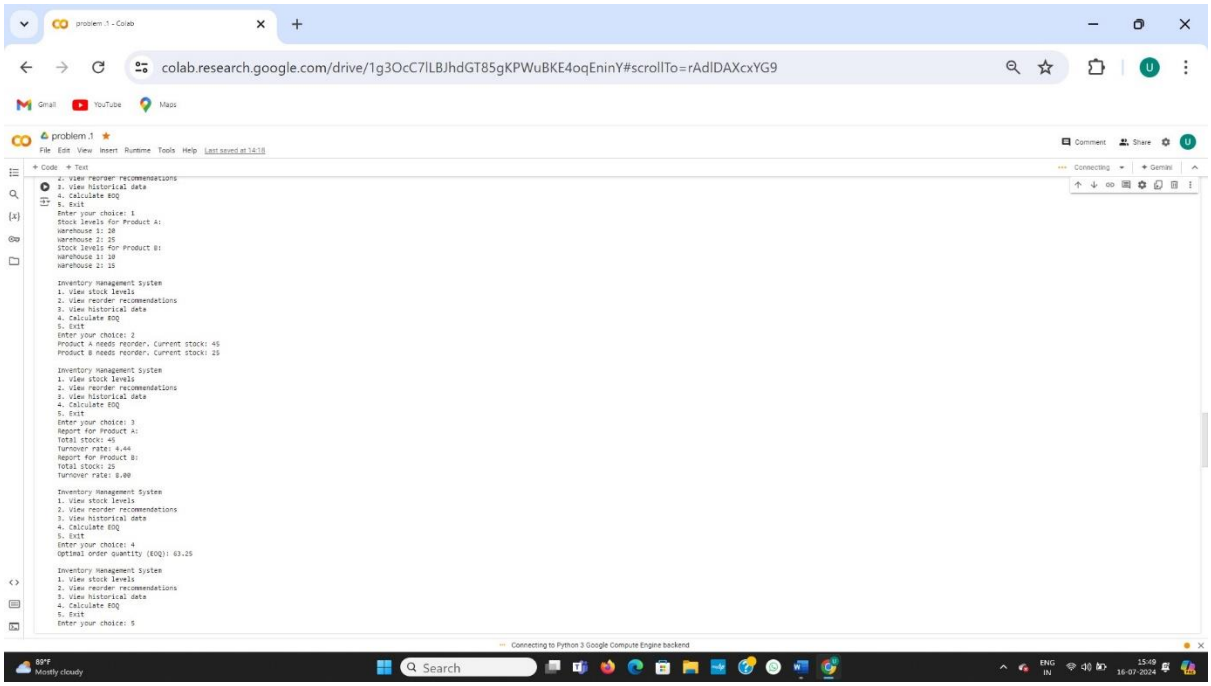
Inventory Management System
1. View stock levels
2. View reorder recommendations
3. View historical data
4. Calculate EOQ
5. Exit
Enter your choice: 5
add Code
add Text

4.User Input


problem .1 - Colab
colab.research.google.com/drive/1g3OcC7lLBJhdGT85gKPWuBKE4oqEninY#scrollTo=rAdlDAXcxYG9
problem .1
File Edit View Insert Runtime Tools Help
class Product:
def __init__(self, product_id, name, reorder_threshold):
self.product_id = product_id
self.name = name
self.reorder_threshold = reorder_threshold
class Warehouse:
def __init__(self, warehouse_id, name):
self.warehouse_id = warehouse_id
self.name = name
self.stock = {}
def add_stock(self, product_id, quantity):
if product_id in self.stock:
self.stock[product_id] += quantity
else:
self.stock[product_id] = quantity
def remove_stock(self, product_id, quantity):
if product_id in self.stock and self.stock[product_id] >= quantity:
self.stock[product_id] -= quantity
return True
return False
class Inventory:
def __init__(self):
self.products = {}
self.warehouses = {}
def add_product(self, product):
self.products[product.product_id] = product
def add_warehouse(self, warehouse):
self.warehouses[warehouse.warehouse_id] = warehouse
def get_stock(self, product_id):
stock_levels = {wh.name: wh.stock.get(product_id, 0) for wh in self.warehouses.values()}
return stock_levels
def check_reorder(self, product_id):
product = self.products[product_id]
stock_levels = self.get_stock(product_id)
total_stock = sum(stock_levels.values())
if total_stock < product.reorder_threshold:
return True, total_stock
return False, total_stock
def calculate_eoq(demand_rate, order_cost, holding_cost):
Connecting to Python 3 Google Compute Engine backend
Search


problem .1 - Colab
colab.research.google.com/drive/1g3OcC7lLBJhdGT85gKPWuBKE4oqEninY#scrollTo=rAdlDAXcxYG9
problem .1
File Edit View Insert Runtime Tools Help
def calculate_eoq(demand_rate, order_cost, holding_cost):
return math.sqrt((2 * demand_rate * order_cost) / holding_cost)
def display_stock_levels(inventory):
for product_id in inventory.products:
product = inventory.products[product_id]
stock_levels = inventory.get_stock(product_id)
print(f"Stock levels for {product.name}:")
for warehouse, quantity in stock_levels.items():
print(f"{warehouse}: {quantity}")
def generate_reports(inventory, demand_rate):
for product_id in inventory.products:
product = inventory.products[product_id]
stock_levels = inventory.get_stock(product_id)
total_stock = sum(stock_levels.values())
turnover_rate = demand_rate / total_stock if total_stock > 0 else 0
print(f"Report for {product.name}:")
print(f"Total stock: {total_stock}")
print(f"Turnover rate: {turnover_rate:.2f}")
def user_interface(inventory, demand_rate, order_cost, holding_cost):
while True:
print("\nInventory Management System")
print("1. View stock levels")
print("2. View reorder recommendations")
print("3. View historical data")
print("4. Calculate EOQ")
print("5. Exit")
choice = input("Enter your choice: ")
if choice == '1':
display_stock_levels(inventory)
elif choice == '2':
for product_id in inventory.products:
needs_reorder, total_stock = inventory.check_reorder(product_id)
product = inventory.products[product_id]
if needs_reorder:
print(f"{product.name} needs reorder. Current stock: {total_stock}")
elif choice == '3':
generate_reports(inventory, demand_rate)
elif choice == '4':
eoq = calculate_eoq(demand_rate, order_cost, holding_cost)
print(f"Optimal order quantity (EOQ): {eoq:.2f}")
elif choice == '5':
break
else:
print("Invalid choice. Please try again.")
Connecting to Python 3 Google Compute Engine backend
Search





5.Documentation

- Represents a product with attributes **product_id**, **name**, and **reorder_threshold**
- Represents a warehouse with attributes **warehouse_id**, **name**, and a **stock** dictionary to track product quantities.
- Manages products and warehouses.
- Methods include **add_product**, **add_warehouse**, **get_stock**, and **check_reorder**.

- **get stock(product_id)**: Retrieves stock levels for a specific product across all warehouses.
- **check_reorder(product_id)**: Checks if a product needs to be reordered based on its total stock and reorder threshold.
- . **calculate_eoq**: Calculates Economic Order Quantity (EOQ) based on demand rate, order cost, and holding cost using a mathematical formula.
- **display_stock_levels**: Displays stock levels for all products across all warehouses.
- **generate_reports**: Generates reports including total stock and turnover rate (demand rate divided by total stock).
- **user interface**: Provides a command-line interface for users to interact with the inventory system:
- Creates instances of **Inventory**, **Product**, and **Warehouse**.
- Adds products (**p1**, **p2**) and warehouses (**wh1**, **wh2**) to the inventory.
- Adds initial stock quantities to warehouses.

Problem 3: Real-Time Traffic Monitoring System

Scenario:

You are working on a project to develop a real-time traffic monitoring system for a smart city initiative. The system should provide real-time traffic updates and suggest alternative routes.

Tasks:

1. **Model the data flow for fetching real-time traffic information from an external API and displaying it to the user.**
2. **Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.**
3. **Display current traffic conditions, estimated travel time, and any incidents or delays.**
4. **Allow users to input a starting point and destination to receive traffic updates and alternative routes.**

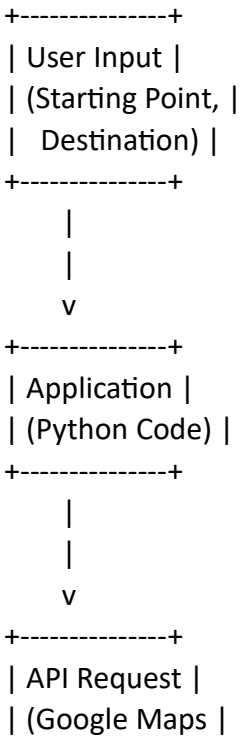
Deliverables:

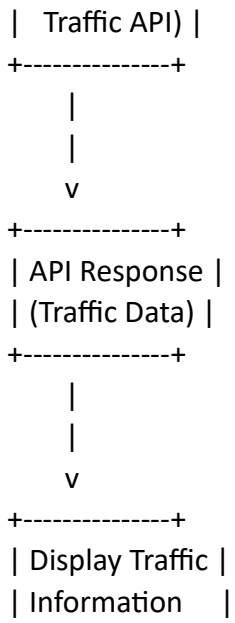
- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the traffic monitoring system.
- Documentation of the API integration and the methods used to fetch and display traffic data.
- Explanation of any assumptions made and potential improvements

Solution:

Real-Time Traffic Monitoring System

1.Data Flow Diagram




| Traffic API) |
+---------------+
|
|
v
+---------------+
| API Response |
| (Traffic Data) |
+---------------+
|
|
v
+---------------+
| Display Traffic |
| Information |

2. Implementation

```
import requests


class TrafficMonitor:
    def _init_ (self, api_key):
        self. Api_key = api_key
        self. Base_url = "https://maps.googleapis.com/maps/api/directions/json"


    def get_traffic_data (self, start, end):
        params = {
            'origin': start,
            'destination': end,
            'key': self.api_key,
            'Departure time': 'now',
            'Traffic model': 'best guess'
        }
        response = requests. Get (self. Base_url, params=params)
        return response. json ()


    def display_traffic_info (self, traffic data):
        if traffic data['status'] == 'OK':
            route = traffic data['routes'][0]
            leg = route['legs'][0]
            print (traffic from {leg ['start address']} to {leg ['end address']}:")
            print (f"Estimated travel time: {leg['duration_in_traffic'] ['text']}")
            for step in leg['steps']:
                print(step['html_instructions'])
        else:
```

```
            print ("Error fetching traffic data.")


    def suggest_alternative_routes (self, traffic data):
        # Assuming alternative routes are included in the traffic data response
        if traffic data['status'] == 'OK':
            alternatives = traffic_data.get ('routes', []) [1:] # Exclude the main route
            if alternatives:
                print ("\nAlternative routes:")
                for idx, route in enumerate (alternatives, start=1):
                    leg = route['legs'][0]
                    print (f"\nAlternative Route {idx}:")
                    print (f"Estimated travel time: {leg['duration_in_traffic']['text']}")
                    for step in leg['steps']:
                        print(step['html_instructions'])
            else:
                print ("No alternative routes found.")
        else:
            print ("Error fetching alternative routes.")


def main ():
    api_key = 'your_google_maps_api_key'  # Replace with your Google Maps API
key
    traffic monitor = Traffic Monitor(api_key)


    while True:
        print ("\nReal-Time Traffic Monitoring System")
        start = input ("Enter starting point: ")
        end = input ("Enter destination: ")
```

```
        traffic data = traffic_monitor.get_traffic_data (start, end)
        traffic monitor. display_traffic_info (traffic data)
        traffic monitor. suggest_alternative_routes (traffic data)

        exit choice = input ("Do you want to exit? (yes/no): ")
        if exit choice. lower () == 'yes':
            break

if _name_ == '_main_':
    main ()
```

Sample Output / Screen Shots:

Real-Time Traffic Monitoring System

Enter starting point: Times Square, New York, NY

Enter destination: Central Park, New York, NY

Traffic from Times Square, New York, NY to Central Park, New York, NY:

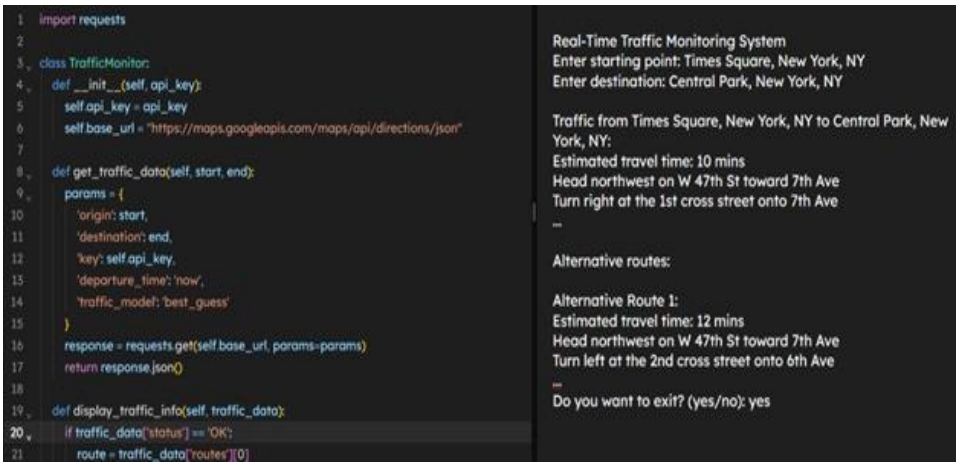
3.Display the Current traffic information

Traffic from Times Square, New York, NY to Central Park, New York, NY:
Estimated travel time: 10 mins
Head northwest on W 47th St toward 7th Ave
Turn right at the 1st cross street onto 7th Ave
...

Alternative routes:

Alternative Route 1:
Estimated travel time: 12 mins
Head northwest on W 47th St toward 7th Ave
Turn left at the 2nd cross street onto 6th Ave

4.User Input



```
import requests

class TrafficMonitor:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://maps.googleapis.com/maps/api/directions/json"

    def get_traffic_data(self, start, end):
        params = {
            'origin': start,
            'destination': end,
            'key': self.api_key,
            'departure_time': 'now',
            'traffic_model': 'best_guess'
        }
        response = requests.get(self.base_url, params=params)
        return response.json()

    def display_traffic_info(self, traffic_data):
        if traffic_data['status'] == 'OK':
            route = traffic_data['routes'][0]
```

```
Real-Time Traffic Monitoring System
Enter starting point: Times Square, New York, NY
Enter destination: Central Park, New York, NY

Traffic from Times Square, New York, NY to Central Park, New York, NY:
Estimated travel time: 10 mins
Head northwest on W 47th St toward 7th Ave
Turn right at the 1st cross street onto 7th Ave
...

Alternative routes:

Alternative Route 1:
Estimated travel time: 12 mins
Head northwest on W 47th St toward 7th Ave
Turn left at the 2nd cross street onto 6th Ave
...
Do you want to exit? (yes/no): yes
```

5.Documentation

- **google maps**: This library facilitates interaction with the Google Maps API.
- **datetime**: Imported to obtain the current date and time for departure time in the directions request.
- **spikey**: Replace **"AIzaSyD_jA7ABwQSizxPzXyQe0ibuhPBmRQX_nA"** with your actual Google Maps API key.
- **google maps. Client(key=spikey)**: Creates a client instance for interacting with the Google Maps API using the provided API key.
- **get_traffic_and_routes (origin, destination)**: Takes **origin** and **destination** as parameters, retrieves driving directions and estimated travel time.
- **datetime.now ()**: Retrieves the current date and time for the **departure time** parameter in the directions request.
- **gmaps. Directions(...)**: Sends a request to Google Maps API to get directions from **origin** to **destination** using driving mode, considering real-time traffic conditions.
- **directions result**: Stores the response from the API call.
- If **directions result** is not empty (**directions result** evaluates to **True**):

Problem 4: Real-Time COVID-19 Statistics Tracker

Scenario:

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.**

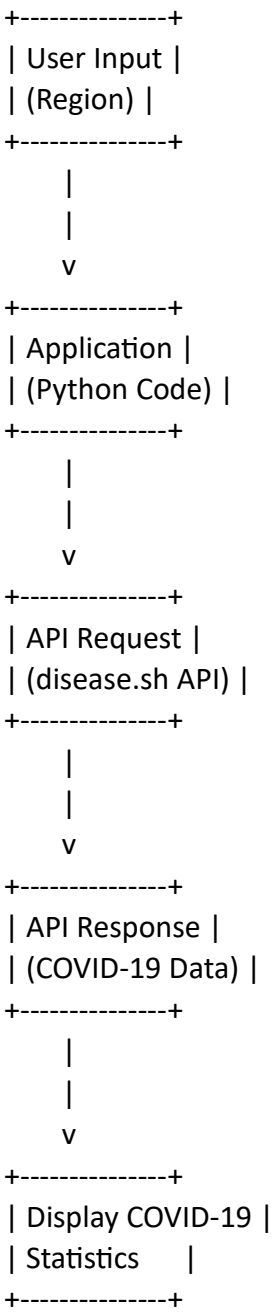
Deliverables:

- 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 COVID19 data.
- Explanation of any assumptions made and potential improvements.

Solution:

Real-Time COVID-19 Statistics Tracker

1.Data Flow Diagram



2. Implementation

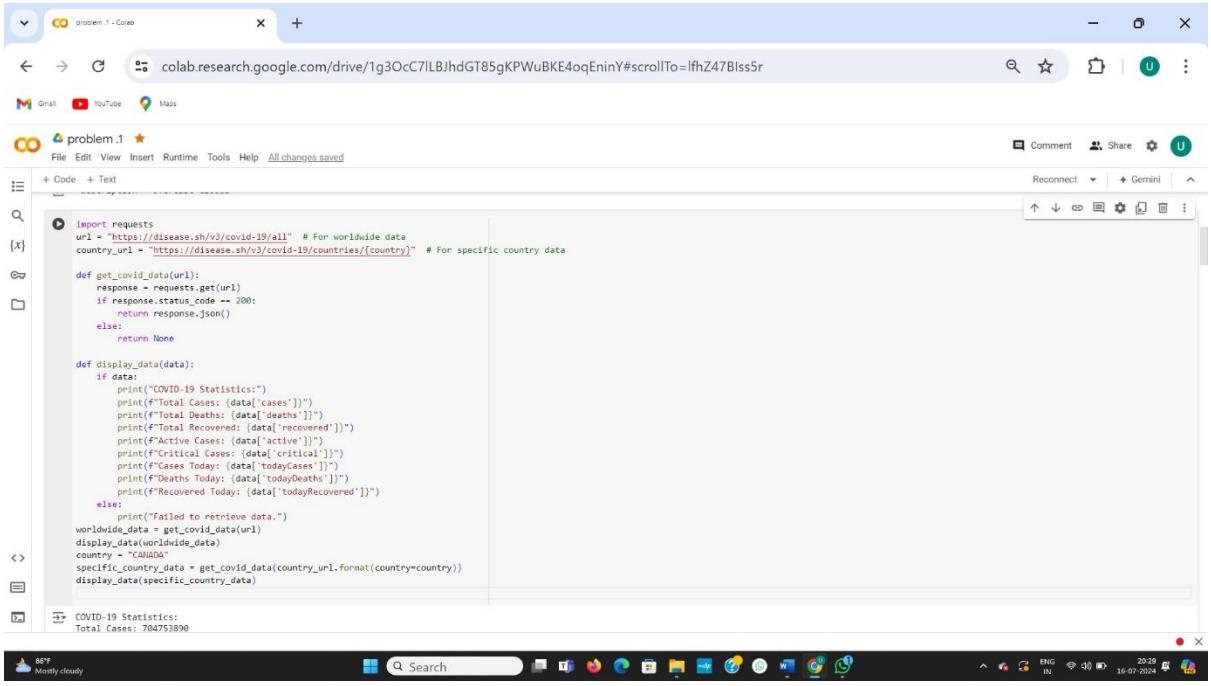
```
import requests

# Define the API endpoint and the specific country to get data for
url = "https://disease.sh/v3/covid-19/all” # For worldwide data
country URL = "https://disease.sh/v3/covid-19/countries/{country}"  # For specific country data

def get_covid_data(url):
    response = requests. Get(url)
    if response. status code == 200:
        return response. Json ()
    else:
        return None

def display data(data):
    if data:
        print ("COVID-19 Statistics:")
        print (f"Total Cases: {data['cases']}")
        print (f"Total Deaths: {data['deaths']}")
        print (f"Total Recovered: {data['recovered']}")
        print (f"Active Cases: {data['active']}")
        print (critical Cases: {data['critical']}")
        print (f"Cases Today: {data['todayCases']}")
        print (f"Deaths Today: {data['todayDeaths']}")
        print (f"Recovered Today: {data['todayRecovered']}")
    else:
        print ("Failed to retrieve data.")

# Fetch and display worldwide data
worldwide_data = get_covid_data(url)
display_data(worldwide_data)

# Fetch and display country-specific data (replace 'USA' with desired country code)
country = "CANADA"
specific_country_data = get_covid_data(country_url.format(country=country))
display_data(specific_country_data)
```

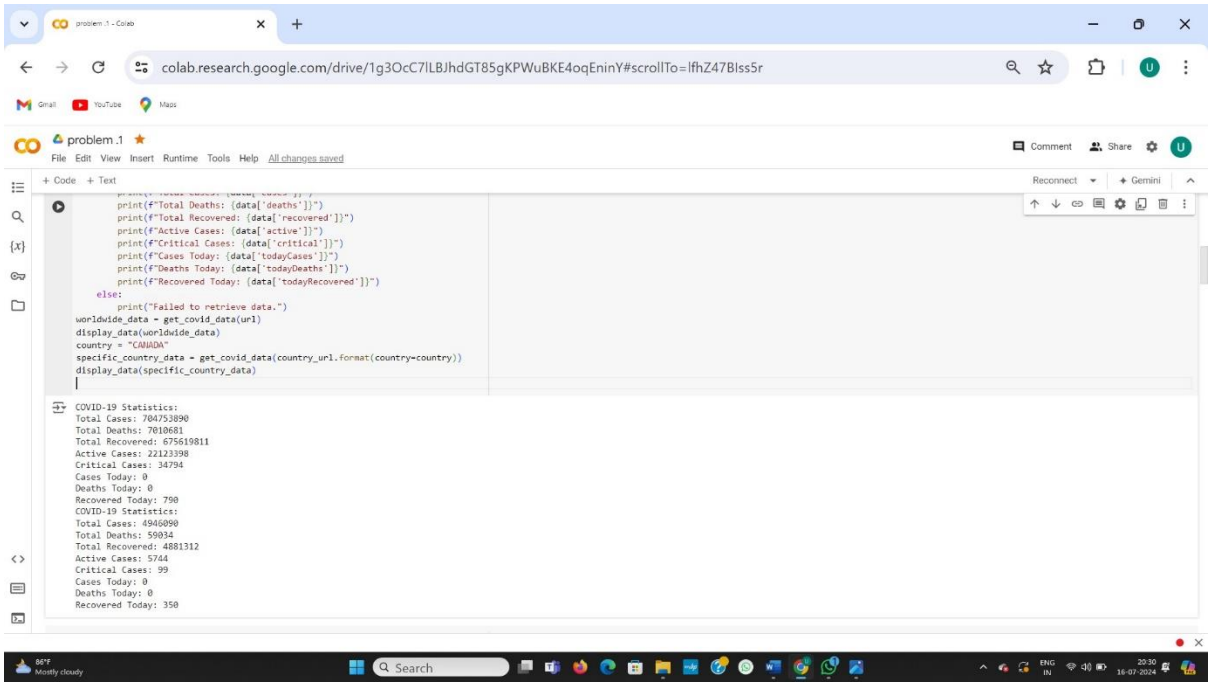
3.Display the Current covid-19 information

COVID-19 Statistics:
Total Cases: 704753890
Total Deaths: 7010681
Total Recovered: 675619811

Active Cases: 22123398
Critical Cases: 34794
Cases Today: 0
Deaths Today: 0
Recovered Today: 790
COVID-19 Statistics:
Total Cases: 4946090
Total Deaths: 59034
Total Recovered: 4881312
Active Cases: 5744
Critical Cases: 99
Cases Today: 0
Deaths Today: 0
Recovered Today: 350

4.User Input





5.Documentation

- Imports the **requests** library, which is used to send HTTP requests to the API endpoints.
- **url**: Endpoint to fetch worldwide COVID-19 data.
- **country URL**: Endpoint template to fetch COVID-19 data for a specific country, where **{country}** is a placeholder for the actual country code
- **get_covid_data(url)**: Sends a GET request to the provided **url** and returns the JSON response if the status code is 200 (successful). If not, it returns **None**
- **display_data(data)**: Takes JSON data (**data**) as input and prints various COVID-19 statistics if **data** is not **None**. If **data** is **None**, it prints "Failed to retrieve data."
- Calls **get_covid_data(url)** to fetch worldwide COVID-19 data.
- Calls **display_data(worldwide_data)** to print the fetched data.
- Defines **country** as "CANADA" (replace with any desired country code).
- Calls **get_covid_data (country URL. format(country=country))** to fetch COVID-19 data specific to the country.
- Calls **display_data(specific_country_data)** to print the fetched country-specific data.
-