Assignment: Python Programming for DL

Name: P.eshwar

Register Number: 192321103

Department: information technology

Date of Submission: 17-07-24

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., Open Weather Map) 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.

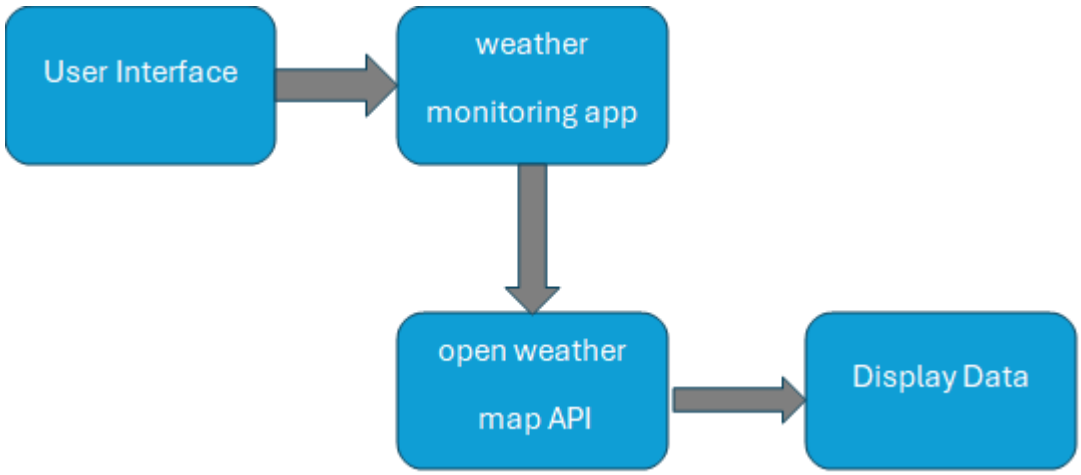
• Documentation of the API integration and the methods used to fetch and display weather data.

• Explanation of any assumptions made and potential improvements

Solution:

Problem 1: Real-Time Weather Monitoring System

Data Flow Diagram



Pseudocode:

1. Get user input for the location.

2. Send a request to the weather API with the location.

3. Receive and parse the weather data from the API.

4. Display the weather information to the user.

CODE

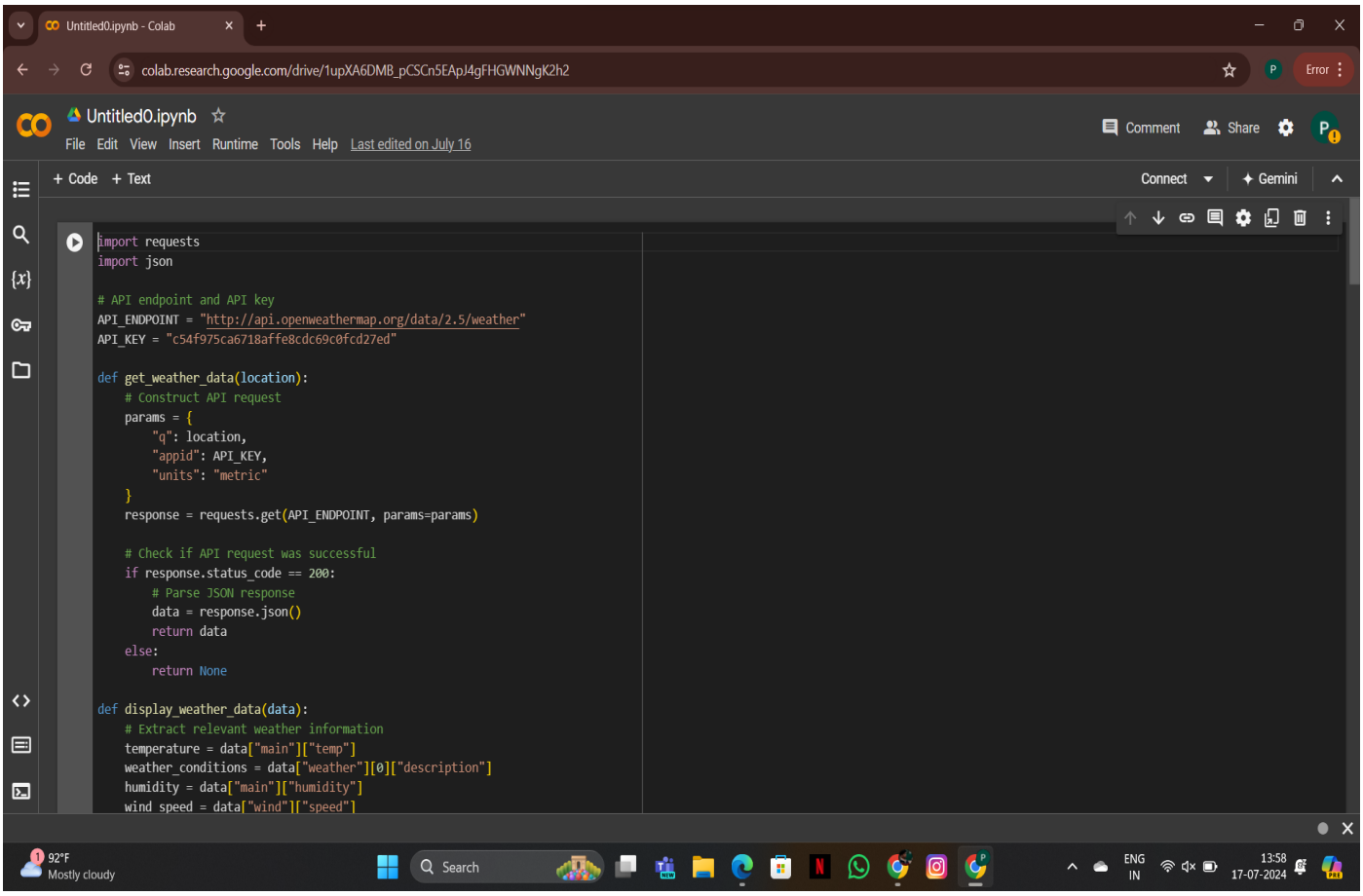
```
import requests
```

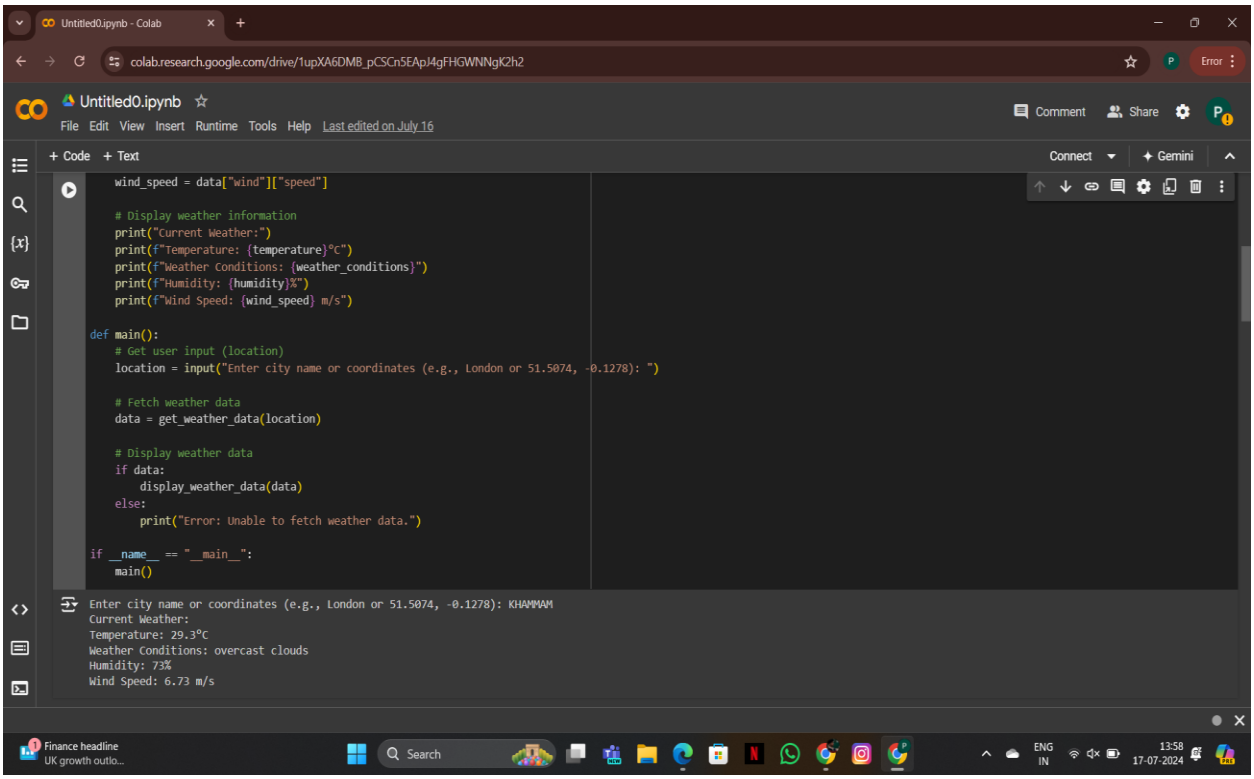
```
import json

# API endpoint and API key
API_ENDPOINT = "http://api.openweathermap.org/data/2.5/weather"
API_KEY = "c54f975ca6718affe8cdc69c0fcd27ed"

def get_weather_data(location):
    # Construct API request
    params = {
        "q": location,
        "appid": API_KEY,
        "units": "metric"
    }
    response = requests.get(API_ENDPOINT, params=params)

    # Check if API request was successful
    if response.status_code == 200:
        # Parse JSON response
        data = response.json()
        return data
    else:
        return None

def display_weather_data(data):
    # Extract relevant weather information
    temperature = data["main"]["temp"]
    weather_conditions = data["weather"][0]["description"]
    humidity = data["main"]["humidity"]
    wind_speed = data["wind"]["speed"]

    # Display weather information
    print("Current Weather:")
    print(f"Temperature: {temperature}°C")
    print(f"Weather Conditions: {weather_conditions}")
    print(f"Humidity: {humidity}%")
    print(f"Wind Speed: {wind_speed} m/s")
```

```
def main():
    # Get user input (location)
    location = input("Enter city name or coordinates (e.g., London or 51.5074, -
0.1278): ")

    # Fetch weather data
    data = get_weather_data(location)

    # Display weather data
    if data:
        display_weather_data(data)
    else:
        print("Error: Unable to fetch weather data.")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
Enter city name or coordinates (e.g., London or 51.5074, -0.1278): KHAMMAM
Current Weather: Temperature: 29.3°C
 Weather Conditions: overcast clouds Humidity: 73%
 Wind Speed: 6.73 m/s

USER INPUT
```


Untitled0.ipynb - Colab
colab.research.google.com/drive/1upXA6DMB_pCSCn5EApJ4gFHGWNNgK2h2
Untitled0.ipynb
File Edit View Insert Runtime Tools Help Last edited on July 16
Comment
Share
+ Code + Text
Connect
Gemini
import requests
import json
# API endpoint and API key
API_ENDPOINT = "http://api.openweathermap.org/data/2.5/weather"
API_KEY = "c54f975ca6718affe8cdc69c0fcd27ed"
def get_weather_data(location):
# Construct API request
params = {
"q": location,
"appid": API_KEY,
"units": "metric"
}
response = requests.get(API_ENDPOINT, params=params)
# Check if API request was successful
if response.status_code == 200:
# Parse JSON response
data = response.json()
return data
else:
return None
def display_weather_data(data):
# Extract relevant weather information
temperature = data["main"]["temp"]
weather_conditions = data["weather"][0]["description"]
humidity = data["main"]["humidity"]
wind_speed = data["wind"]["speed"]
92°F
Mostly cloudy
Search
ENG
IN
13:58
17-07-2024



Documentation:

1API Integration: We use the Open Weather Map API to fetch real-time weather data

.2.**Methods**: The get weather function handles the API request and response processing. The main function handles user input and displays the data.

3.**Assumptions**: The user provides a valid city name.

4.Improvements: Error handling can be enhanced, and additional features like forecast data can be added.

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).

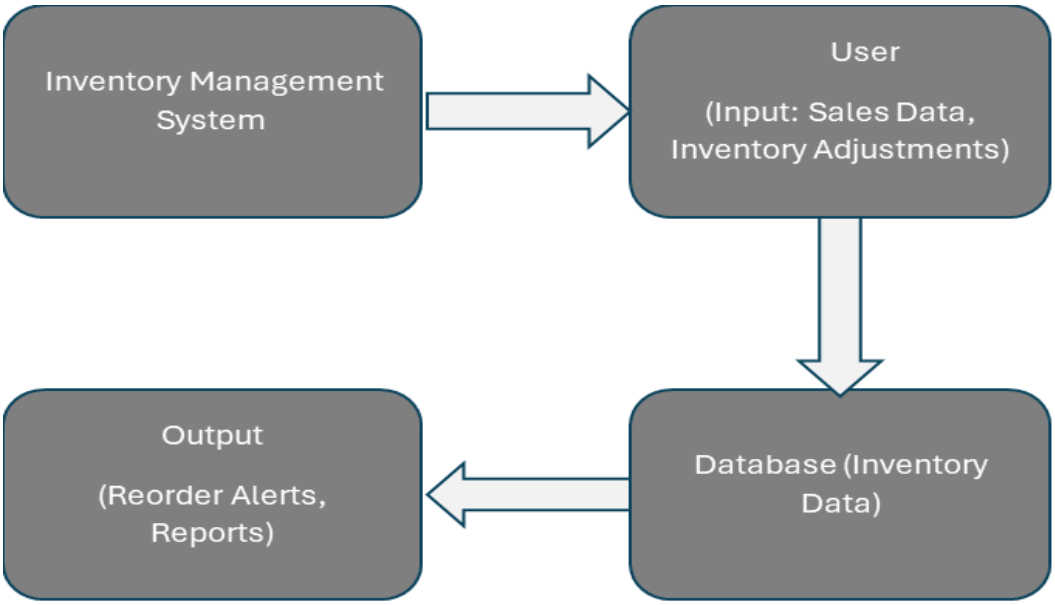
• User Interface: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.

• 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

DATA FLOW DIAGRAM



Pseudocode:

1. Define the structure for products, warehouses, and stock levels.
2. Track inventory levels in real-time.
3. Calculate reorder points based on historical sales data, lead times, and demand forecasts.
4. Generate reports on inventory turnover rates, stockout occurrences, and overstock costs.
5. Allow user interaction to view inventory levels, reorder recommendations, and historical data.

CODE

```
# inventory.py
class Product:
    def __init__(self, id, name, stock_level, reorder_point, reorder_quantity):
```

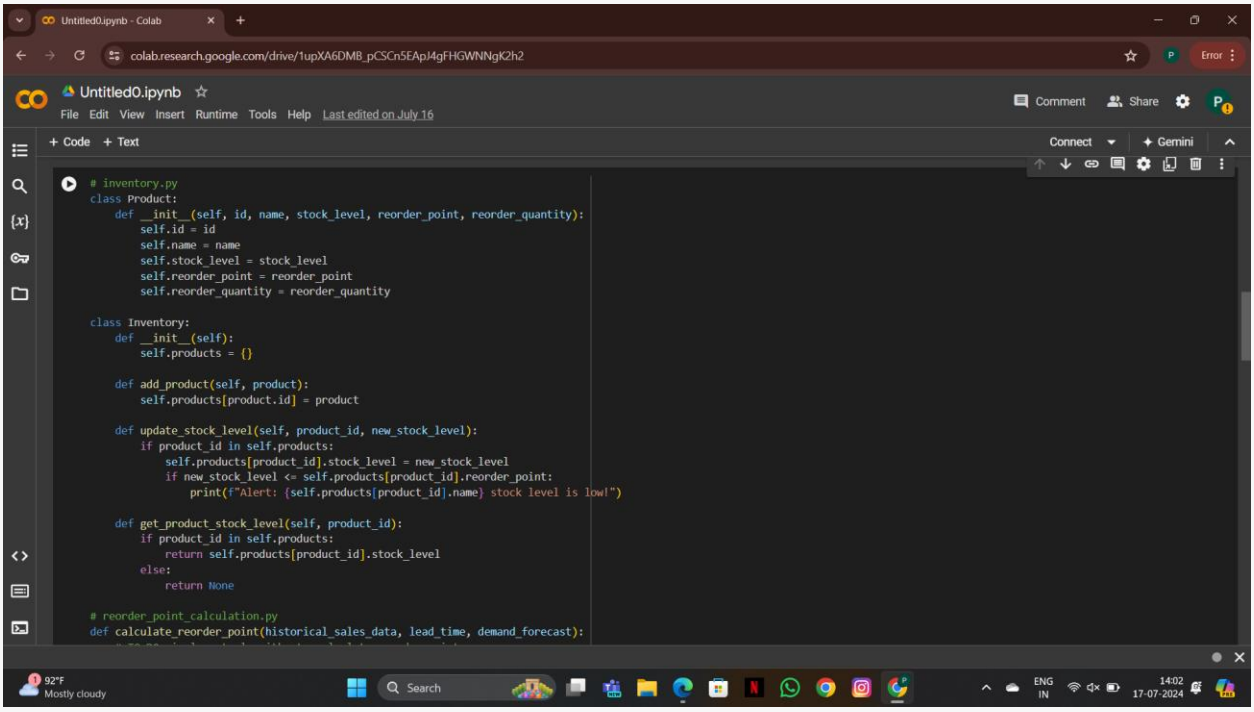
```
        self.id = id
        self.name = name
        self.stock_level = stock_level
        self.reorder_point = reorder_point
        self.reorder_quantity = reorder_quantity

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

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

    def update_stock_level(self, product_id, new_stock_level):
        if product_id in self.products:
            self.products[product_id].stock_level = new_stock_level
            if new_stock_level <= self.products[product_id].reorder_point:
                print(f"Alert: {self.products[product_id].name} stock level is
low!")

    def get_product_stock_level(self, product_id):
        if product_id in self.products:
            return self.products[product_id].stock_level
        else:
            return None

# reorder_point_calculation.py
def calculate_reorder_point(historical_sales_data, lead_time, demand_forecast):
    # TO DO: implement algorithm to calculate reorder point
    pass

# main.py
inventory = Inventory()

# add products to inventory
product1 = Product(1, "Product A", 100, 50, 200)
product2 = Product(2, "Product B", 200, 100, 300)
inventory.add_product(product1)
inventory.add_product(product2)

# update stock levels
inventory.update_stock_level(1, 80)
inventory.update_stock_level(2, 250)
```

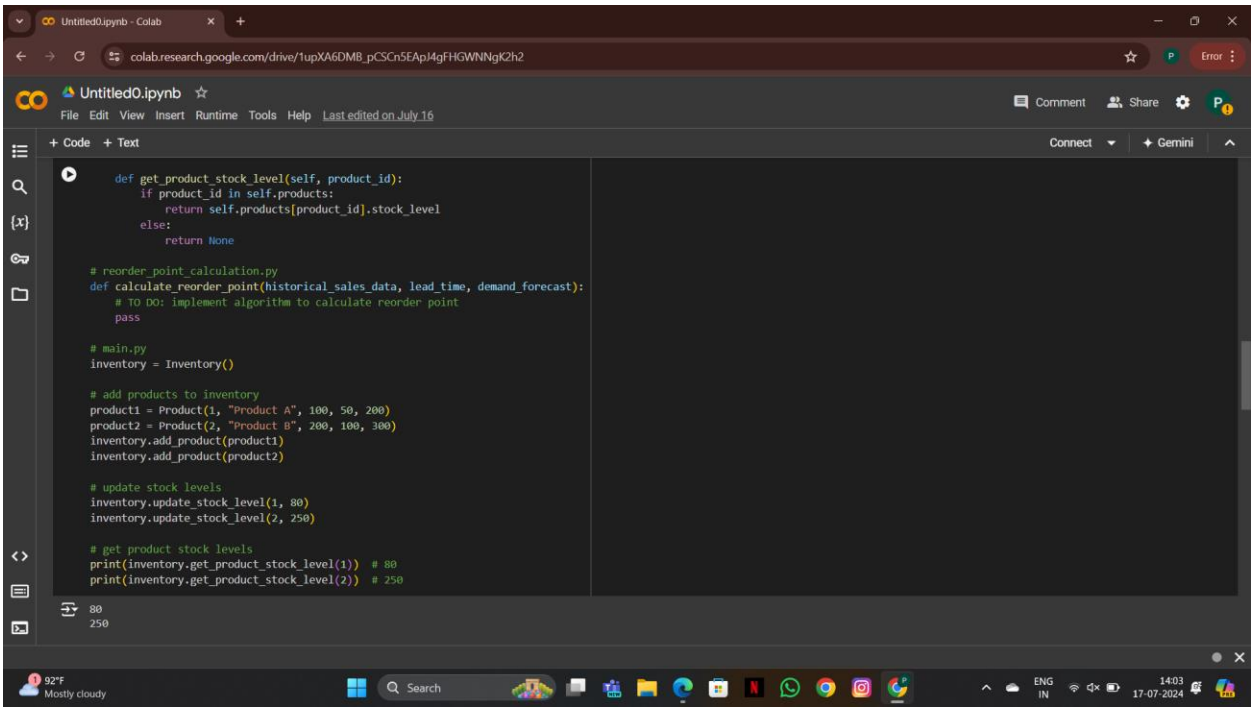
```
# get product stock levels
print(inventory.get_product_stock_level(1))  # 80
print(inventory.get_product_stock_level(2))  # 250
```

OUTPUT:

```
inventory.update_stock_level(1) 80
inventory.update_stock_level(2) 250
```

USER INPUT





Documentation:

1. Algorithms: Reorder point calculation based on lead time and daily demand.
2. Methods: The Inventory Management class handles product addition, stock updates, reorder point calculation, and report generation.
3. Assumptions: Constant lead times and daily demand.
4. Improvements: More complex forecasting algorithms, integration with sales systems for automatic updates.

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.

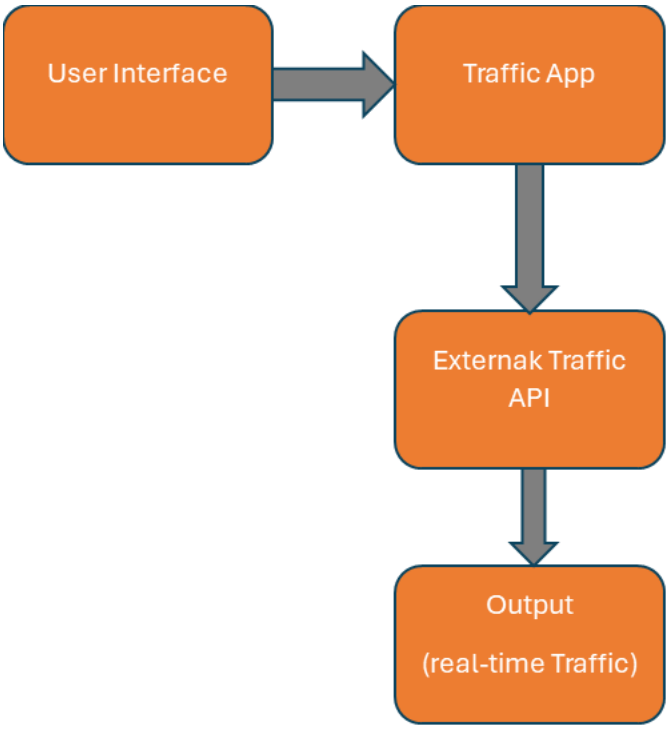
• Documentation of the API integration and the methods used to fetch and display traffic data.

• Explanation of any assumptions made and potential improvements

Solution:

Real-Time Traffic Monitoring System

DATA FLOW DIAGRAM



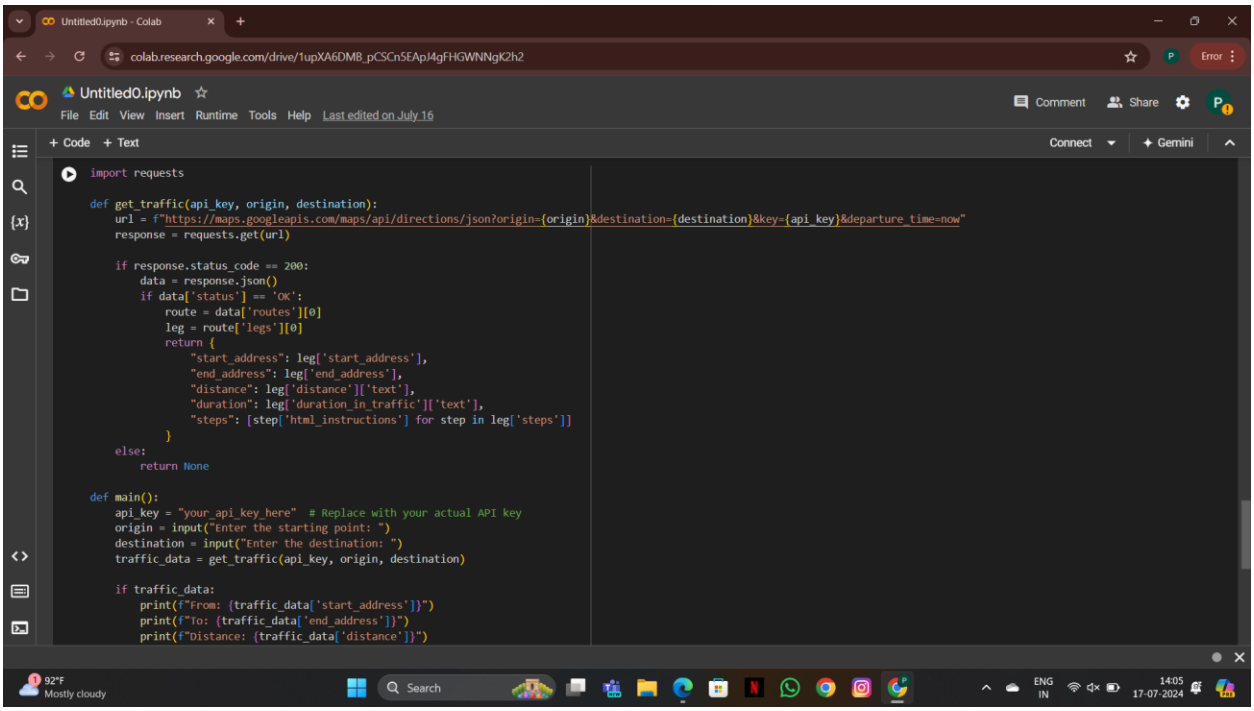
CODE

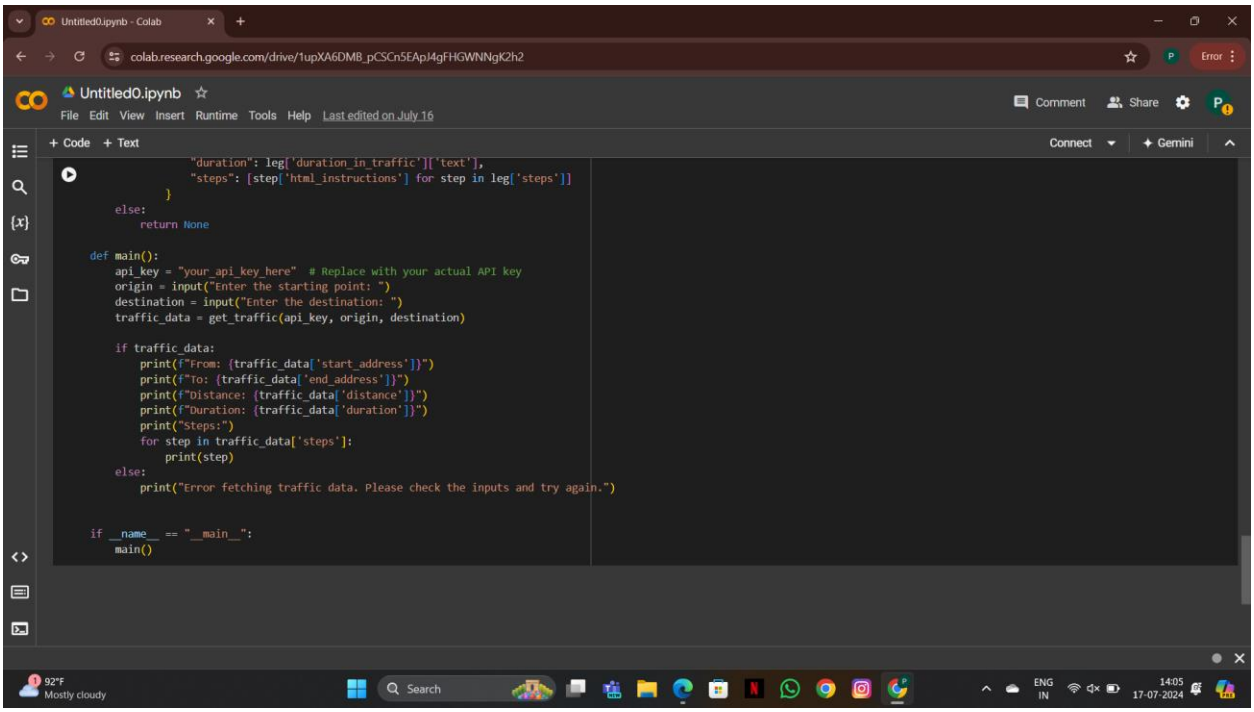
```
import requests

def get_traffic(api_key, origin, destination):
    url = f"https://maps.googleapis.com/maps/api/directions/json?origin={origin}&destination={destination}&key={api_key}&departure_time=now"
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        if data['status'] == 'OK':
            route = data['routes'][0]
            leg = route['legs'][0]
            return {
                "start_address": leg['start_address'],
                "end_address": leg['end_address'],
                "distance": leg['distance']['text'],
                "duration": leg['duration_in_traffic']['text'],
                "steps": [step['html_instructions'] for step in leg['steps']]
            }
    else:
```

```
        return None

def main():
    api_key = "your_api_key_here"  # Replace with your actual API key
    origin = input("Enter the starting point: ")
    destination = input("Enter the destination: ")
    traffic_data = get_traffic(api_key, origin, destination)

    if traffic_data:
        print(f"From: {traffic_data['start_address']}")
        print(f"To: {traffic_data['end_address']}")
        print(f"Distance: {traffic_data['distance']}")
        print(f"Duration: {traffic_data['duration']}")
        print("Steps:")
        for step in traffic_data['steps']:
            print(step)
    else:
        print("Error fetching traffic data. Please check the inputs and try
again.")


if __name__ == "__main__":
    main()
```

OUTPUT:

USER INPUT





Documentation:

 API Integration: Using Google Maps Traffic API for real-time traffic data.

 Methods: The get traffic function handles API requests and response processing. The main function manages user input and displays traffic updates.

 Assumptions: Valid addresses provided by the user.

 Improvements: Enhance error handling, provide alternative routes, and integrate with other transportation modes.

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.

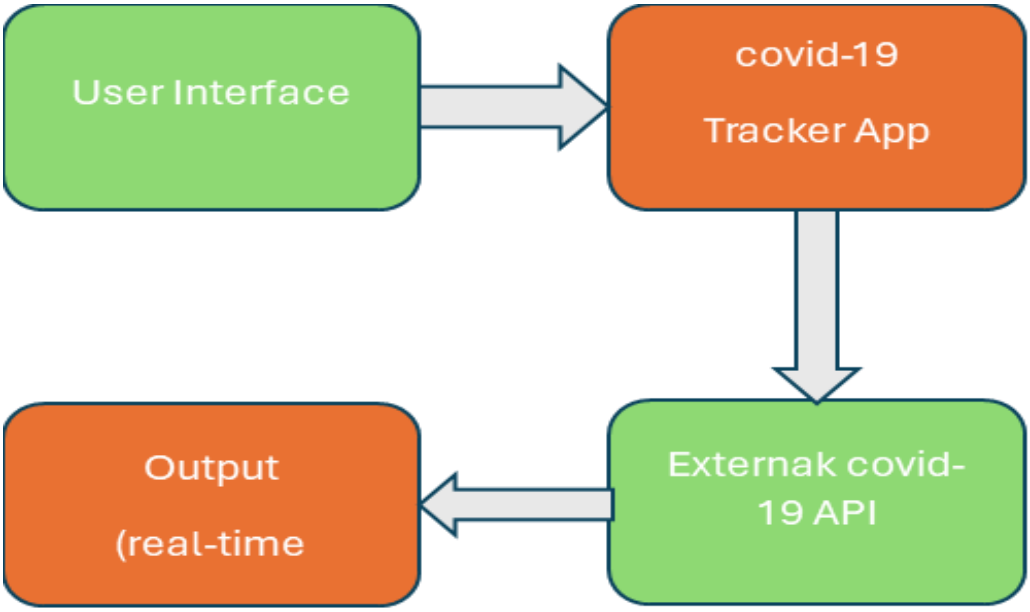
• Documentation of the API integration and the methods used to fetch and display COVID19 data.

• Explanation of any assumptions made and potential improvements.

Solution:

Real-Time COVID-19 Statistics Tracker

Data Flow Diagram:



Pseudocode:

1. Get user input for the region.
2. Send a request to the COVID-19 statistics API with the region.
3. Receive and parse the COVID-19 data from the API.
4. Display the number of cases, recoveries, and deaths for the region.

CODE

```
import requests

def get_covid_stats(region, api_key):
    url = f"https://disease.sh/v3/covid-19/countries/{region}"
    headers = {
        "Authorization": f"Bearer {api_key}"
    }
    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        data = response.json()
        return {
```

```
            "cases": data["cases"],
            "recoveries": data["recovered"],
            "deaths": data["deaths"]
        }
    else:
        return None

def main():
    region = input("Enter the country name: ")
    api_key = "dc4d2e2f2bmshe1e80669720aef1p180707jsnc79d6e5283d4"
    covid_data = get_covid_stats(region, api_key)

    if covid_data:
        print(f"COVID-19 Statistics for {region}:")
        print(f"Total Cases: {covid_data['cases']}")
        print(f"Recoveries: {covid_data['recoveries']}")
        print(f"Deaths: {covid_data['deaths']}")
    else:
        print("Error fetching COVID-19 data. Please check the region name and try
again.")

if __name__ == "__main__":
    main()
```

OUTPUT:

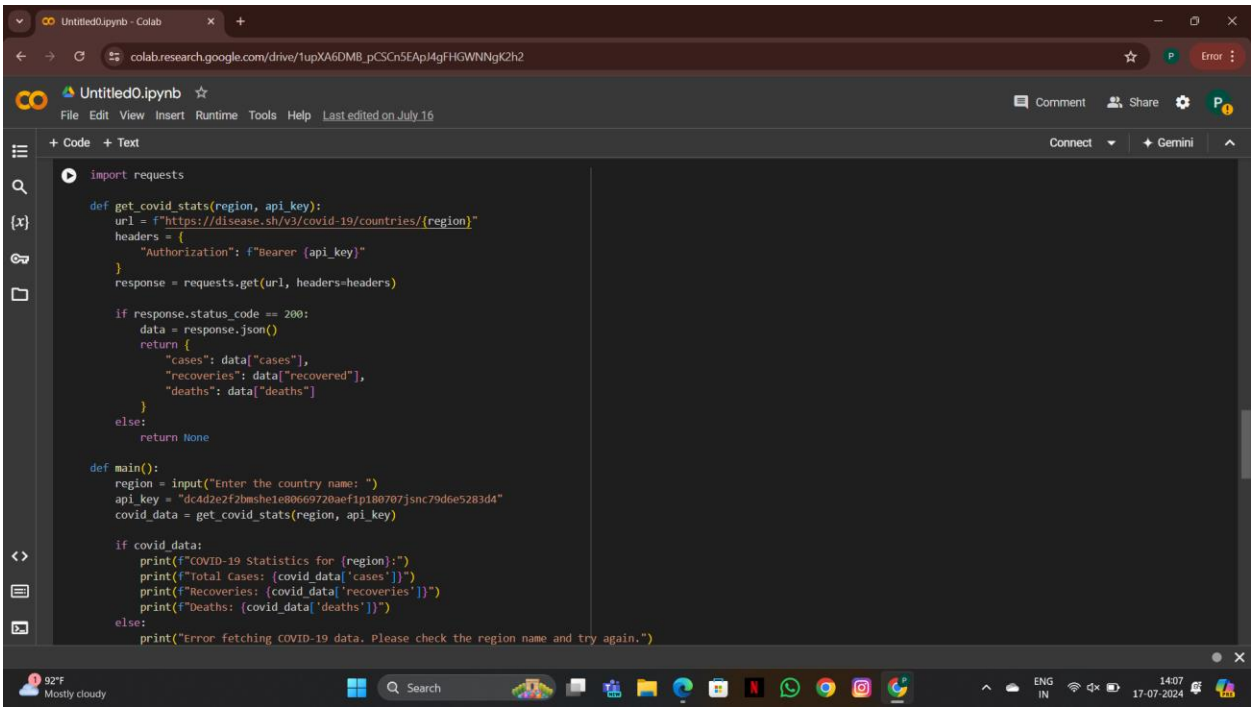
Enter the country name: china COVID-19

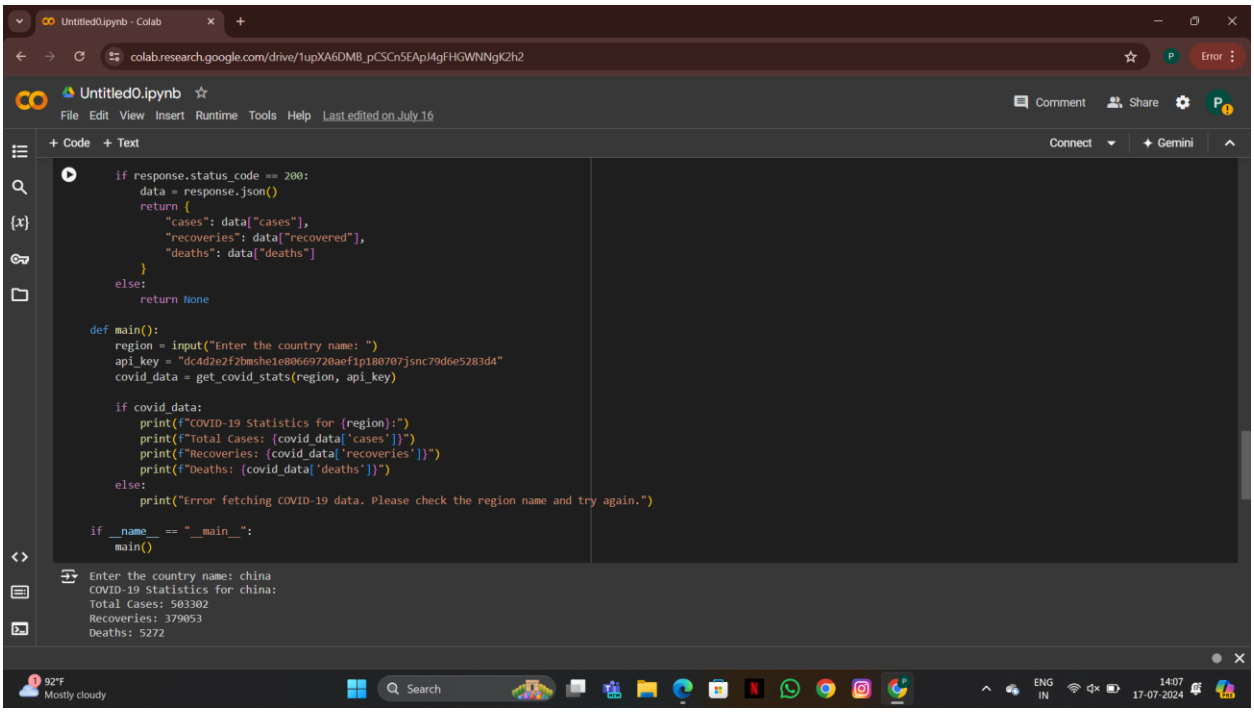
Statistics for china: Total Cases: 503302

Recoveries: 379053

Deaths: 5272

USER INPUT





Documentation:

5. API Integration: Using disease.sh API for real-time COVID-19 statistics.
6. Methods: The get _covid _stats function handles the API request and response. The main function manages user input and displays statistics.
7. Assumptions: The user provides a valid country name.

8. Improvements: Enhance error handling, provide historical data, and integrate with vaccination statistics.