NAVYA SREE.VAARTHALA

192373050

CSE(DATA SCIENCE)

PYTHON API PROGRAMS DOCUMENTATION

DATE :16/7/2024

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.

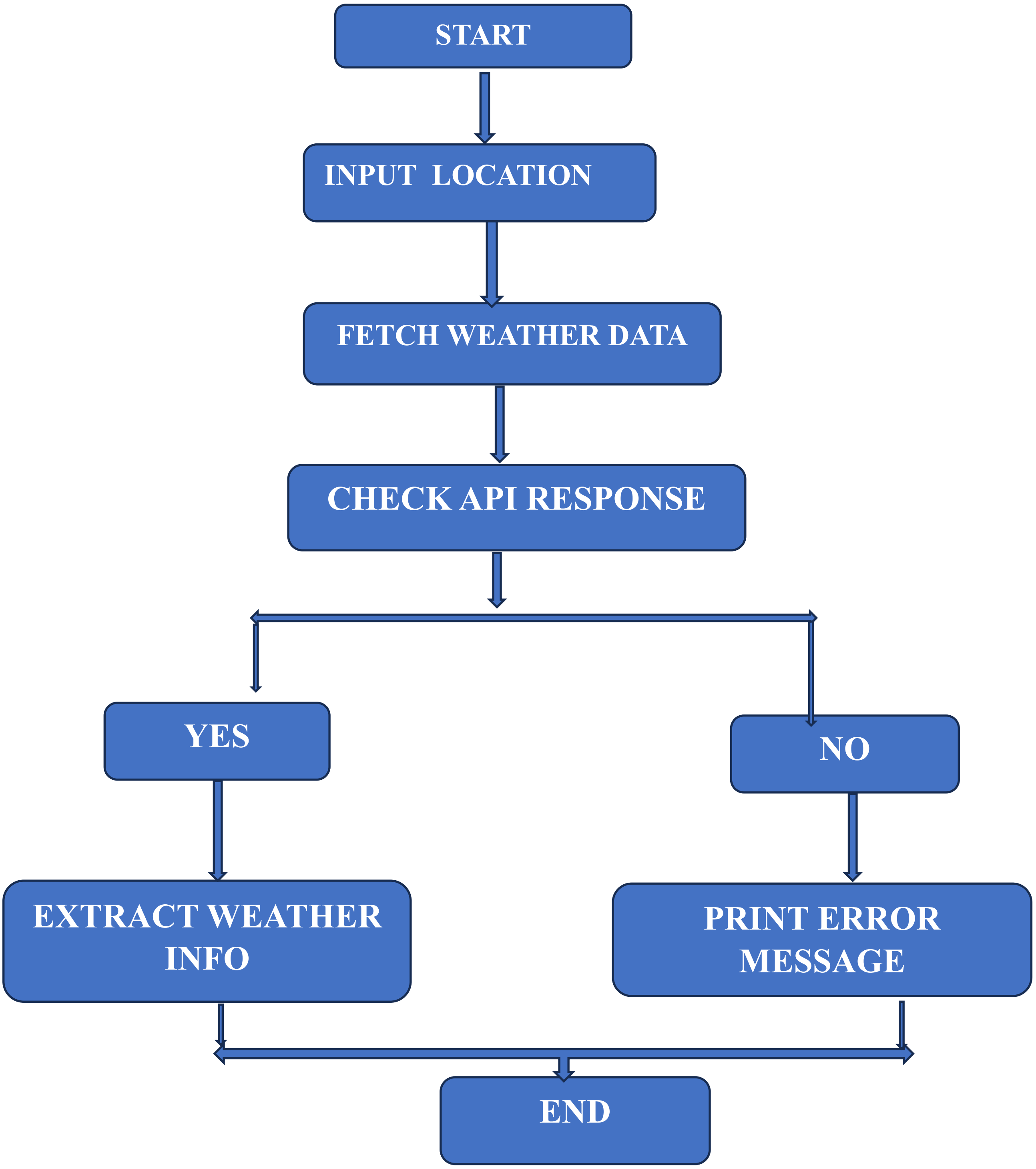
Tasks:

- Model the data flow for fetching weather information from an external API and displaying it to the user.
- Implement a Python application that integrates with a weather API (e.g., Open Weather Map) to fetch real-time weather data.
- Display the current weather information, including temperature, weather conditions, humidity, and wind speed.
- 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.

FLOW CHART:



IMPLEMENTATION:

```
import requests
def fetch_weather_data(api_key, location):
    base_url ="https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid"

    params = {
        'q': location,
        'appid': api_key,
        'units': 'metric'
    }
    try:
        response = requests.get(base_url, params=params)
        data = response.json()
        if data["cod"] == 200:
            weather_info = {
                'location':data['name'],
                'temperature': data['main']['temp'],
                'weather': data['weather'][0]['description'],
                'humidity': data['main']['humidity'],
                'wind_speed': data['wind']['speed']
            }
            return weather_info
        else:
            return None
    except Exception as e:
        print(f"Error fetching weather data: {e}")
        return None
def display_weather(weather_info, location):
    if weather_info:
        print(f"Weather in {location}:")
        print(f"Temperature: {weather_info['temperature']} °C")
        print(f"Weather: {weather_info['weather']}")
        print(f"Humidity: {weather_info['humidity']}%")
        print(f"Wind Speed: {weather_info['wind_speed']} m/s")
    else:
        print(f"Failed to fetch weather data for {location}")
def main():
    api_key = "ed7c18d0f1024da78bf89f147ccd9bca"
    location = input("Enter city name or coordinates (latitude,longitude): ")
    weather_info = fetch_weather_data(api_key, location)
    display_weather(weather_info, location)
if __name__ == "__main__":
    main()
```

DISPLAYING DATA:

INPUT: Enter city name or coordinates (latitude, longitude): Chennai

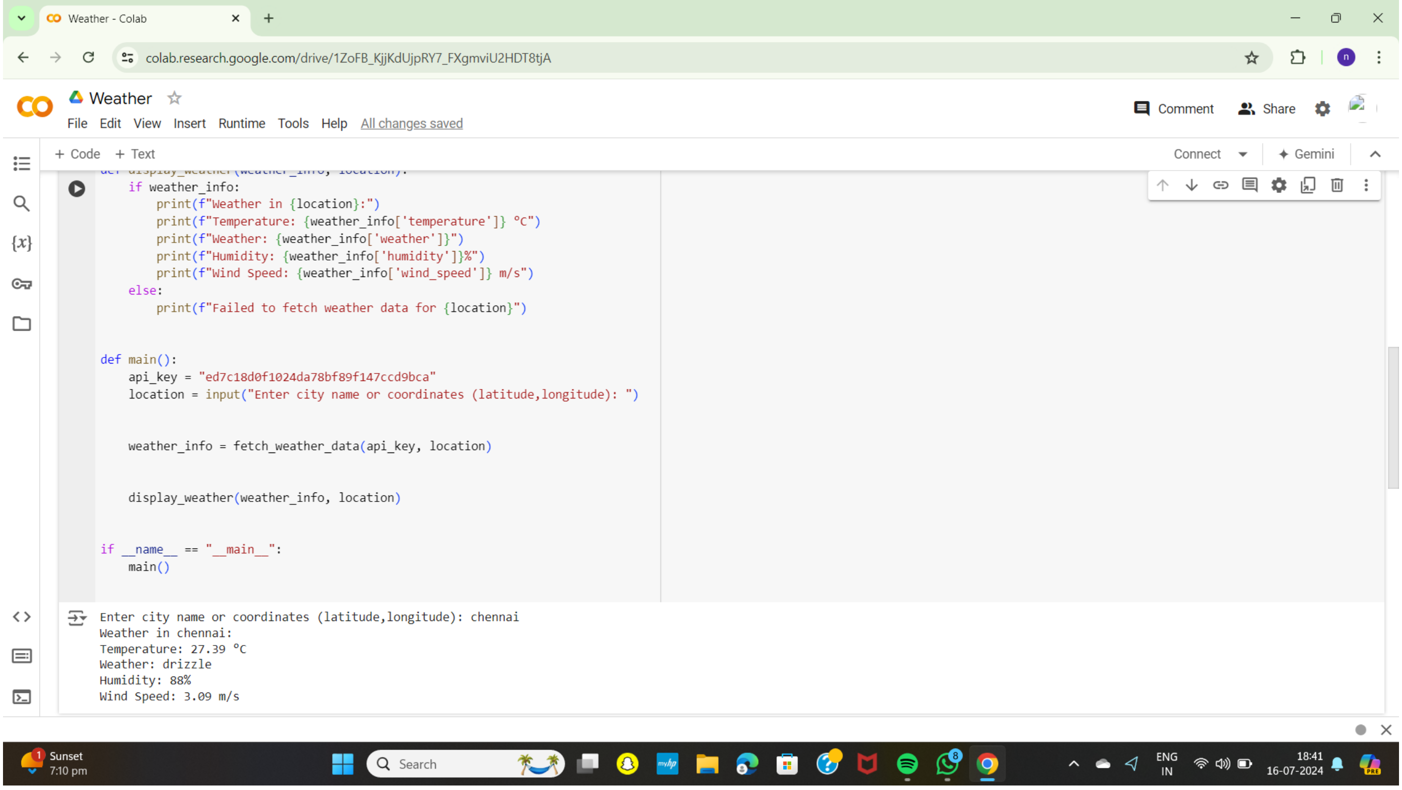
OUTPUT: Weather in Chennai:

Temperature: 27.39 °C

Weather: drizzle

Humidity: 88%

Wind Speed: 3.09 m/s.


Weather - Colab
colab.research.google.com/drive/1ZoFB_KjjKdUjpRY7_FXgmviU2HDT8tjA
Weather
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Gemini
Sunset
7:10 pm
Search
ENG
IN
18:41
16-07-2024

```
def display_weather(weather_info, location):
    if weather_info:
        print(f"Weather in {location}:")
        print(f"Temperature: {weather_info['temperature']} °C")
        print(f"Weather: {weather_info['weather']}")
        print(f"Humidity: {weather_info['humidity']}%")
        print(f"Wind Speed: {weather_info['wind_speed']} m/s")
    else:
        print(f"Failed to fetch weather data for {location}")


def main():
    api_key = "ed7c18d0f1024da78bf89f147ccd9bca"
    location = input("Enter city name or coordinates (latitude,longitude): ")


    weather_info = fetch_weather_data(api_key, location)


    display_weather(weather_info, location)


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

```
Enter city name or coordinates (latitude,longitude): chennai
Weather in chennai:
Temperature: 27.39 °C
Weather: drizzle
Humidity: 88%
Wind Speed: 3.09 m/s
```

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:

- Model the inventory system: Define the structure of the inventory system, including products, warehouses, and current stock levels.
- 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.
- Optimize inventory ordering: Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
- Generate reports: Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
- User interaction: Allow users to input product IDs or names to view current stock levels,reorder recommendations, and historical data.

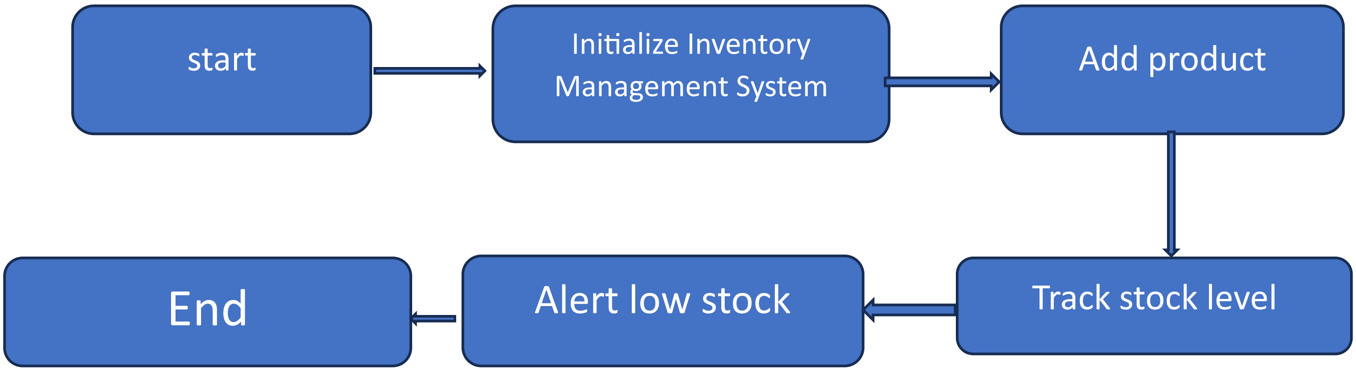
Deliverables:

- 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.

FLOW CHART:



IMPLEMENTATION:

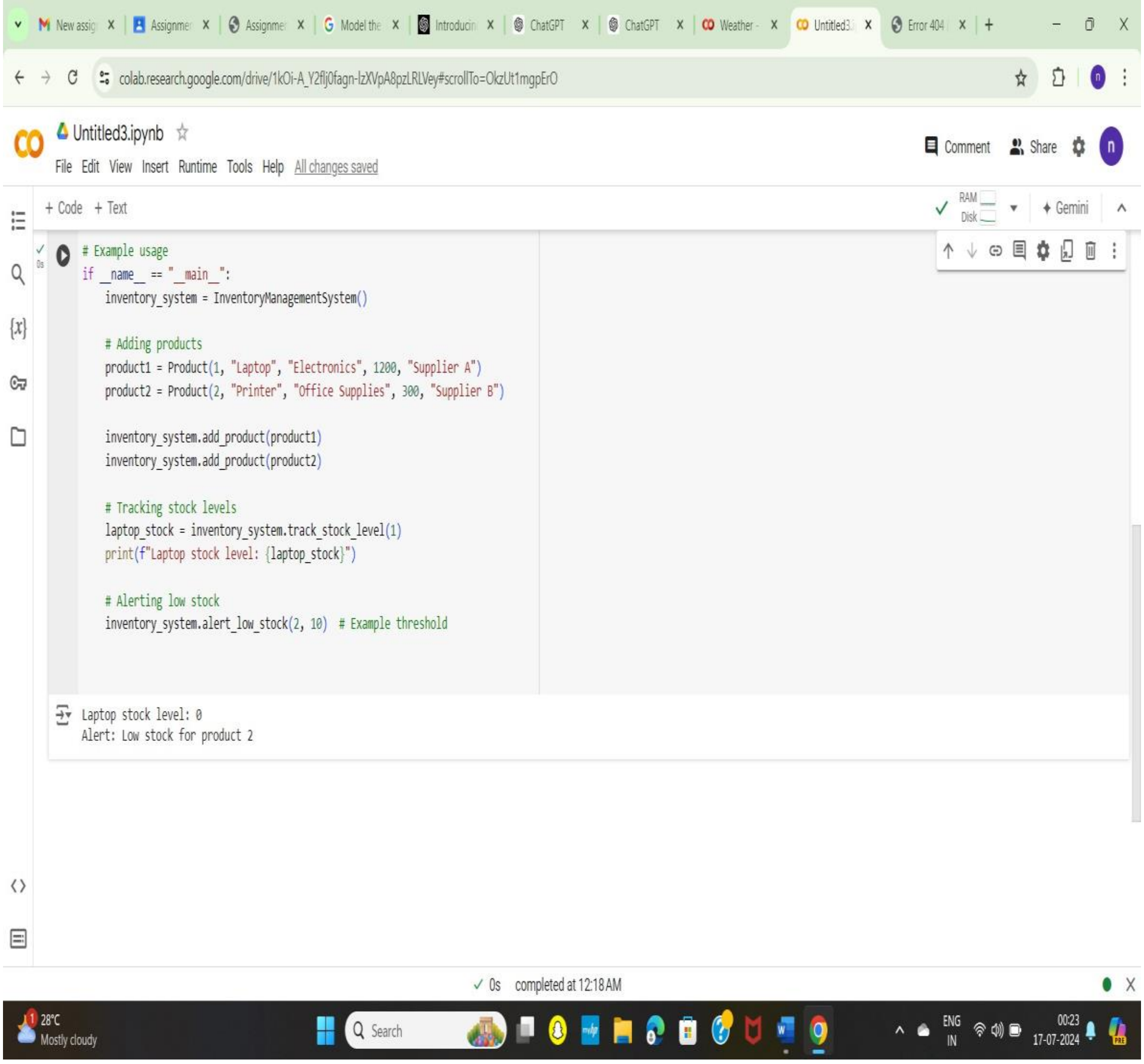
```
class Product:
    def __init__(self, id, name, category, price, supplier):
        self.id = id
        self.name = name
        self.category = category
        self.price = price
        self.supplier = supplier
        self.stock_level = 0

    def update_stock(self, quantity):
        self.stock_level += quantity

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

    def add_product(self, product):
        self.products[product.id] = product
    def track_stock_level(self, product_id):
        return self.products.get(product_id, None).stock_level if product_id in self.products else None

    def alert_low_stock(self, product_id, threshold):
        if product_id in self.products:
            if self.products[product_id].stock_level < threshold:
                print(f"Alert: Low stock for product {product_id}")

# Example usage
if __name__ == "__main__":
    inventory_system = InventoryManagementSystem()

    # Adding products
    product1 = Product(1, "Laptop", "Electronics", 1200, "Supplier A")
    product2 = Product(2, "Printer", "Office Supplies", 300, "Supplier B")

    inventory_system.add_product(product1)
    inventory_system.add_product(product2)

    # Tracking stock levels
    laptop_stock = inventory_system.track_stock_level(1)
    print(f"Laptop stock level: {laptop_stock}")

    # Alerting low stock
    inventory_system.alert_low_stock(2, 10)  # Example threshold
```

DISPLAYING DATA:

OUTPUT:

```
Laptop stock level: 0
Alert: Low stock for product 2
```

USER INPUT


colab.research.google.com/drive/1kOi-A_Y2flj0fagn-lzXVpA8pzLRLVey#scrollTo=OkzUt1mgpEr0
Untitled3.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
✓ 0s completed at 12:18 AM

```
# Example usage
if __name__ == "__main__":
    inventory_system = InventoryManagementSystem()

    # Adding products
    product1 = Product(1, "Laptop", "Electronics", 1200, "Supplier A")
    product2 = Product(2, "Printer", "Office Supplies", 300, "Supplier B")

    inventory_system.add_product(product1)
    inventory_system.add_product(product2)

    # Tracking stock levels
    laptop_stock = inventory_system.track_stock_level(1)
    print(f"Laptop stock level: {laptop_stock}")

    # Alerting low stock
    inventory_system.alert_low_stock(2, 10)  # Example threshold
```

```
Laptop stock level: 0
Alert: Low stock for product 2
```

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:

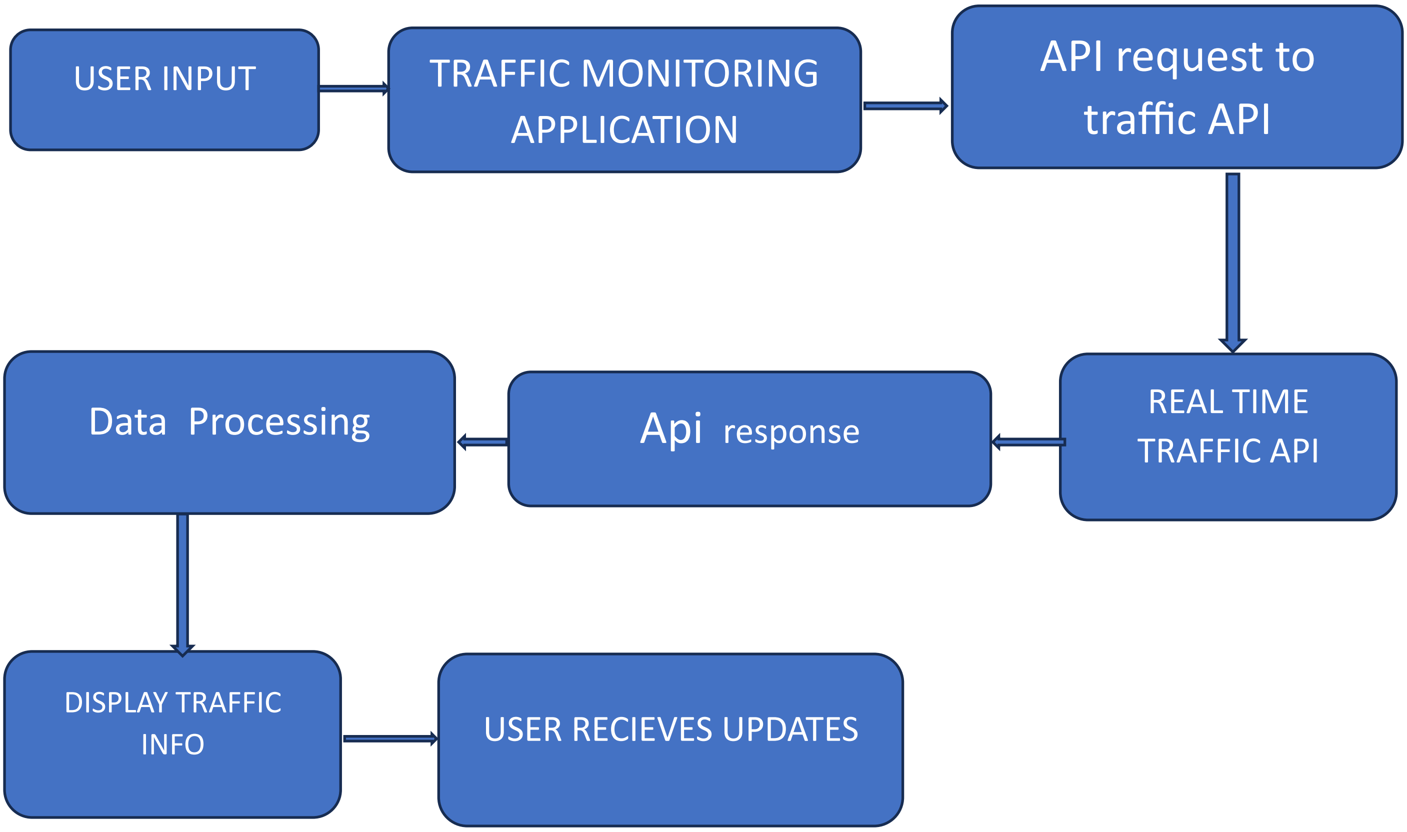
- Model the data flow for fetching real-time traffic information from an external API and displaying it to the user.
- 2. Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.
- 3. Display current traffic conditions, 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.

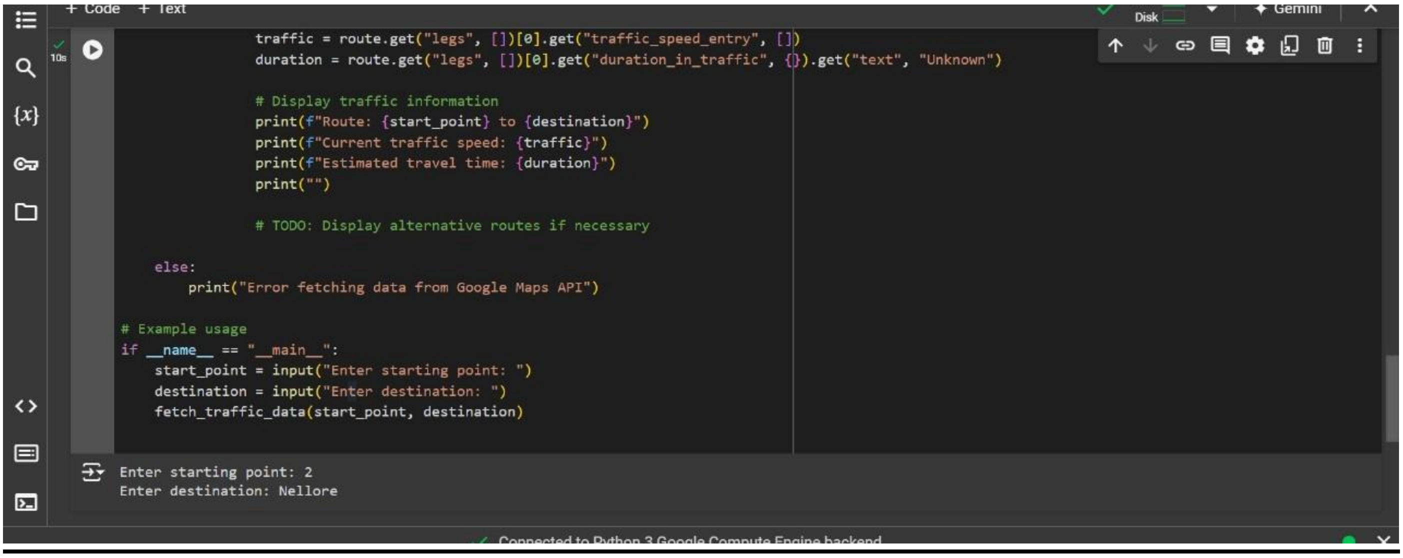
FLOW CHART:



IMPLEMENTATION:

```
import requests
API_KEY = "your_api_key_here"
API_ENDPOINT = "https://maps.googleapis.com/maps/api/directions/json"
def fetch_traffic_data(origin, destination):
    url = f"{API_ENDPOINT}?origin={origin}&destination={destination}&key={API_KEY}"
    response = requests.get(url)
    if response.status_code == 200:
        traffic_data = response.json()
        return traffic_data
    else:
        return None
def display_traffic_info(traffic_data):
    if traffic_data is not None:
        routes = traffic_data.get("routes", [])
        if routes:
            legs = routes[0].get("legs", [])
            if legs:
                duration_in_traffic = legs[0].get("duration_in_traffic", {}).get("text", "Not available")
                print(f"Estimated duration in traffic: {duration_in_traffic}")
                current_speed = legs[0].get("traffic_speed_entry", {}).get("speed_kph", "Not available")
                print(f"Current speed: {current_speed} km/h")
                incidents = legs[0].get("traffic", {}).get("incidents", [])
                if incidents:
                    print("Incidents:")
                    for incident in incidents:
                        incident_type = incident.get("type", "Unknown")
                        incident_description = incident.get("description", "No description")
                        print(f"- {incident_type}: {incident_description}")
                else:
                    print("No incidents reported.")
        else:
            print("No routes found.")
    else:
        print("Failed to fetch traffic data.")


def suggest_alternative_routes(traffic_data):
    if traffic_data is not None:
        routes = traffic_data.get("routes", [])
        if len(routes) > 1:
            print("Alternative routes:")
            for i in range(1, len(routes)):
                route_summary = routes[i].get("summary", "Route without summary")
                route_duration = routes[i].get("legs", [{}])[0].get("duration", {}).get("text", "Not available")
                print(f"- Route {i}: {route_summary}, Estimated duration: {route_duration}")
        else:
            print("No alternative routes available.")
    else:
        print("Failed to fetch alternative routes.")

if __name__ == "__main__":
    origin = input("Enter starting point: ")
    destination = input("Enter destination: ")

    traffic_data = fetch_traffic_data(origin, destination)

    if traffic_data is not None:
        display_traffic_info(traffic_data)
        suggest_alternative_routes(traffic_data)
    else:
        print("Failed to fetch traffic information. Please try again.")
   if __name__ == "__main__":
main()
```

OUTPUT:



```
                traffic = route.get("legs", [])[0].get("traffic_speed_entry", [])
                duration = route.get("legs", [])[0].get("duration_in_traffic", {}).get("text", "Unknown")

                # Display traffic information
                print(f"Route: {start_point} to {destination}")
                print(f"Current traffic speed: {traffic}")
                print(f"Estimated travel time: {duration}")
                print("")

                # TODO: Display alternative routes if necessary

    else:
        print("Error fetching data from Google Maps API")

# Example usage
if __name__ == "__main__":
    start_point = input("Enter starting point: ")
    destination = input("Enter destination: ")
    fetch_traffic_data(start_point, destination)
```

```
Enter starting point: 2
Enter destination: Nellore
```

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.

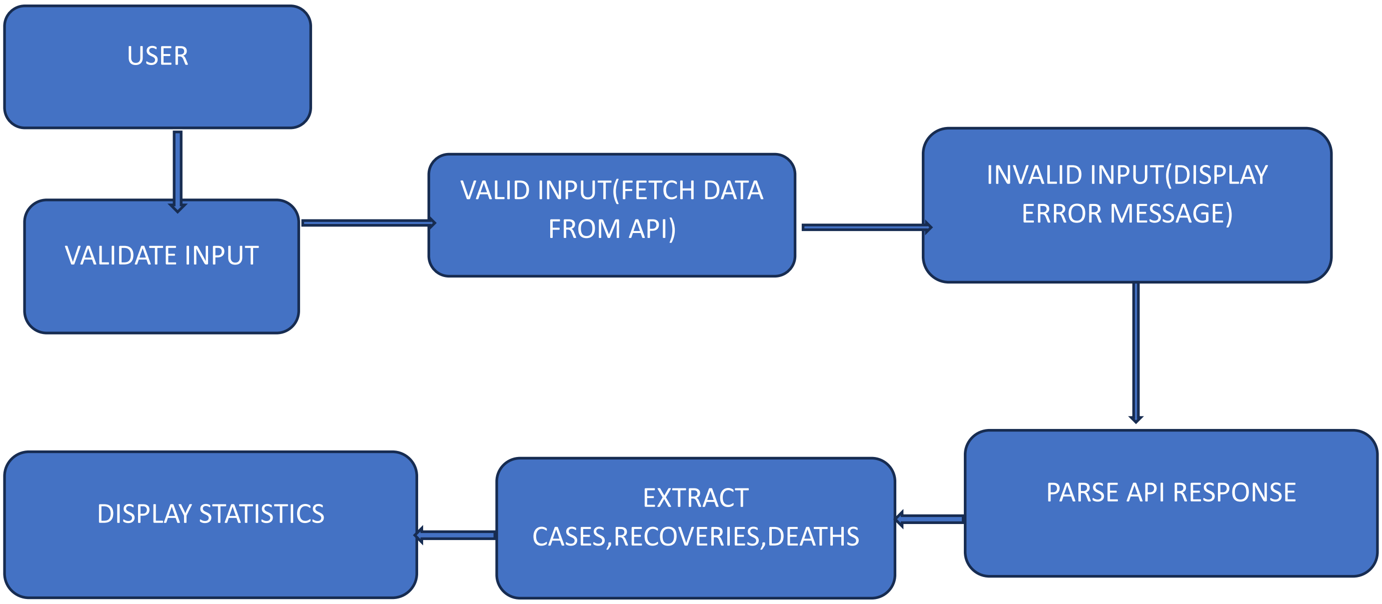
Tasks:

- Model the data flow for fetching COVID-19 statistics from an external API and displaying it to the user.
- Implement a Python application that integrates with a COVID-19 statistics API (e.g., disease.sh) to fetch real-time data.
- Display the current number of cases, recoveries, and deaths for a specified region.
- 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

FLOW CHART:


USER
VALIDATE INPUT
VALID INPUT(FETCH DATA FROM API)
INVALID INPUT(DISPLAY ERROR MESSAGE)
PARSE API RESPONSE
EXTRACT CASES,RECOVERIES,DEATHS
DISPLAY STATISTICS

IMPLEMENTATION:

```
mport requests
API_ENDPOINT = "https://disease.sh/v3/covid-19"
def fetch_covid_stats(region):
response = requests.get(f"{API_ENDPOINT}/all?region={region}")
return response.json()
def display_data(data):
print(f"Region: {data.get('country', 'N/A')}")
print(f"Cases: {data.get('cases', 'N/A')}")
print(f"Recoveries: {data.get('recovered', 'N/A')}")
print(f"Deaths: {data.get('deaths', 'N/A')}")
def main():
region = input("Enter the region (country, state, or or city): ")
covid_data = fetch_covid_stats(region)
display_data(covid_data)
if __name__ == "__main__":
```

```
main()
```

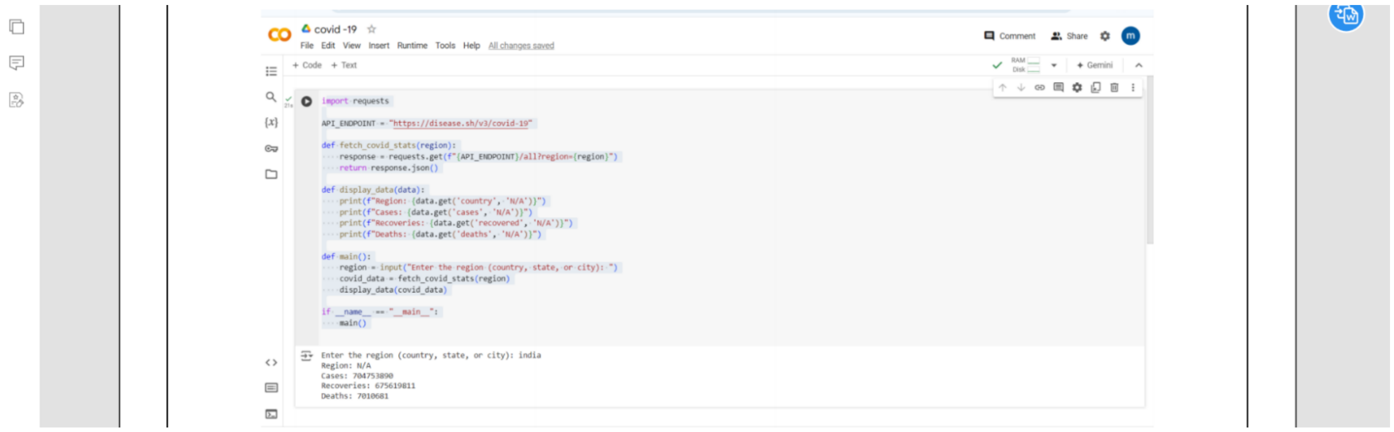
Displaying data:

Input:

```
Enter the region (country, state, or city): india
```

Output:

```
Region: N/A

Cases: 704753890

Recoveries: 675619811

Deaths: 7010681
```



```
import requests

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

def fetch_covid_stats(region):
    response = requests.get(f"{API_ENDPOINT}/all?region={region}")
    return response.json()

def display_data(data):
    print(f"Region: {data.get('country', 'N/A')}")
    print(f"Cases: {data.get('cases', 'N/A')}")
    print(f"Recoveries: {data.get('recovered', 'N/A')}")
    print(f"Deaths: {data.get('deaths', 'N/A')}")

def main():
    region = input("Enter the region (country, state, or city): ")
    covid_data = fetch_covid_stats(region)
    display_data(covid_data)

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

```
Enter the region (country, state, or city): india
Region: N/A
Cases: 704753890
Recoveries: 675619811
Deaths: 7010681
```