

PUDIPARTHI. VINEETHA

192373053

CSE [DATA SCIENCE]

PYTHON API PROGRAMMING DOCUMENTATION

DATE: 16-07-2024

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:

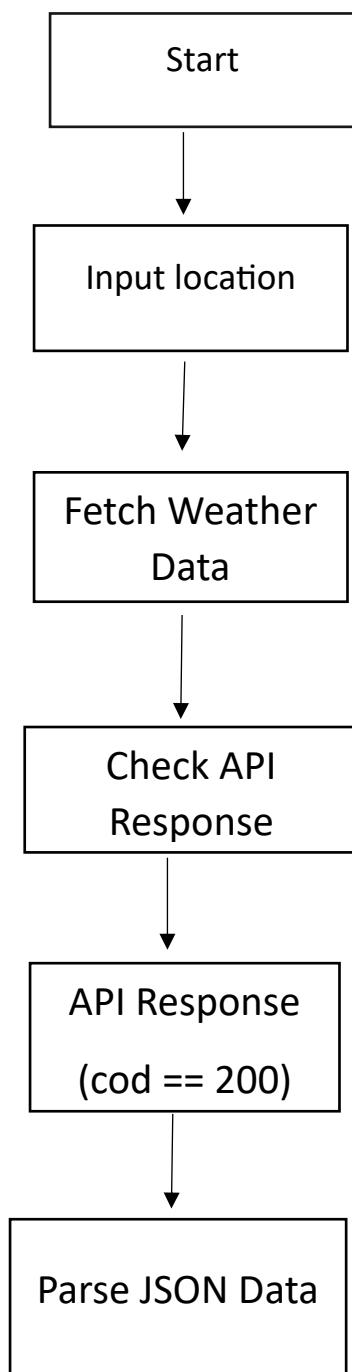
- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the weather monitoring system.

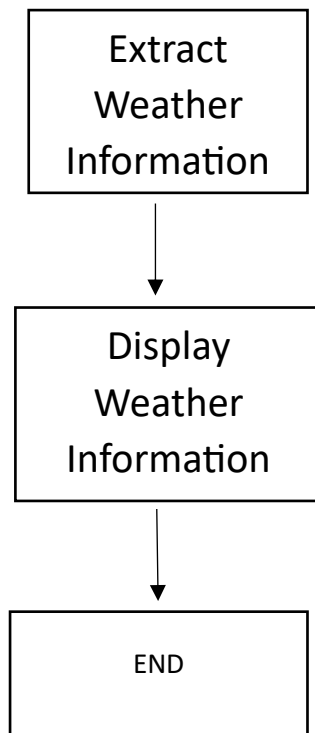
- Documentation of the API integration and the methods used to fetch and display

weather data.

- Explanation of any assumptions made and potential improvements.

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)

```

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

DISPLAYING THE DATA:

Enter city name or coordinates (latitude,longitude): Nellore

Weather in Nellore:

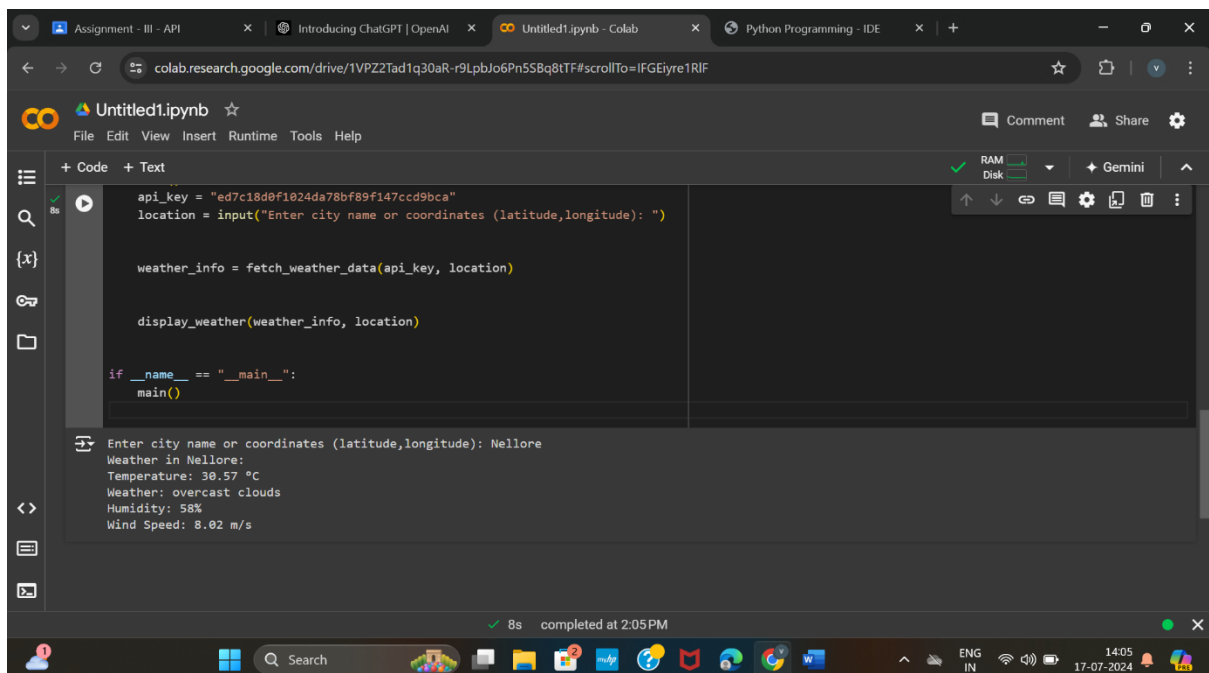
Temperature: 30.57 °C

Weather: overcast clouds

Humidity: 58%

Wind Speed: 8.02 m/s

OUTPUT:



The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'Assignment - III - API', 'Introducing ChatGPT | OpenAI', 'Untitled1.ipynb - Colab', and 'Python Programming - IDE'. The address bar shows the Colab URL. The notebook title is 'Untitled1.ipynb'. The left sidebar contains icons for file explorer, search, and other tools. The main area displays a code cell with the following Python code:

```
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()
```

Below the code cell, the output is displayed:

```
Enter city name or coordinates (latitude,longitude): Nellore  
Weather in Nellore:  
Temperature: 30.57 °C  
Weather: overcast clouds  
Humidity: 58%  
Wind Speed: 8.02 m/s
```

The bottom status bar indicates '8s completed at 2:05 PM'. The Windows taskbar at the very bottom shows the date as 17-07-2024 and time as 14:05.

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).

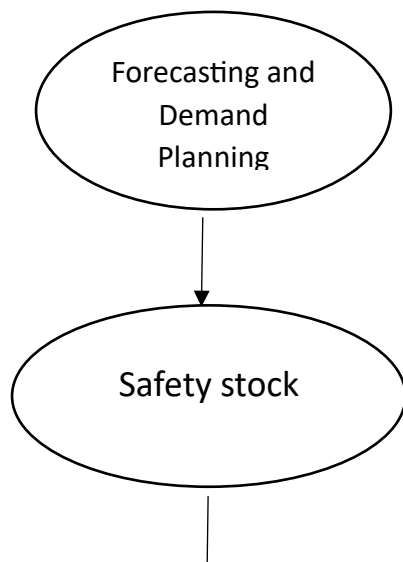
- **User Interface:** Develop a user-friendly interface for accessing inventory information,

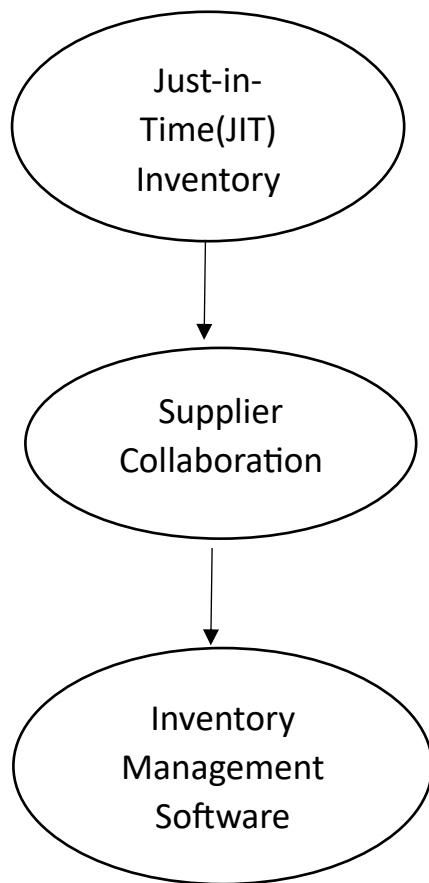
viewing reports, and receiving alerts.

- **Assumptions and Improvements:** Discuss assumptions about demand patterns, supplier

reliability, and potential improvements for the inventory management system's efficiency and accuracy.

FLOW CHART:





IMPLEMENTATION:

class Product:

```
def _init_(self, product_id, name, description, category, supplier):
```

```
    self.product_id = product_id
```

```
    self.name = name
```

```
    self.description = description
```

```
    self.category = category
```

```
    self.supplier = supplier
```

```
    self.stock = 0
```

```
def update_stock(self, quantity):
```

```
    self.stock += quantity
```

```
def check_stock(self, threshold):
```



```

        if self.stock <= threshold:
            return True
        return False
class InventorySystem:
    def __init__(self):
        self.products = {}
    def add_product(self, product):
        self.products[product.product_id] = product
    def update_inventory(self, product_id, quantity):
        if product_id in self.products:
            self.products[product_id].update_stock(quantity)
        else:
            print(f"Product with ID {product_id} not found.")
    def check_stock_levels(self, threshold):
        low_stock_products = []
        for product_id, product in self.products.items():
            if product.check_stock(threshold):
                low_stock_products.append(product)
        return low_stock_products
# Example usage
inventory_system = InventorySystem()
# Adding products
product1 = Product("P1", "Product 1", "Description of Product 1", "Category A",
"Supplier A")
product2 = Product("P2", "Product 2", "Description of Product 2", "Category B",
"Supplier B")
inventory_system.add_product(product1)

```

```
inventory_system.add_product(product2)
```

```
# Simulate sales and updates
```

```
inventory_system.update_inventory("P1", -5) # Sold 5 units of Product 1
```

```
inventory_system.update_inventory("P2", -3) # Sold 3 units of Product 2
```

```
# Check for low stock products
```

```
low_stock_threshold = 10
```

```
low_stock_products =
```

```
inventory_system.check_stock_levels(low_stock_threshold)
```

```
for product in low_stock_products:
```

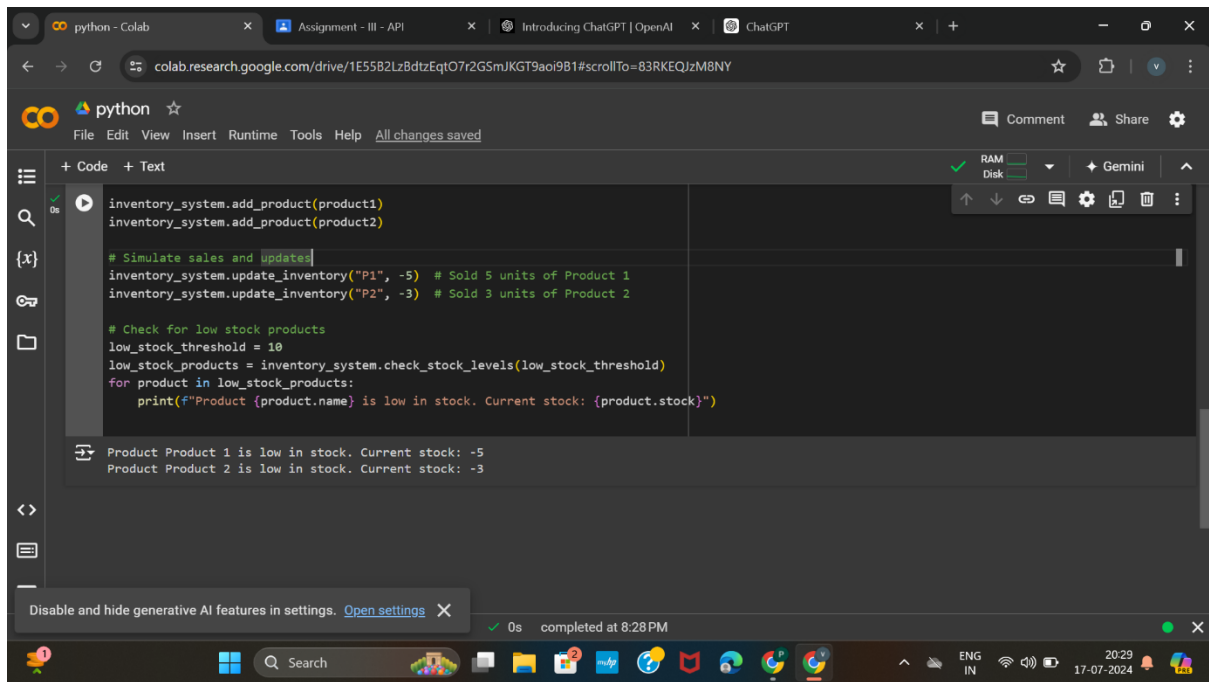
```
    print(f"Product {product.name} is low in stock. Current stock:  
    {product.stock}")
```

DISPLAY THE DATA:

Product Product 1 is low in stock. Current stock: -5

Product Product 2 is low in stock. Current stock: -3

OUTPUT :



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

# Simulate sales and updates
inventory_system.update_inventory("P1", -5) # Sold 5 units of Product 1
inventory_system.update_inventory("P2", -3) # Sold 3 units of Product 2

# Check for low stock products
low_stock_threshold = 10
low_stock_products = inventory_system.check_stock_levels(low_stock_threshold)
for product in low_stock_products:
    print(f"Product {product.name} is low in stock. Current stock: {product.stock}")
```

Product Product 1 is low in stock. Current stock: -5
Product Product 2 is low in stock. Current stock: -3

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:

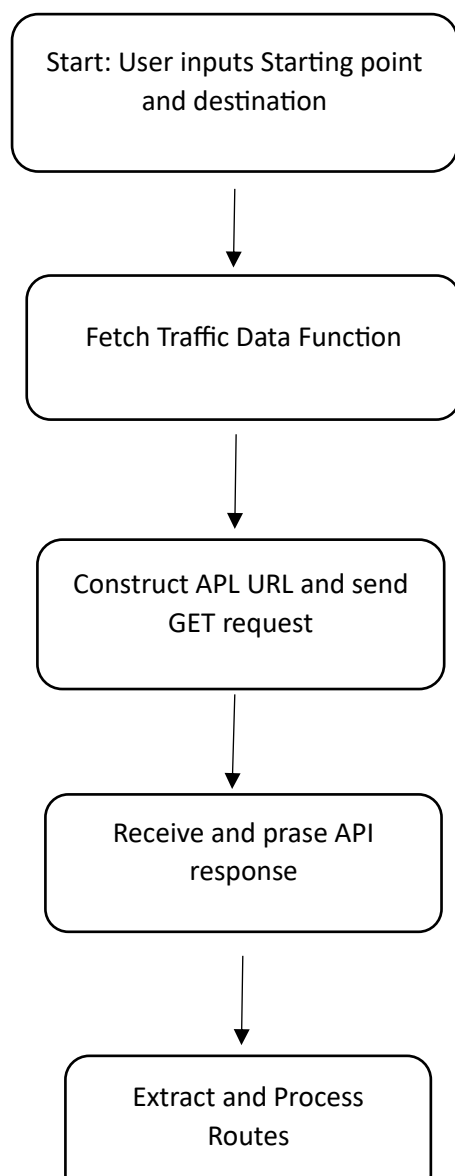
1. Model the data flow for fetching real-time traffic information from an external API and displaying it to the user.
2. Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.
3. Display current traffic conditions,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

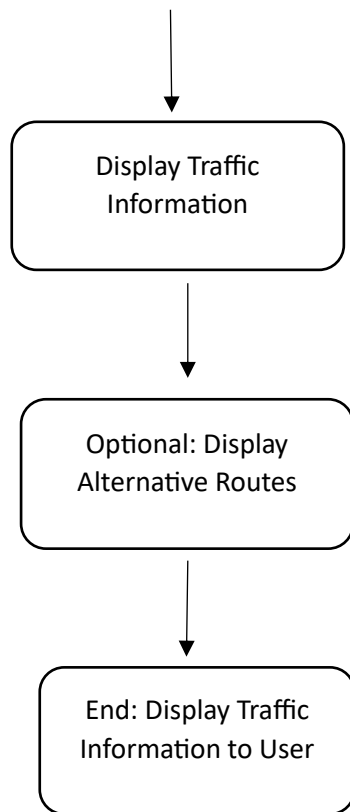
4. Allow users to input a starting point and destination to receive traffic updates and alternative routes.

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.

FLOW CHART:





IMPLEMENTATION:

```
import requests
```

```
def fetch_traffic_data(start_point, destination):  
    # Construct URL for Google Maps Traffic API request  
    url = "https://maps.googleapis.com/maps/api/directions/json"  
    params = {  
        "origin": start_point,  
        "destination": destination,  
        "key": "YOUR_API_KEY",  
        "departure_time": "now",  
        "traffic_model": "best_guess",  
    }
```

```
# Send GET request to the API
response = requests.get(url, params=params)

if response.status_code == 200:
    # Parse JSON response
    data = response.json()

    # Extract relevant traffic information
    routes = data.get("routes", [])

    if routes:
        for route in routes:
            # Get traffic information for each route
            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: ")
```

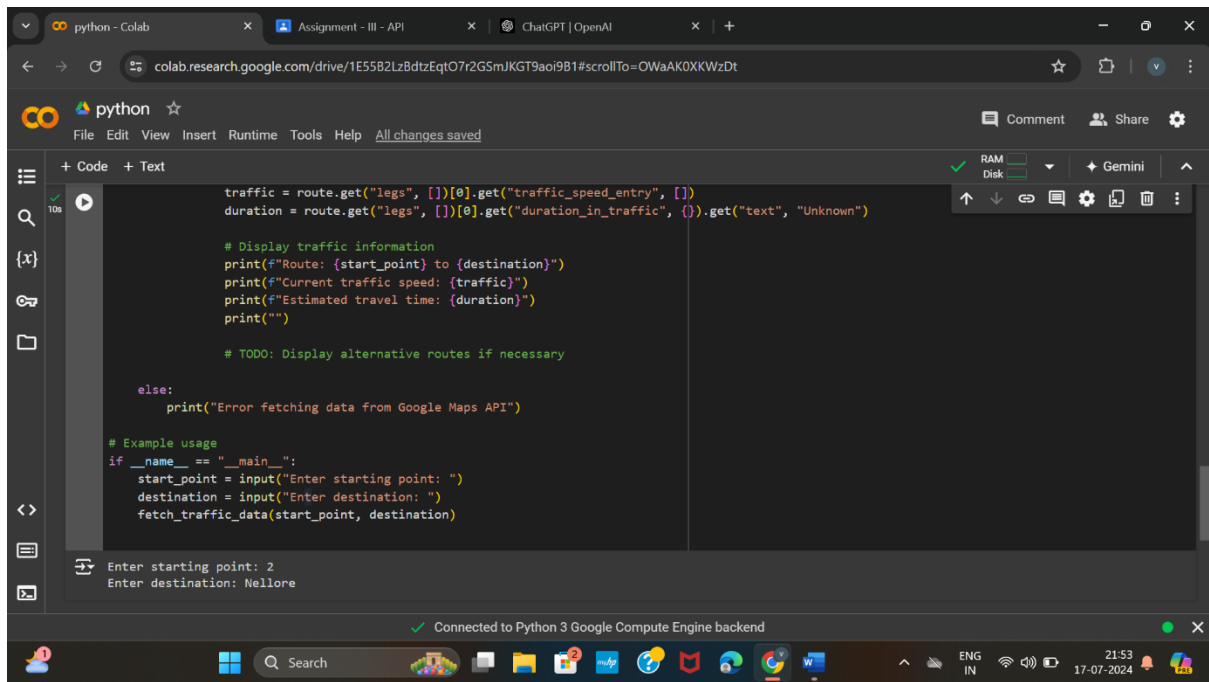
```
    fetch_traffic_data(start_point, destination)
```

DISPLAY THE DATA:

Enter starting point: 2

Enter destination: Nellore

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

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.

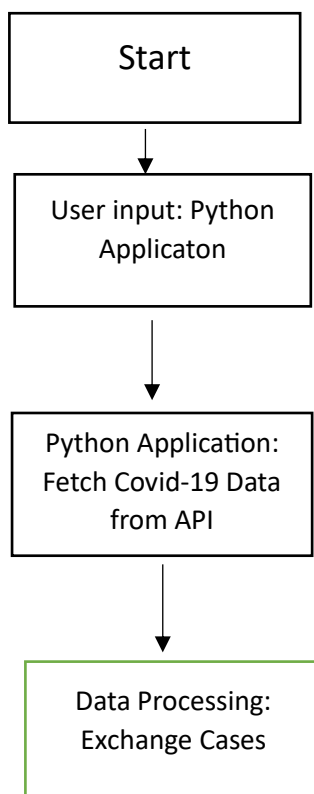
- Pseudocode and implementation of the COVID-19 statistics tracking application.

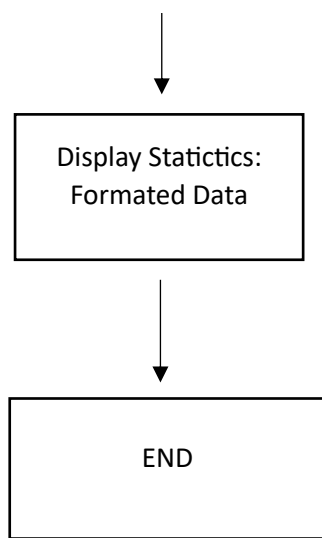
- Documentation of the API integration and the methods used to fetch and display COVID-

19 data.

- Explanation of any assumptions made and potential improvements.

FLOW CHART:





IMPLEMENTATION:

Import necessary libraries

```
import requests
```

Function to fetch COVID-19 statistics for a specified region

```
def fetch_covid_statistics(region):
```

```
    url = f"https://disease.sh/v3/covid-19/countries/{region}" # Example API endpoint
```

```
    try:
```

```
        response = requests.get(url)
```

```
        if response.status_code == 200:
```

```
            data = response.json()
```

```
            return data
```

```
        else:
```

```
            print(f"Error fetching data: {response.status_code}")
```

```
            return None
```

```
except requests.exceptions.RequestException as e:
```

```
    print(f"Request error: {e}")
```

```
    return None
```

```
# Function to display COVID-19 statistics
def display_statistics(data):
    if data:
        print(f"COVID-19 Statistics for {data['country']}:")
        print(f"Total cases: {data['cases']}")
        print(f"Total recoveries: {data['recovered']}")
        print(f"Total deaths: {data['deaths']}")
    else:
        print("No data available.")

# Main function
def main():
    region = input("Enter a country name to fetch COVID-19 statistics: ")
    data = fetch_covid_statistics(region)
    display_statistics(data)

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

DISPLAY THE DATA:

Enter a country name to fetch COVID-19 statistics: India

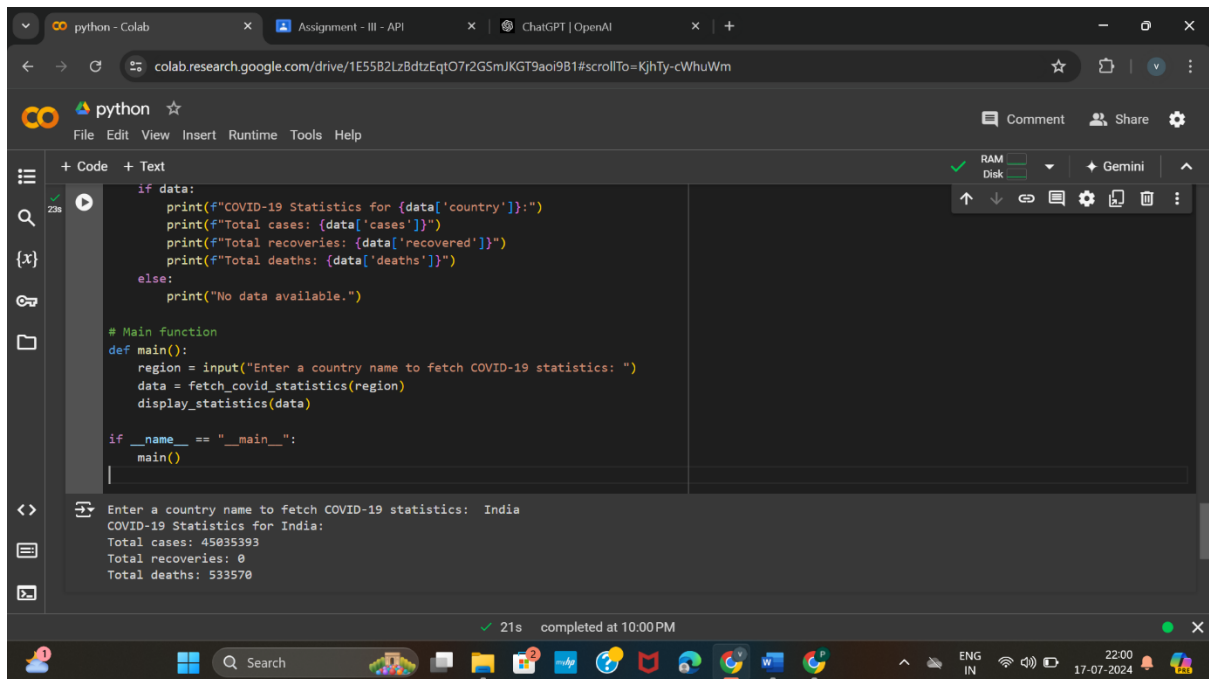
COVID-19 Statistics for India:

Total cases: 45035393

Total recoveries: 0

Total deaths: 533570

OUTPUT:



The screenshot shows a Google Colab notebook interface. The top bar indicates the notebook is named 'python - Colab' and is running on a Google Cloud Platform instance. The URL bar shows the Colab link. The notebook has a dark theme. The code editor shows a Python script that fetches COVID-19 statistics for a given country. The script includes a function to fetch data and a main function that prompts the user for a country name. The output of the script is displayed in the bottom panel, showing the statistics for India.

```
if data:
    print(f"COVID-19 Statistics for {data['country']}:")
    print(f"Total cases: {data['cases']}")
    print(f"Total recoveries: {data['recovered']}")
    print(f"Total deaths: {data['deaths']}")
else:
    print("No data available.")

# Main function
def main():
    region = input("Enter a country name to fetch COVID-19 statistics: ")
    data = fetch_covid_statistics(region)
    display_statistics(data)

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

Enter a country name to fetch COVID-19 statistics: India
COVID-19 Statistics for India:
Total cases: 45035393
Total recoveries: 0
Total deaths: 533570

21s completed at 10:00 PM