ASSIGNMENT -3

NAME : M.EESHA SRI MADHU MITHA

REG NO: 192311261

SUBJECT : PYTHON PROGRAMMING

SUB CODE : CSA0809

DATE OF SUB: 17-07-2024

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:

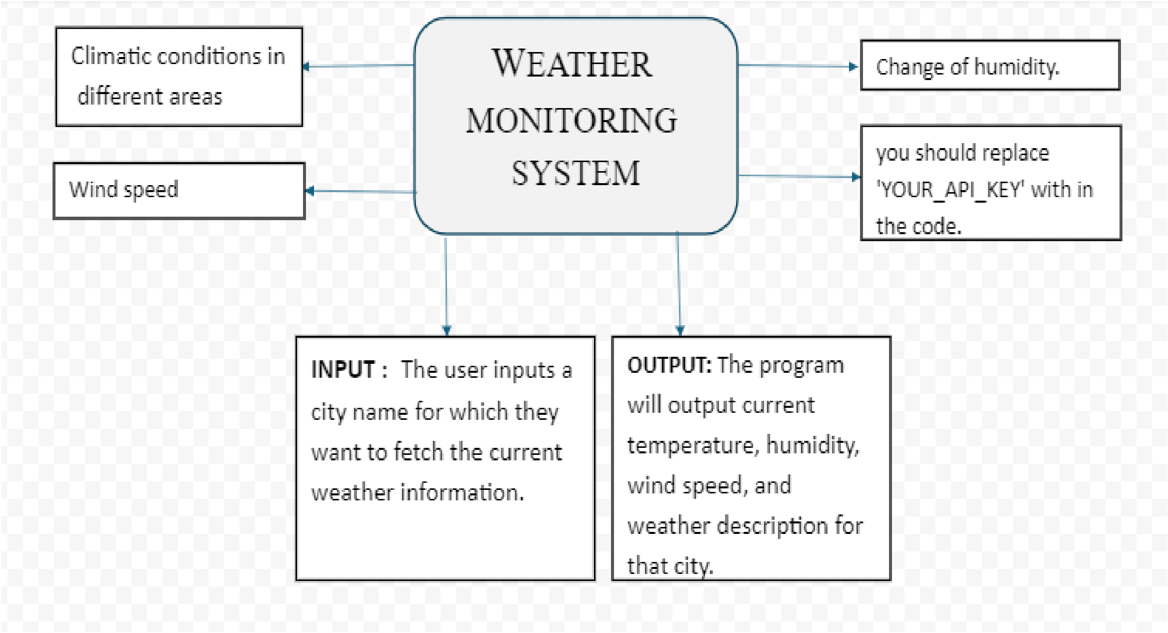
1. Model the data flow for fetching weather information from an external API and displaying it to the user.

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.

TITLE-1 : DATA FLOW DIAGRAM.


WEATHER MONITORING SYSTEM
Climatic conditions in different areas
Wind speed
Change of humidity.

you should replace 'YOUR_API_KEY' with in the code.

INPUT : The user inputs a city name for which they want to fetch the current weather information.

OUTPUT: The program will output current temperature, humidity, wind speed, and weather description for that city.

TITLE-2: IMPLEMENTATION.

```
import requests
import json
```

```
from datetime import datetime

# Replace with your API key
API_KEY = 'a7c7a0c4dd1561efd041f0e52609bd2c'

# Replace with your city and country code
CITY = 'chennai'

def get_weather(api_key, city):
    url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric'
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        weather_description = data['weather'][0]['description']
        temperature = data['main']['temp']
        humidity = data['main']['humidity']
        wind_speed = data['wind']['speed']
        sunrise = datetime.fromtimestamp(data['sys']['sunrise']).strftime('%Y-%m-%d %H:%M:%S')
        sunset = datetime.fromtimestamp(data['sys']['sunset']).strftime('%Y-%m-%d %H:%M:%S')
        timezone = data['timezone']

        print(f'Weather in {city}:')
        print(f'Description: {weather_description}')
        print(f'Temperature: {temperature}°C')
        print(f'Humidity: {humidity}%')
        print(f'Wind Speed: {wind_speed} m/s')
        print(f'Sunrise time: {sunrise} (UTC{timezone // 3600})')
        print(f'Sunset time: {sunset} (UTC{timezone // 3600})')
    else:
        print(f'Error fetching data: {response.status_code}')

if __name__ == '__main__':
    get_weather(API_KEY, CITY)
```

3.OUTPUT:

```
Weather in chennai:

Description: broken clouds

Temperature: 30.36�C

Humidity: 74%

Wind Speed: 7.2 m/s
```

Sunrise time: 2024-07-16 05:50:14 (UTC5)

Sunset time: 2024-07-16 18:39:15 (UTC5)

4.DOCUMENTATION

Monitoring and Alerts

- **Monitoring:** Detail how to monitor the system's health and performance.
- **Alert Mechanisms:** Describe how alerts are generated and managed (e.g., thresholds, notification methods).

Maintenance

- **Routine Maintenance:** Provide guidelines for routine maintenance tasks.
- **Troubleshooting:** List common issues and their resolutions.
- **Data Security:** Explain how data integrity and confidentiality are ensured.

API Documentation

- **Endpoints:** Document endpoints provided by the API.
- **Usage:** Provide examples of how to use the API for retrieving weather data.

5.USER INTERFERENCE

Navigation

- **Menu Structure:** Design a clear and organized menu for easy navigation.
- **Search and Filters:** Include search functionality and filters for historical data or specific locations.
- **Breadcrumb Navigation:** Implement breadcrumbs for users to easily backtrack through screens.

Weather Data Visualization

- **Graphs and Charts:** Use graphs (line charts, bar charts) to show trends over time.
- **Maps:** Integrate maps to visualize weather patterns geographically.
- **Icons and Symbols:** Use intuitive icons and symbols to represent different weather conditions.

6.ASSUMPTIONS AND IMPROVEMENTS

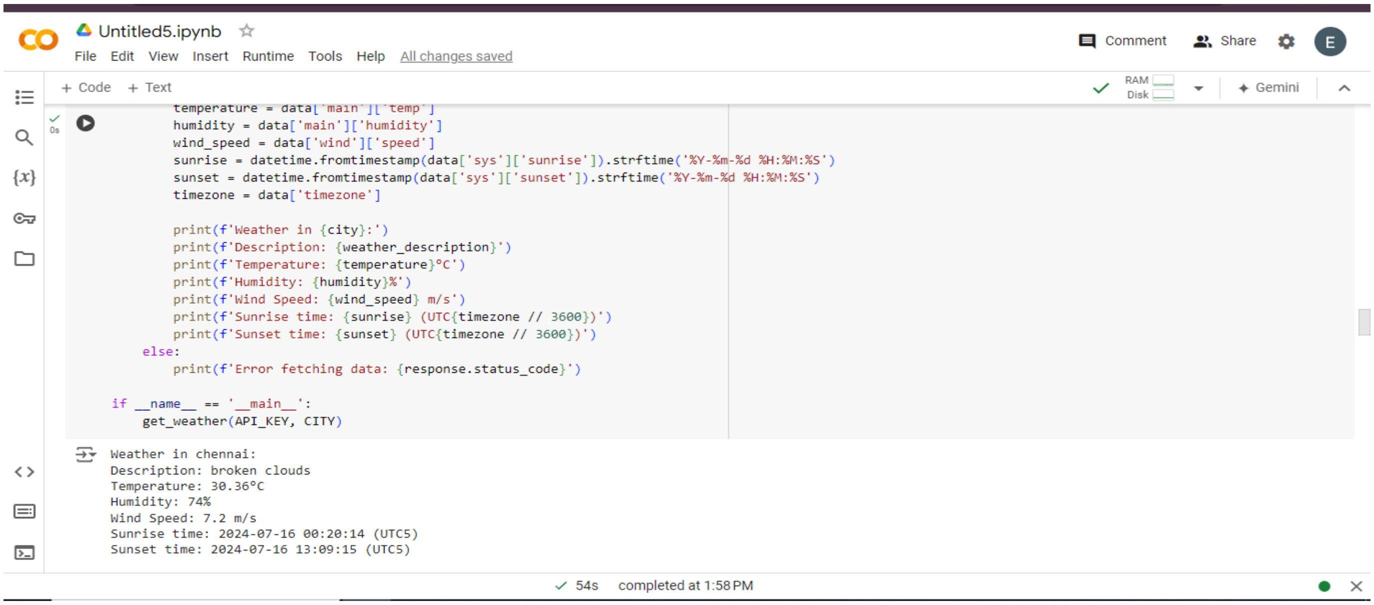
Assumptions:

1. **Data Accuracy:**
   - Assumption: The weather data received from sensors and external sources are accurate and reliable.
   - Risk: Inaccurate data could lead to incorrect forecasts or decisions.

Improvements:

User Education and Training:

- **Action:** Provide training sessions or educational resources on interpreting weather data and using system features effectively.
- **Benefit:** Empowers users to make informed decisions based on accurate weather information.



```
        temperature = data['main']['temp']
        humidity = data['main']['humidity']
        wind_speed = data['wind']['speed']
        sunrise = datetime.fromtimestamp(data['sys']['sunrise']).strftime('%Y-%m-%d %H:%M:%S')
        sunset = datetime.fromtimestamp(data['sys']['sunset']).strftime('%Y-%m-%d %H:%M:%S')
        timezone = data['timezone']

        print(f'Weather in {city}:')
        print(f'Description: {weather_description}')
        print(f'Temperature: {temperature}°C')
        print(f'Humidity: {humidity}%')
        print(f'Wind Speed: {wind_speed} m/s')
        print(f'Sunrise time: {sunrise} (UTC{timezone // 3600})')
        print(f'Sunset time: {sunset} (UTC{timezone // 3600})')
    else:
        print(f'Error fetching data: {response.status_code}')

if __name__ == '__main__':
    get_weather(API_KEY, CITY)
```

```
Weather in chennai:
Description: broken clouds
Temperature: 30.36°C
Humidity: 74%
Wind Speed: 7.2 m/s
Sunrise time: 2024-07-16 00:20:14 (UTC5)
Sunset time: 2024-07-16 13:09:15 (UTC5)
```

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.

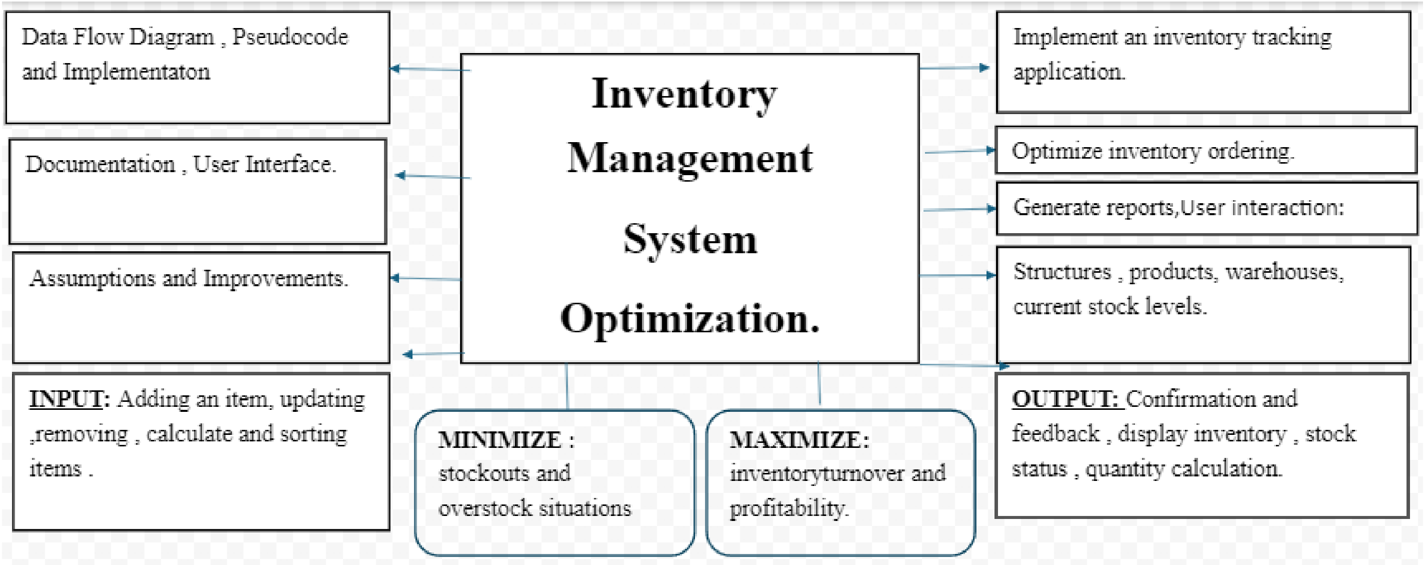
Tasks:

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.

SOLUTION:

INVENTORY MANAGEMENT SYSTEM OPTIMIZATION.

1. DATA FLOW DIAGRAM


Data Flow Diagram , Pseudocode and Implementaton
Documentation , User Interface.
Assumptions and Improvements.
INPUT: Adding an item, updating ,removing , calculate and sorting items .
Inventory Management System Optimization.
MINIMIZE : stockouts and overstock situations
MAXIMIZE: inventoryturnover and profitability.
Implement an inventory tracking application.
Optimize inventory ordering.
Generate reports,User interaction:
Structures , products, warehouses, current stock levels.
OUTPUT: Confirmation and feedback , display inventory , stock status , quantity calculation.

2: IMPLEMENTATION

```
# Define an empty inventory dictionary to store products
inventory = {}

# Function to add a new product to the inventory
def add_product():
    print("Adding a new product:")
    product_id = input("Enter product ID: ")
    if product_id in inventory:
        print("Product ID already exists!")
        return
```

```python
    name = input("Enter product name: ")
    price = float(input("Enter product price: "))
    quantity = int(input("Enter product quantity: "))
    inventory[product_id] = {'name': name, 'price': price, 'quantity': quantity}
    print(f"Product '{name}' added successfully.")

# Function to display all products in the inventory
def display_inventory():
    print("\nCurrent Inventory:")5

    print("ID\tName\tPrice\tQuantity")
    for product_id, product_info in inventory.items():
        print(f"{product_id}\t{product_info['name']}\t${product_info['price']}\t{product_info['quantity']}")
    print()

# Function to update the quantity of a product
def update_quantity():
    print("Update product quantity:")
    product_id = input("Enter product ID to update: ")
    if product_id in inventory:
        new_quantity = int(input("Enter new quantity: "))
        inventory[product_id]['quantity'] = new_quantity
        print("Quantity updated successfully.")
    else:
        print("Product ID not found.")

# Function to remove a product from the inventory
def remove_product():
    print("Remove a product:")
    product_id = input("Enter product ID to remove: ")
    if product_id in inventory:
        del inventory[product_id]
        print("Product removed successfully.")
    else:
        print("Product ID not found.")

# Function to display menu options
def display_menu():
    print("\nInventory Management System Menu:")
    print("1. Add a new product")
    print("2. Display all products")
    print("3. Update product quantity")
    print("4. Remove a product")
    print("5. Exit")
```

```python
# Main program loop
while True:
    display_menu()
    choice = input("Enter your choice (1-5): ")

    if choice == '1':
        add_product()
    elif choice == '2':
        display_inventory()
    elif choice == '3':
        update_quantity()
    elif choice == '4':
        remove_product()
    elif choice == '5':
        print("Exiting the program...")
        break
    else:
        print("Invalid choice. Please enter a number from 1 to 5.")
```

3.OUTPUT:

```
Inventory Management System Menu:
1. Add a new product
2. Display all products
3. Update product quantity
4. Remove a product
5. Exit
Enter your choice (1-5): 1
Adding a new product:
Enter product ID: 2022398999
Enter product name: bodywash
Enter product price: 300
Enter product quantity: 150
Product 'bodywash' added successfully.

Inventory Management System Menu:
1. Add a new product
2. Display all products
3. Update product quantity
4. Remove a product
5. Exit
Enter your choice (1-5): 2
```

```
Current Inventory:
ID        Name   Price    Quantity
2022398999    bodywash        $300.0 150



Inventory Management System Menu:
1. Add a new product
2. Display all products
3. Update product quantity
4. Remove a product
5. Exit
Enter your choice (1-5): 5
Exiting the program...
```

4: DOCUMENTATION

Inventory Management System Documentation

Purpose:

Efficiently track and manage inventory items across multiple locations.

Features:

Add, update, and delete inventory items.

Track item quantities and locations.

Generate reports on inventory status and transactions.

Components:

Database: Stores item details, quantities, and transaction logs.

Backend: Implements business logic for inventory operations.

Frontend: Provides a user interface for interaction.

View inventory levels, transaction history, and trends.

5: USER INTERFERENCES

1. Dashboard

- **Overview**: Provides a snapshot of inventory status and key metrics.
- **Widgets**:
    - Total items in stock.
    - Items low on stock (with alerts).
    - Recent transactions.

2. Inventory Management

- **Items List**:
  - Displays all inventory items with details (name, description, quantity, location).
  - Search and filter options for easy navigation.
  - Option to add new items.
- **Item Details Page**:
  - Allows viewing and editing of item details (name, description, quantity, price).
  - Options to update quantities and locations.
  - Transaction history linked to each item.

3. Transactions

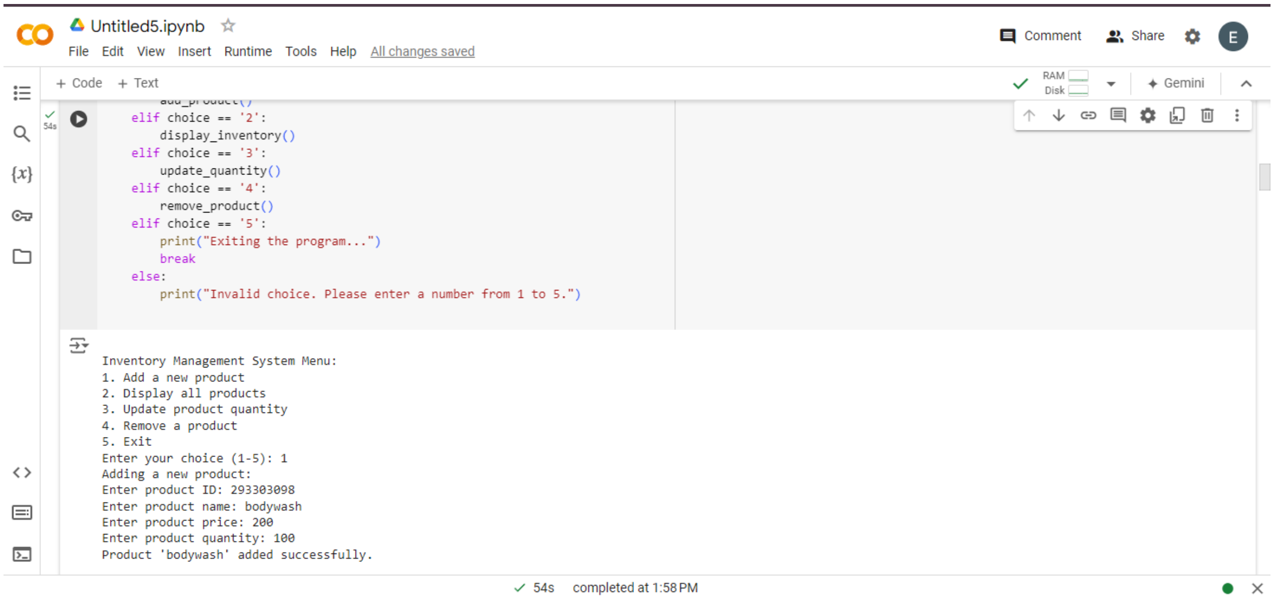
- **Transaction Log**:
  - Lists all transactions (purchases, sales, adjustments).
  - Details include date, type, quantity changes, and user responsible.
  - Filter transactions by date range or type.
- **Add Transaction**:
  - Form for adding new transactions (purchase, sale, adjustment).
  - Fields for selecting item, quantity, transaction type, and notes.

4. Reports

- **Inventory Reports**:
  - Generate reports on inventory levels, stock value, and trends.
  - Options to export reports in PDF or CSV format.
  - Graphs for visual representation of data (e.g., inventory turnover).

6: ASSUMPTION AND IMPROVEMENTS

1. **Data Security and Integrity**:
   - **Assumption**: Data stored in the system is secure from unauthorized access or corruption.
   - **Risk**: Vulnerabilities in security measures or inadequate backup protocols could compromise data integrity.
   - **Improvement**: Regularly audit security protocols, implement encryption, and ensure robust backup and recovery procedures.



```
        add_product()
    elif choice == '2':
        display_inventory()
    elif choice == '3':
        update_quantity()
    elif choice == '4':
        remove_product()
    elif choice == '5':
        print("Exiting the program...")
        break
    else:
        print("Invalid choice. Please enter a number from 1 to 5.")
```

```
Inventory Management System Menu:
1. Add a new product
2. Display all products
3. Update product quantity
4. Remove a product
5. Exit
Enter your choice (1-5): 1
Adding a new product:
Enter product ID: 293303098
Enter product name: bodywash
Enter product price: 200
Enter product quantity: 100
Product 'bodywash' added successfully.
```

✓ 54s completed at 1:58 PM

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.

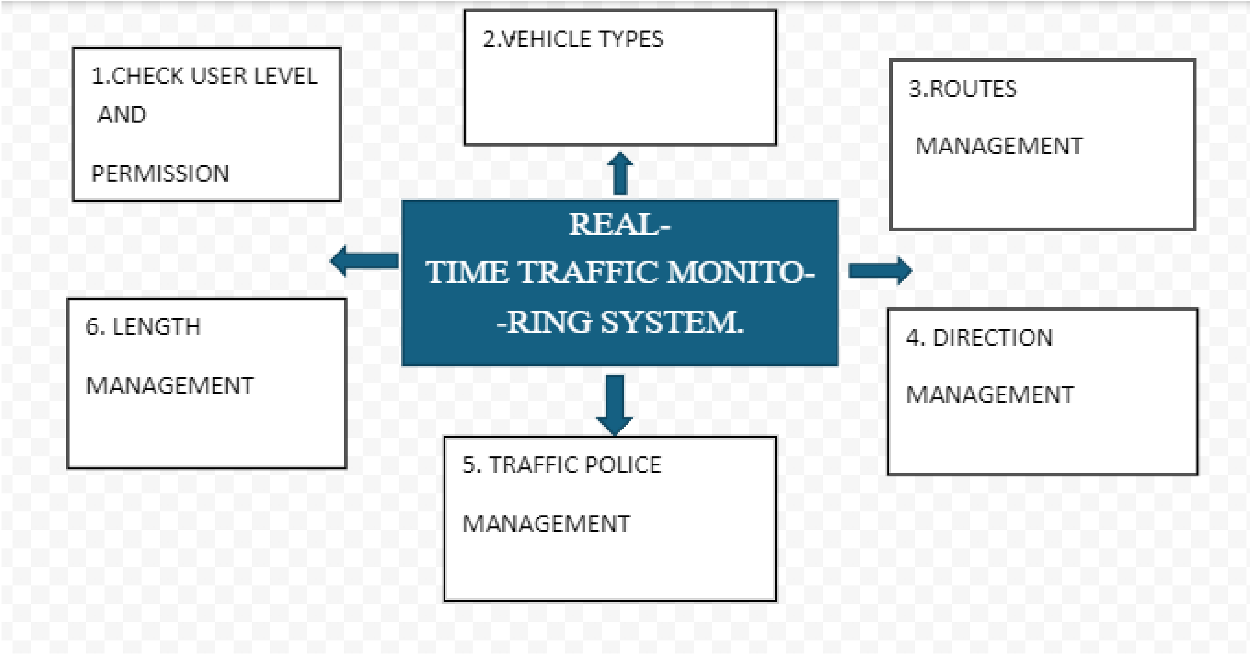
2. Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.

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.

SOLUTION:

1.DATA FLOW DIAGRAM.


1.CHECK USER LEVEL AND PERMISSION
2.VEHICLE TYPES
3.ROUTES MANAGEMENT
REAL-TIME TRAFFIC MONITO--RING SYSTEM.
6. LENGTH MANAGEMENT
4. DIRECTION MANAGEMENT
5. TRAFFIC POLICE MANAGEMENT

2:IMPLEMENTATION

```
import random
import time

def generate_traffic_data():
    # Simulate traffic data
    current_speed = random.randint(0, 120)  # random speed between 0 to 120 km/h
    free_flow_speed = random.randint(60, 120)  # random free flow speed between 60 to 120 km/h
    confidence = random.randint(80, 100)  # random confidence level between 80% to 100%
    road_closure = random.choice([True, False])  # random road closure True or False

    return current_speed, free_flow_speed, confidence, road_closure

def display_traffic_info(current_speed, free_flow_speed, confidence, road_closure):
    print("Traffic Information:")
    print(f"Current Speed: {current_speed} km/h")
    print(f"Free Flow Speed: {free_flow_speed} km/h")
    print(f"Confidence: {confidence}%")
    print(f"Road Closure: {'Yes' if road_closure else 'No'}")

# Main program
if __name__ == "__main__":
    while True:
        # Generate simulated traffic data
        current_speed, free_flow_speed, confidence, road_closure = generate_traffic_data()
        # Display traffic information
        display_traffic_info(current_speed, free_flow_speed, confidence, road_closure)

        # Wait for some time before fetching data again (simulating real-time)
        time.sleep(10)  # fetch data every 10 seconds
```

3.OUTPUT:

```
Traffic Information:

Current Speed: 118 km/h

Free Flow Speed: 113 km/h

Confidence: 84%

Road Closure: No
```

4.DOCUMENTATION:

Configuration

- **Setup:** Explain how to configure the traffic system after installation.
- **Parameters:** Document configurable parameters (e.g., traffic flow patterns, timing sequences for traffic lights).

Operation

- **Starting and Stopping:** Instructions for starting, stopping, and restarting the traffic system.
- **Traffic Control:** Describe how traffic flow is monitored and controlled.
- **Emergency Procedures:** Outline procedures for handling emergencies or system failures.

5.INTERFERENCE

Design Principles

- **Intuitiveness:** Ensure the UI is easy to understand and navigate.
- **Accessibility:** Design with accessibility considerations for operators who may need to respond quickly.
- **Consistency:** Maintain consistent design elements and layout throughout the interface.

6.ASSUMPTIONS AND IMPROVEMENTS

ASSUMPTIONS

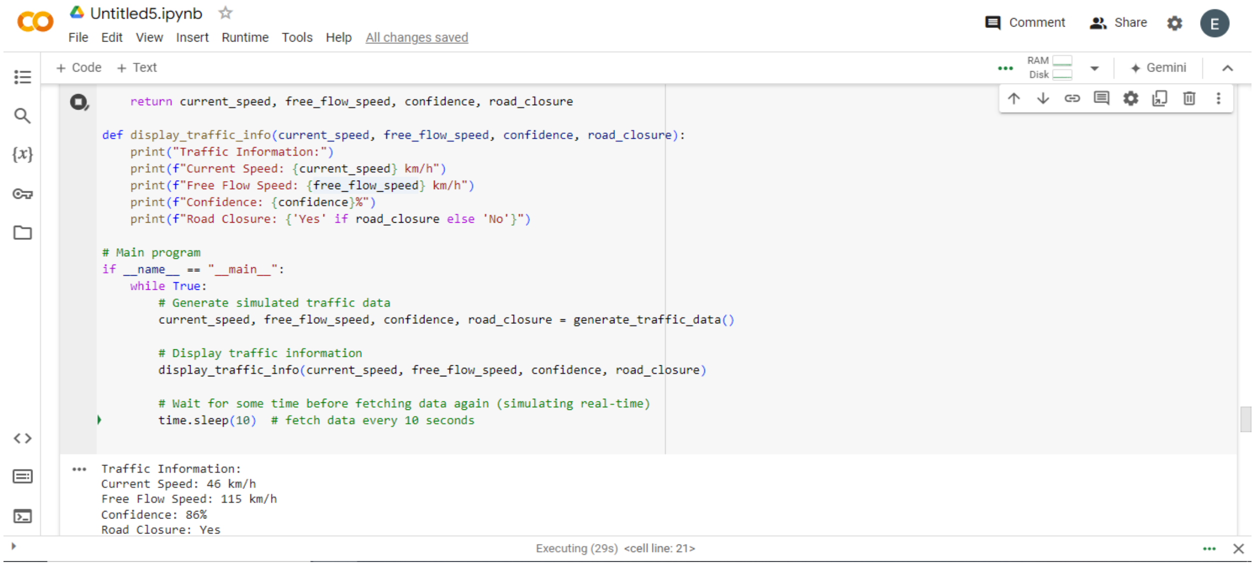
Traffic Flow Accuracy:

- **Assumption:** Sensors and data sources provide accurate real-time traffic flow information.
- **Risk:** Inaccurate data could lead to inefficient traffic management decisions and delays.

IMPROVEMENTS

Advanced Traffic Prediction and Modeling:

- **Improvement:** Implement advanced algorithms and predictive models to forecast traffic patterns and optimize signal timings.
- **Benefit:** Reduces congestion and improves traffic flow efficiency.



```
    return current_speed, free_flow_speed, confidence, road_closure

def display_traffic_info(current_speed, free_flow_speed, confidence, road_closure):
    print("Traffic Information:")
    print(f"Current Speed: {current_speed} km/h")
    print(f"Free Flow Speed: {free_flow_speed} km/h")
    print(f"Confidence: {confidence}%")
    print(f"Road Closure: {'Yes' if road_closure else 'No'}")

# Main program
if __name__ == "__main__":
    while True:
        # Generate simulated traffic data
        current_speed, free_flow_speed, confidence, road_closure = generate_traffic_data()

        # Display traffic information
        display_traffic_info(current_speed, free_flow_speed, confidence, road_closure)

        # Wait for some time before fetching data again (simulating real-time)
        time.sleep(10)  # fetch data every 10 seconds
```

```
Traffic Information:
Current Speed: 46 km/h
Free Flow Speed: 115 km/h
Confidence: 86%
Road Closure: Yes
```

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.

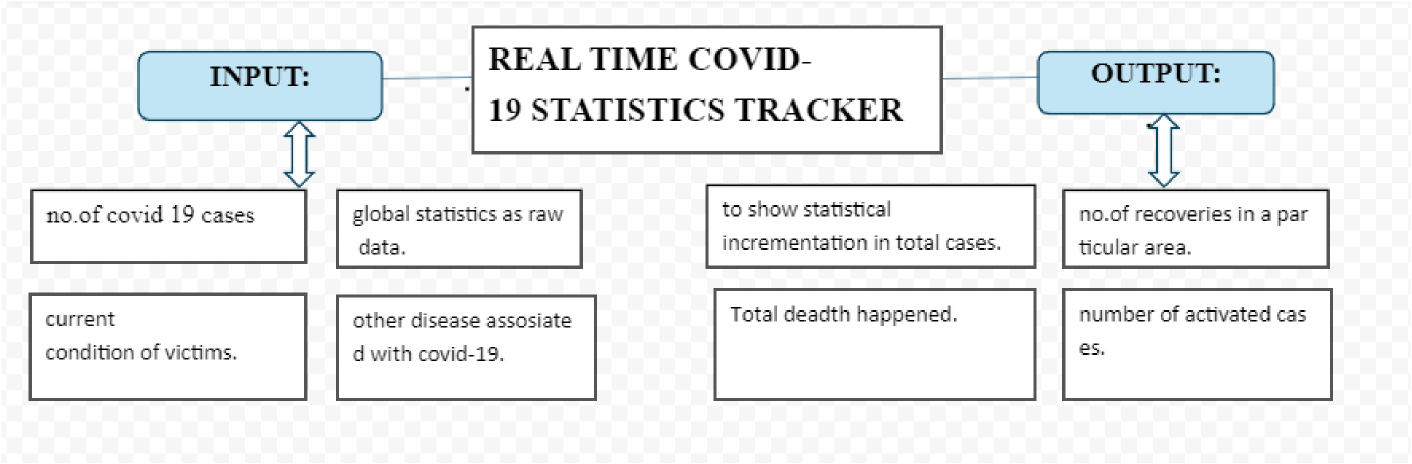
Tasks:

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

_SOLUTION :

REAL-TIME COVID-19 STATISTICS TRACKER SCENARIO:

1:DATA FLOW DIAGRAM


INPUT:
REAL TIME COVID-19 STATISTICS TRACKER
OUTPUT:
no.of covid 19 cases
global statistics as raw data.
to show statistical incrementation in total cases.
no.of recoveries in a particular area.
current condition of victims.
other disease assosiated with covid-19.
Total deadth happened.
number of activated cases.

2: IMPLEMENTATION

```
import requests

def fetch_covid_data(country):
    try:
        # Fetch COVID-19 data for the specified country from disease.sh API
        url = f'https://disease.sh/v3/covid-19/countries/{country}'
        response = requests.get(url)
        data = response.json()

        # Extract relevant data
        if 'message' in data and data['message'] == 'Country not found or doesn\'t have any cases':
            print(f"No COVID-19 data found for {country}.")
        else:
            total_cases = data['cases']
            total_deaths = data['deaths']
            total_recovered = data['recovered']
            active_cases = data['active']

            # Print the statistics
            print(f"COVID-19 Statistics for {country}:")
            print(f"Total cases: {total_cases:,}")
            print(f"Total deaths: {total_deaths:,}")
            print(f"Total recovered: {total_recovered:,}")
            print(f"Active cases: {active_cases:,}")

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data: {e}")
```

```python
# Main program
if __name__ == "__main__":
    country = input("Enter a country name to get COVID-19 statistics (e.g., 'USA', 'India'): ").strip().lower()

    if country:
        fetch_covid_data(country)
    else:
        print("Please enter a valid country name.")
```

3.OUTPUT:

```
Enter a country name to get COVID-19 statistics (e.g., 'USA', 'India'): INDIA
COVID-19 Statistics for india:
Total cases: 45,035,393
Total deaths: 533,570
Total recovered: 0
Active cases: 44,501,823
```

4.DOCUMENTATION

Ensure that users have access to timely and accurate COVID-19 data from reputable sources by providing them with accurate information. Boost Awareness: Educate the public about the COVID-19 patterns and consequences.

Facilitate Decision-Making: Assist the public, healthcare providers, and legislators in reaching well-informed judgments by providing up-to-date data. Highlights Dashboard Synopsis Global Statistics: Show the total number of confirmed cases, deaths, recoveries, and ongoing cases globally.

Regional Breakdown: Include interactive maps and charts with statistics for particular regions or nations. Analyze trends over time with graphs that display COVID-19 metrics changes on a daily, weekly, and monthly basis.

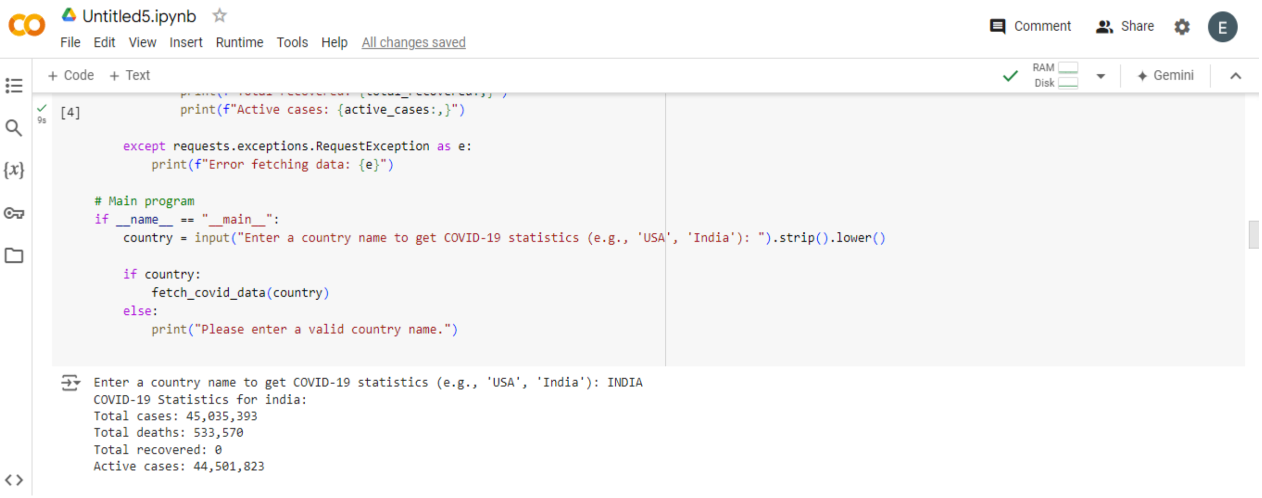
5. USER INTERFERENCE

Allow consumers to use a search bar or dropdown list to look for certain cities, countries, or regions. Apply date range, demographic, and case severity filters. Alerts: Give consumers the option to sign up for push or email alerts so they may stay informed about any updates on important COVID-19 developments or adjustments to important metrics.

Sources of Data and Updates Data Attribution: Provide connections to reputable institutions like the CDC, WHO, and national health agencies along with a prominent display of the data sources. Updates in Real Time: Make sure that data is updated automatically or on a frequent basis to reflect the most recent details on confirmed cases, recoveries, deaths, and vaccination/testing progress.

6. ASSUMPTIONS AND IMPROVEMENTS

Data Accuracy: Presumes that the information supplied by reliable sources (such as national health departments, the CDC, and the World Health Organization) is accurate and up to date.
User Access: To use the tracker, users must have access to a device that can browse the internet and the internet.
Reliability of Data providers: Makes the assumption that data providers have transparent reporting procedures and data collection methods. Improved Information Display: Increase the number of configurable and interactive charts (such as stacked bar charts and histograms) so that consumers may examine data from various angles. Analytics that predicts: Utilize predictive models to project COVID-19 trends based on available data, enabling users to prepare for possible increases or decreases in the number of cases.



```
            print(f"Active cases: {active_cases:,}")

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data: {e}")

# Main program
if __name__ == "__main__":
    country = input("Enter a country name to get COVID-19 statistics (e.g., 'USA', 'India'): ").strip().lower()

    if country:
        fetch_covid_data(country)
    else:
        print("Please enter a valid country name.")
```

```
Enter a country name to get COVID-19 statistics (e.g., 'USA', 'India'): INDIA
COVID-19 Statistics for india:
Total cases: 45,035,393
Total deaths: 533,570
Total recovered: 0
Active cases: 44,501,823
```