

# Assignment-3

*Name: U. Divya*

*Register Number:192372277*

*Department: CSE(AI)*

*Course Name: Python Programming*

*Course Code: CSA0809*

*Date of Submission: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.

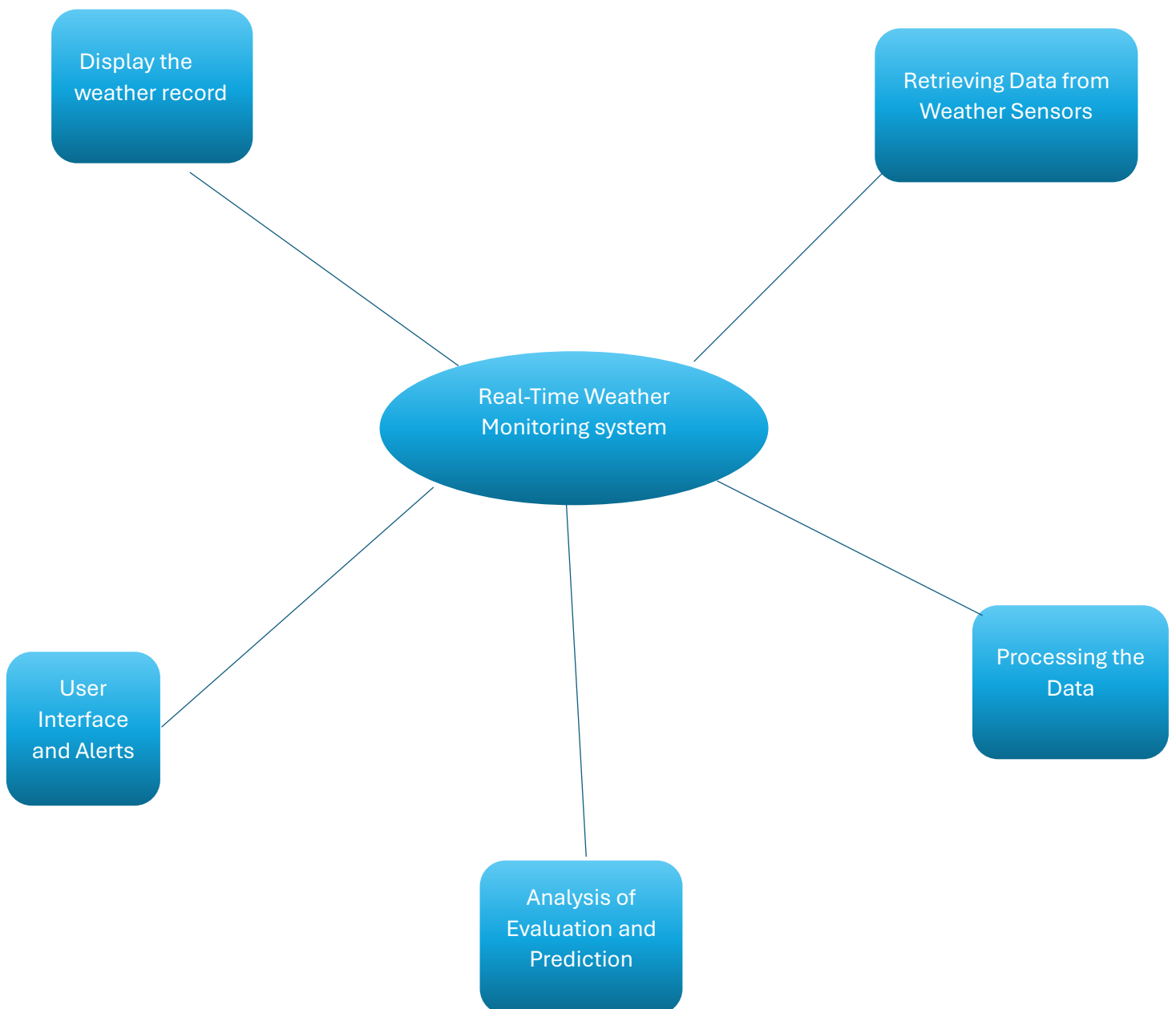
## Tasks:

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.

# SOLUTION:

## Real-Time Weather Monitoring System

### 1. Chart Diagram



## 2. Pseudocode

```
import time
import random

def collect_weather_data():
    temperature = random.uniform(20, 35)
    humidity = random.uniform(40, 80)
    precipitation = random.choice(["None", "Light",
    "Moderate", "Heavy"])
    return temperature, humidity, precipitation

def analyze_weather(temperature, humidity, precipitation):
    print(f"Current Weather Conditions:")
    print(f"Temperature: {temperature} °C")
    print(f"Humidity: {humidity}%")
    print(f"Precipitation: {precipitation}")

def main():
    while True:

        temperature, humidity, precipitation =
collect_weather_data()

        analyze_weather(temperature, humidity, precipitation)

        time.sleep(10)

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

## 3. Implementation

### PROGRAM:

```
import requests
import json

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)
data = response.json()
return data

def main():
    api_key = 'da9f274a7edbe1f0a66107f39b5e3955'
    city = 'London'
    weather_data = get_weather(api_key, city)

    if weather_data['cod'] == 200:
        main_weather = weather_data['weather'][0]['main']
        description =
weather_data['weather'][0]['description']
        temperature = weather_data['main']['temp']
        humidity = weather_data['main']['humidity']
        wind_speed = weather_data['wind']['speed']

        print(f'Weather in {city}:')
        print(f'Main weather: {main_weather}')
        print(f'Description: {description}')
        print(f'Temperature: {temperature}°C')
        print(f'Humidity: {humidity}%')
        print(f'Wind Speed: {wind_speed} m/s')
    else:
        print(f'Error: Could not retrieve weather data for
{city}.')
        print(f'Reason: {weather_data["message"]}')

if __name__ == '__main__':
    main()

```

## 4.OUTPUT:

```

Weather in London:
Main weather: Clouds
Description: broken clouds
Temperature: 14.42°C
Humidity: 93%
Wind Speed: 4.12

```

## **4.Documentation:**

### **1.Demand Forecasting:**

- Use historical sales data to predict future demand using techniques such as moving averages, exponential smoothing, or more advanced methods like ARIMA (AutoRegressive Integrated Moving Average).
- Accurate demand forecasting helps in determining the EOQ and setting appropriate reorder points.

### **2. Lead Time Analysis:**

- Historical lead time data helps in understanding the average and variability in lead times.
- This information is crucial for calculating safety stock and setting reorder points accurately.

### **3.Seasonality and Trends:**

- Historical data can reveal seasonal patterns and trends.
- Inventory policies can be adjusted to account for higher demand during peak seasons and reduced demand during off-peak periods.

## **5.Assumptions and Improvements**

### **1. Constant Lead Times:**

- Many models assume constant lead times for simplicity, but in reality, lead times can vary due to supplier performance, transportation issues, or other factors.
- Safety stock is often used to buffer against lead time variability.

## **2.Fixed Costs:**

- Ordering and holding costs are assumed to be constant in basic models, but they can vary based on order quantities, storage conditions, and other factors.
- Periodic review of cost components is necessary to ensure accurate calculations.

## **3.No Stockouts:**

- Many models assume no stockouts occur, but in reality, stockouts can happen due to unexpected demand spikes or supply chain disruptions.
- Safety stock and appropriate reorder points help mitigate this risk.

## **4.Single Product Focus:**

- Basic models often focus on a single product, but real-world inventory systems deal with multiple products with different demand patterns and cost structures.
- Multi-product optimization and ABC analysis (categorizing inventory based on importance) can be used for more comprehensive inventory management.

## **5.Advanced Sensors:**

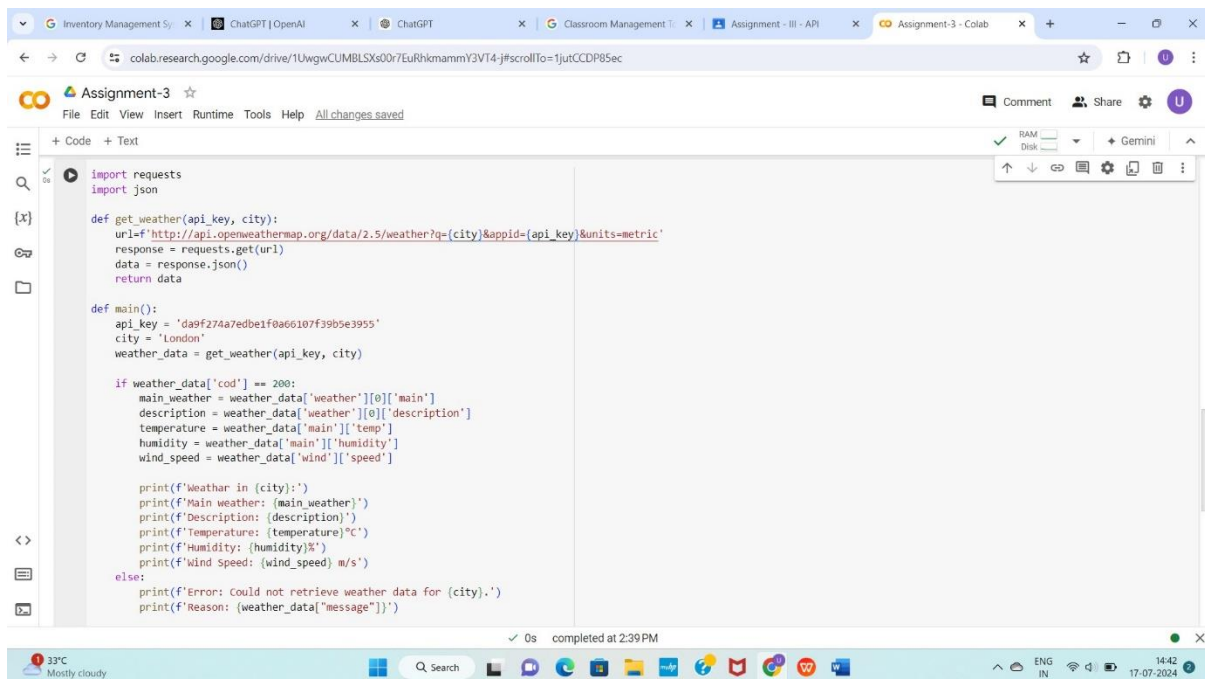
- Utilize high-precision sensors to collect accurate data on temperature, humidity, wind speed, precipitation, and other weather parameters.
- Implement redundant sensors for critical measurements to ensure reliability and accuracy.

## 6. Customizable Alerts:

- Allow users to customize alert settings based on their preferences and needs (e.g., specific weather conditions, locations).
- Provide multi-channel alerts via SMS, email, and social media.

## 7. Data Security:

- Implement strong security measures to protect weather data from unauthorized access and tampering.
- Use encryption, secure communication protocols, and access controls to safeguard data integrity.



The screenshot displays a Google Colab notebook titled "Assignment-3". The code is written in Python and uses the 'requests' library to fetch weather data from the OpenWeatherMap API. The code defines a function 'get\_weather' that takes an API key and a city as input, constructs a URL, and returns the JSON response. A 'main' function is also defined, which calls 'get\_weather' with a specific API key and the city 'London'. It then checks the status code and prints out the main weather information, including description, temperature, humidity, and wind speed. An error handling block is present for cases where data retrieval fails.

```
import requests
import json

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)
    data = response.json()
    return data

def main():
    api_key = 'da9f274a7edbe1f0a66107f39b5e3955'
    city = 'London'
    weather_data = get_weather(api_key, city)

    if weather_data['cod'] == 200:
        main_weather = weather_data['weather'][0]['main']
        description = weather_data['weather'][0]['description']
        temperature = weather_data['main']['temp']
        humidity = weather_data['main']['humidity']
        wind_speed = weather_data['wind']['speed']

        print(f'Weather in {city}:')
        print(f'Main weather: {main_weather}')
        print(f'Description: {description}')
        print(f'Temperature: {temperature}°C')
        print(f'Humidity: {humidity}%')
        print(f'Wind Speed: {wind_speed} m/s')
    else:
        print(f'Error: Could not retrieve weather data for {city}.')
        print(f'Reason: {weather_data["message"]}')

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



## 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.

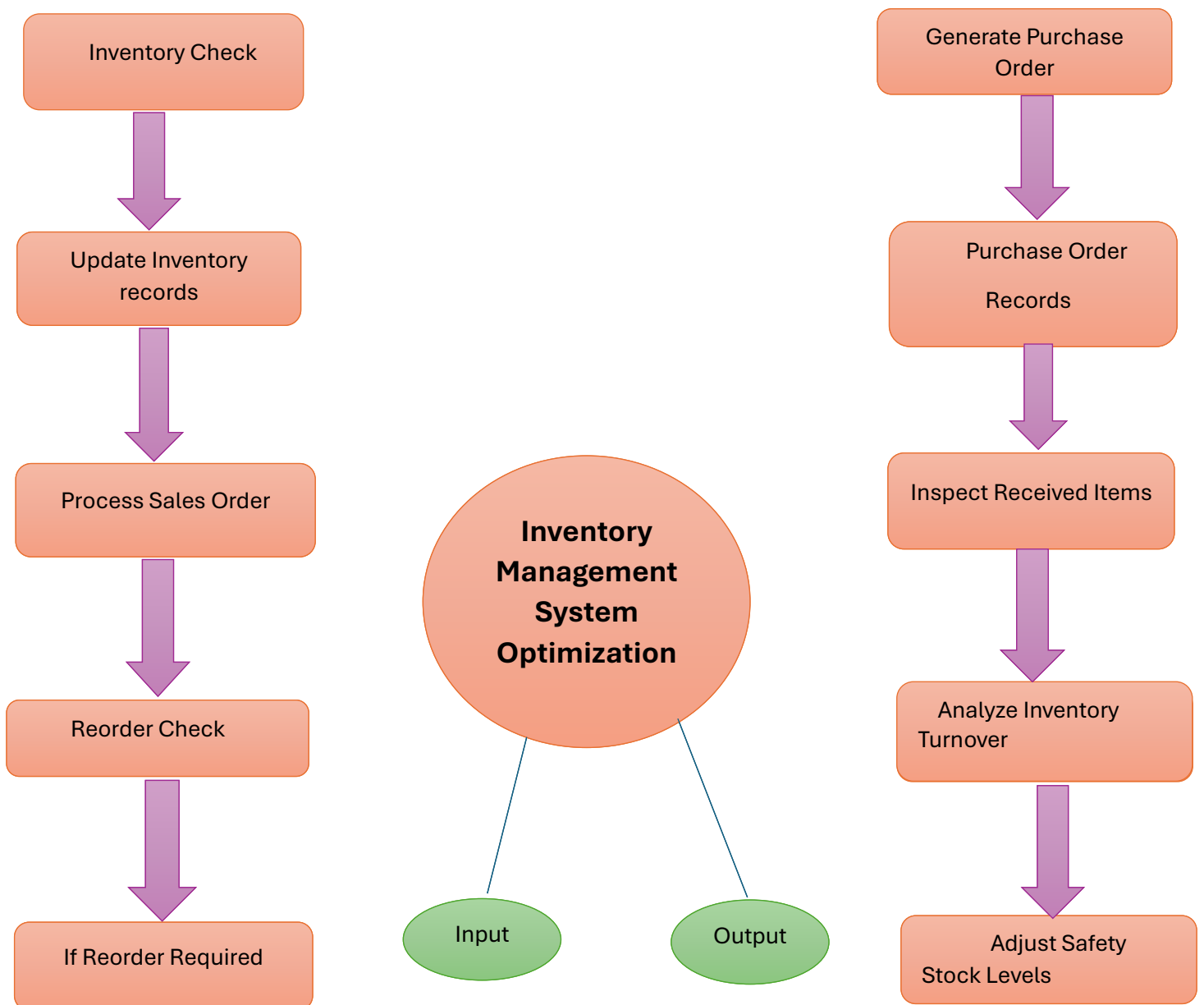
### Tasks:

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 Chart Diagram



## 2: Implementation Code

### Program:

```
class InventoryItem:
    def __init__(self, name, quantity):
        self.name = name
        self.quantity = quantity

class InventoryManagementSystem:
    def __init__(self):
        self.inventory = []

    def add_item(self, name, quantity):

        for item in self.inventory:
            if item.name == name:
                item.quantity += quantity
            return

        new_item = InventoryItem(name, quantity)
        self.inventory.append(new_item)

    def remove_item(self, name, quantity):

        for item in self.inventory:
            if item.name == name:
                if item.quantity >= quantity:
                    item.quantity -= quantity
                else:
                    print(f"Not enough {name} in inventory.")
            return

        print(f"{name} not found in inventory.")

    def print_inventory(self):
        print("Inventory:")
        for item in self.inventory:
            print(f"{item.name}: {item.quantity}")

if __name__ == "__main__":
    ims = InventoryManagementSystem()

    ims.add_item("Apple", 10)
    ims.add_item("Banana", 15)
```

```
ims.add_item("Orange", 20)

ims.print_inventory()

ims.remove_item("Banana", 5)
ims.remove_item("Apple", 15)

ims.print_inventory()
```

### **3.OUTPUT:**

```
Inventory:
Apple: 10
Banana: 15
Orange: 20
Not enough Apple in inventory.
Inventory:
Apple: 10
Banana: 10
Orange: 20
```

## **4.Documentation**

### **1. Economic Order Quantity (EOQ)**

*Determine the optimal order quantity that minimizes total inventory costs.*

### **2.Reorder Point (ROP)**

*Determine the inventory level at which a new order should be placed to avoid stockouts.*

### **3. Safety Stock Calculation**

*Determine the additional inventory to hold to mitigate the risk of stockouts due to demand and lead time variability.*

## **5.Assumptions and Improvements**

### **1.Constant Product Focus:**

*Lead times are assumed to be constant and known. Any variability in lead times is managed using safety stock. Demand is assumed to be constant and predictable. Demand forecasting models are used to account for any fluctuations. Ordering and holding costs are considered constant over time. The models assume no stockouts occur due to accurate demand forecasting and sufficient safety stock. Basic models often focus on a single product, but the methodology can be extended to multiple products*

### **2. Multi-Product Optimization**

- *Use ABC analysis to categorize inventory items based on their importance and adjust policies accordingly.*
- *Implement multi-product inventory optimization models to manage diverse product lines effectively.*

### **3. Inventory Visibility**

- *Enhance inventory visibility across the supply chain using advanced technologies like RFID and blockchain.*
- *Provide real-time inventory data to all stakeholders to improve decision-making.*

### **4. Sustainable Practices**

- *Optimize order quantities and frequencies to reduce environmental impact.*
- *Implement green inventory management practices, such as recycling and waste reduction.*

### **5. Supplier Collaboration**

- *Collaborate closely with suppliers to reduce lead times and improve reliability.*

- *Implement vendor-managed inventory (VMI) systems to streamline ordering processes.*

```
class InventoryItem:
    def __init__(self, name, quantity):
        self.name = name
        self.quantity = quantity

class InventoryManagementSystem:
    def __init__(self):
        self.inventory = []

    def add_item(self, name, quantity):
        for item in self.inventory:
            if item.name == name:
                item.quantity += quantity
                return

        new_item = InventoryItem(name, quantity)
        self.inventory.append(new_item)

    def remove_item(self, name, quantity):
        for item in self.inventory:
            if item.name == name:
                if item.quantity >= quantity:
                    item.quantity -= quantity
                else:
                    print(f"Not enough {name} in inventory.")
                return

        print(f"{name} not found in inventory.")
```

## *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.*

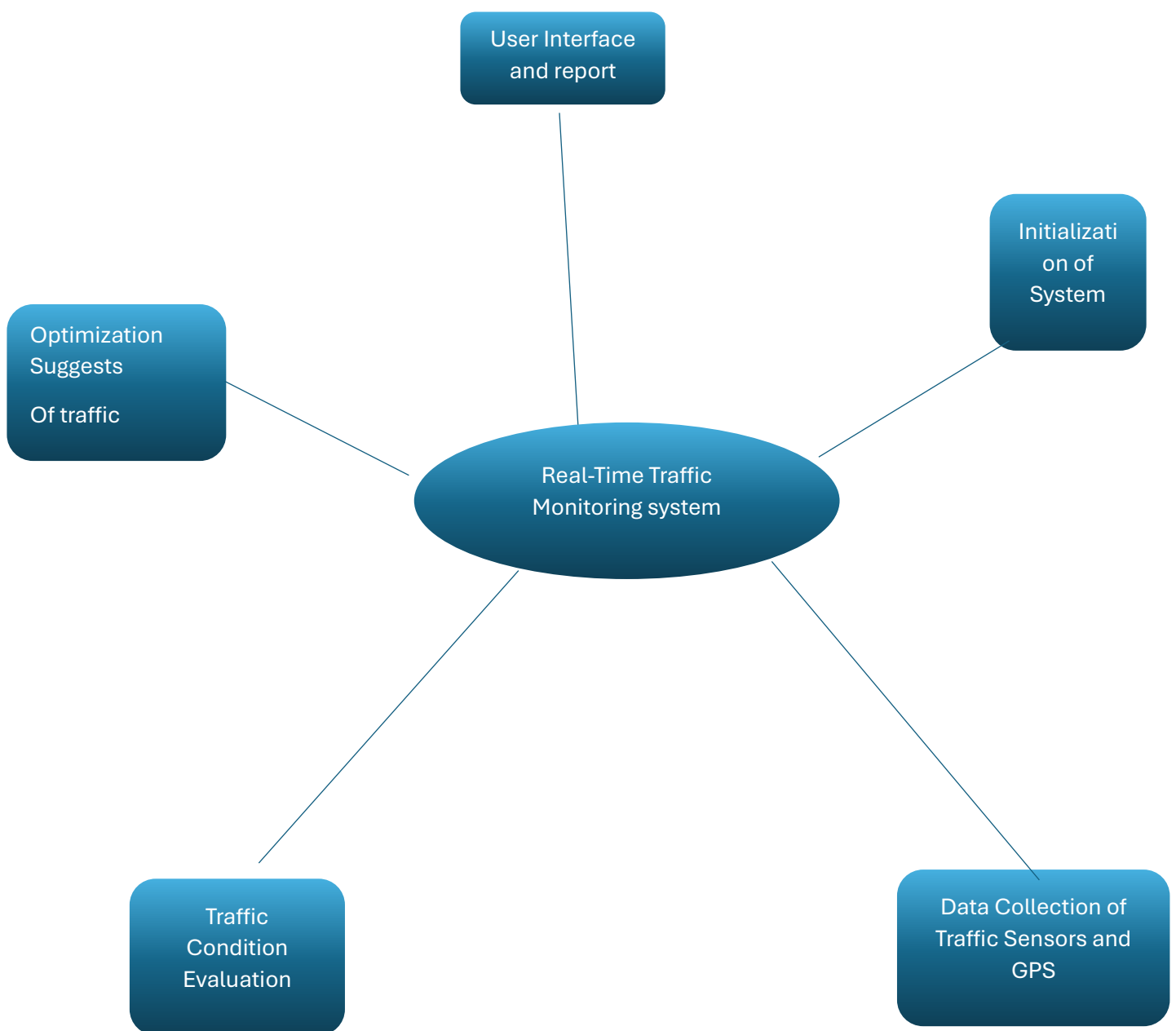
### **Tasks:**

- 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:***

## ***Real-Time Traffic Monitoring System***

### ***TITLE-1: Data Chart Diagram***





## 2: Pseudocode

```
import time
import random

def collect_traffic_data():
    traffic_flow = random.randint(50, 100)
    congestion = random.choice([True, False])
    return traffic_flow, congestion

def analyze_traffic(traffic_flow, congestion):
    if congestion:
        print("Traffic congestion detected!")
    else:
        print("Traffic flow is smooth.")

def main():
    while True:
        traffic_flow, congestion = collect_traffic_data()
        analyze_traffic(traffic_flow, congestion)
        time.sleep(5)

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

## 3. Implementation:

### Program:

```
import random

# Function to simulate traffic data collection
def collect_traffic_data():
    # Simulate traffic flow and congestion status
    traffic_flow = random.randint(50, 100) # Simulate traffic
    flow_rate = (vehicles per minute)
    congestion = random.choice([True, False]) # Simulate
    congestion status
    return traffic_flow, congestion

# Function to analyze traffic conditions
def analyze_traffic(traffic_flow, congestion):
```

```

        if congestion:
            print(f"Traffic congestion detected! Traffic flow:
{traffic_flow} vehicles/min")
        else:
            print(f"Traffic flow is smooth. Traffic flow:
{traffic_flow} vehicles/min")

# Main function to run the traffic monitoring system
def main():
    # Collect real-time traffic data
    traffic_flow, congestion = collect_traffic_data()

    # Analyze traffic conditions based on collected data
    analyze_traffic(traffic_flow, congestion)

# Run the main function
if __name__ == "__main__":
    main()

```

## 4.OUTPUT:

Traffic flow is smooth. Traffic flow: 76 vehicles/min

## **5.Documentation**

### **1. Data Collection**

- *Use of road-side sensors, cameras, and embedded sensors in vehicles to collect real-time traffic data.*
- *Incorporate data from user apps and GPS devices to supplement sensor data.*

### **2. Data Processing and Analysis**

- *Combine data from various sources to create a comprehensive picture of traffic conditions.*
- *Implement machine learning algorithms to predict traffic patterns and identify congestion points.*

- *Use statistical methods to analyze traffic trends and detect anomalies.*

### **3. Traffic Prediction Models**

- *Use historical and real-time data to predict traffic conditions in the near future.*
- *Analyze long-term trends to forecast future traffic scenarios and infrastructure needs.*

### **4. Traffic Management**

- *Implement adaptive traffic signal systems that adjust based on real-time traffic conditions.*
- *Provide real-time route recommendations to drivers to avoid congested areas.*
- *Use algorithms to detect accidents or incidents and alert relevant authorities promptly.*

## **6.Assumptions and Improvements**

### **1.Historical Data Influence**

*Historical traffic data is used to identify trends and recurring congestion patterns. Machine learning models are trained on historical data to improve the accuracy of traffic predictions. Past incidents and their impacts are analyzed to enhance the incident detection algorithms. Historical data helps in understanding seasonal variations in traffic flow and adjusting predictions accordingly.*

### **2. Advanced Analytics**

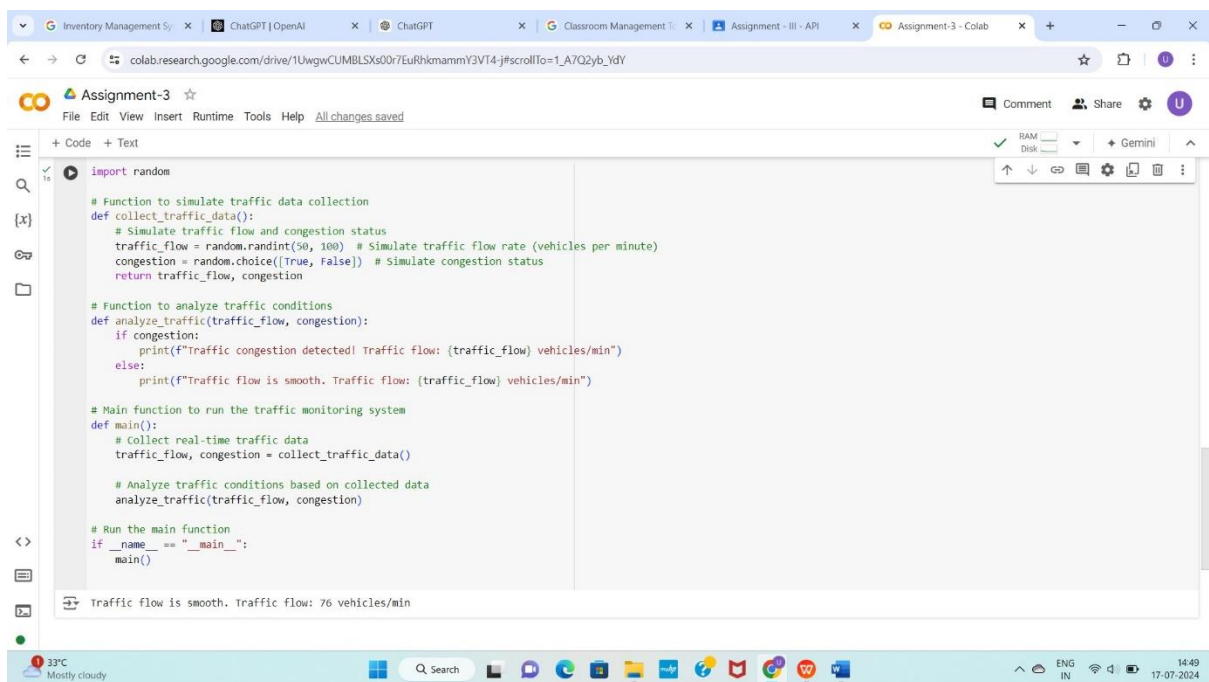
- *Implement more sophisticated machine learning models, such as deep learning, for better traffic prediction.*
- *Use big data analytics to process and analyze large volumes of traffic data in real-time.*

### 3. Enhanced Data Sources

- *Integrate additional data sources such as weather conditions, social media reports, and roadwork schedules.*
- *Use drone technology for aerial traffic monitoring in critical areas.*

### 4. Improved User Interface

- *Develop more intuitive mobile and web applications that provide real-time traffic updates and personalized route suggestions.*
- *Use augmented reality (AR) to display traffic information directly on vehicle windshields.*



```
import random

# Function to simulate traffic data collection
def collect_traffic_data():
    # Simulate traffic flow and congestion status
    traffic_flow = random.randint(50, 100) # Simulate traffic flow rate (vehicles per minute)
    congestion = random.choice([True, False]) # Simulate congestion status
    return traffic_flow, congestion

# Function to analyze traffic conditions
def analyze_traffic(traffic_flow, congestion):
    if congestion:
        print(f"Traffic congestion detected! Traffic flow: {traffic_flow} vehicles/min")
    else:
        print(f"Traffic flow is smooth. Traffic flow: {traffic_flow} vehicles/min")

# Main function to run the traffic monitoring system
def main():
    # Collect real-time traffic data
    traffic_flow, congestion = collect_traffic_data()

    # Analyze traffic conditions based on collected data
    analyze_traffic(traffic_flow, congestion)

# Run the main function
if __name__ == "__main__":
    main()
```

Traffic flow is smooth. Traffic flow: 76 vehicles/min

## **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*

#### *1: Data Chart Diagram*



## 2: Pseudocode

```
import time
import requests
import json

def fetch_covid_data(api_url):
    response = requests.get(api_url)
    if response.status_code == 200:
        data = response.json()
        return data
    else:
        return None

def validate_and_clean_data(data):
    if data and 'cases' in data and 'deaths' in data and 'recovered' in data:
        cleaned_data = {
            'cases': data['cases'],
            'deaths': data['deaths'],
            'recovered': data['recovered'],
            'new_cases': data.get('todayCases', 0),
            'new_deaths': data.get('todayDeaths', 0)
        }
        return cleaned_data
    else:
        return None

def calculate_statistics(data):
    total_cases = data['cases']
    total_deaths = data['deaths']
    total_recovered = data['recovered']
    active_cases = total_cases - total_deaths - total_recovered

    return {
        'total_cases': total_cases,
        'total_deaths': total_deaths,
        'total_recovered': total_recovered,
        'active_cases': active_cases,
        'new_cases': data['new_cases'],
        'new_deaths': data['new_deaths']
    }
```

```

    }

def display_statistics(stats):
    print("COVID-19 Statistics:")
    print(f"Total Cases: {stats['total_cases']}")
    print(f"Total Deaths: {stats['total_deaths']}")
    print(f"Total Recovered: {stats['total_recovered']}")
    print(f"Active Cases: {stats['active_cases']}")
    print(f"New Cases Today: {stats['new_cases']}")
    print(f"New Deaths Today: {stats['new_deaths']}")

def main():
    api_url = "https://disease.sh/v3/covid-19/all"
    while True:

        raw_data = fetch_covid_data(api_url)

        if raw_data:

            clean_data = validate_and_clean_data(raw_data)

            if clean_data:

                stats = calculate_statistics(clean_data)

                display_statistics(stats)

            time.sleep(3600)

if __name__ == "__main__":
    main()

```

### 3.Implementation:

```

import requests
import matplotlib.pyplot as plt

def fetch_covid_data(country):
    url = f"https://disease.sh/v3/covid-19/countries/{country}"
    response = requests.get(url)

```



```

if response.status_code == 200:
    return response.json()
else:
    return None

def display_covid_data(data):
    if data:
        country = data['country']
        cases = data['cases']
        today_cases = data['todayCases']
        deaths = data['deaths']
        today_deaths = data['todayDeaths']
        recovered = data['recovered']
        active = data['active']
        critical = data['critical']

        labels = ['Total Cases', 'Today Cases', 'Total
Deaths', 'Today Deaths', 'Recovered', 'Active', 'Critical']
        values = [cases, today_cases, deaths, today_deaths,
recovered, active, critical]

        plt.figure(figsize=(10, 6))
        plt.bar(labels, values, color=['blue', 'green', 'red',
'orange', 'purple', 'brown', 'pink'])
        plt.xlabel('Categories')
        plt.ylabel('Numbers')
        plt.title(f'COVID-19 Statistics for {country}')
        plt.show()
    else:
        print("Failed to retrieve data")

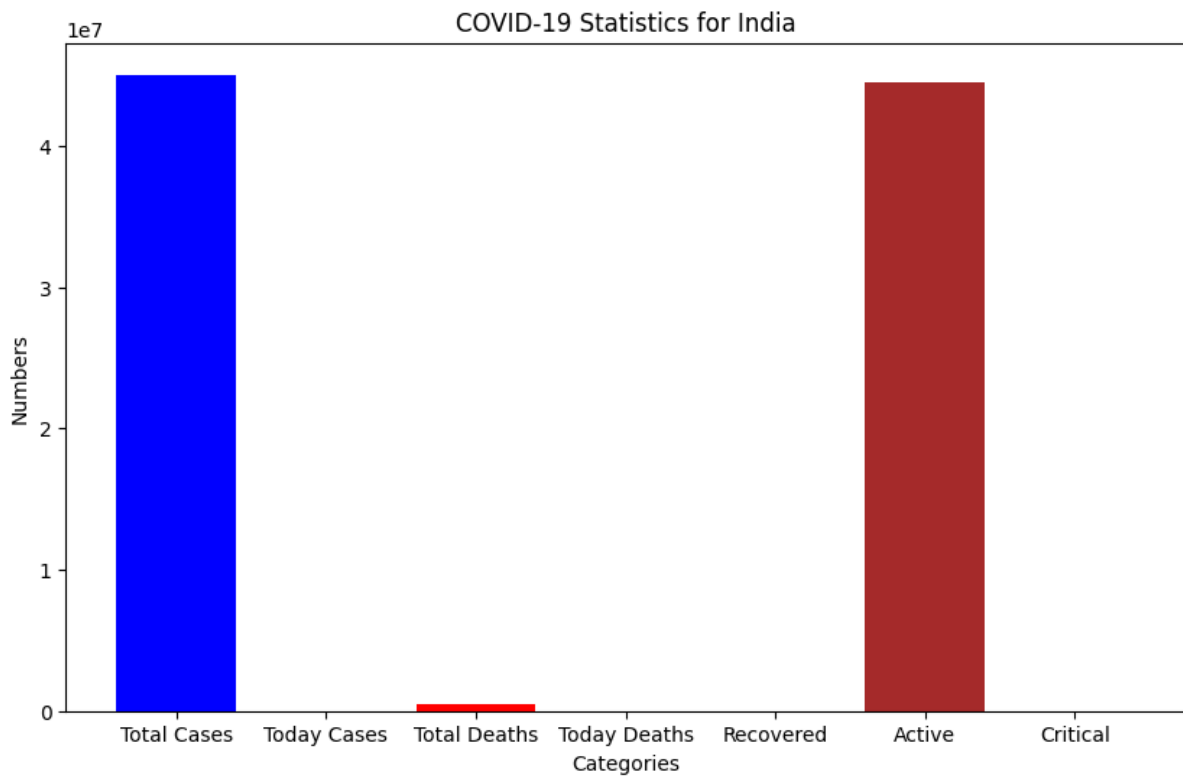
def main():
    country = input("Enter the country name: ")
    data = fetch_covid_data(country)
    display_covid_data(data)

if __name__ == "__main__":
    main()

```

## 4.OUTPUT:

Enter the country name: India



## **5.Documentation:**

### **1. Data Collection**

- *Aggregate data from health departments, government agencies, and international organizations like WHO and CDC.*
- *Utilize data submitted by healthcare providers and verified public reports.*
- *Use web scraping tools to collect data from reliable online sources.*

### **2. Data Processing and Analysis**

- *Ensure the data is accurate, consistent, and free from duplicates.*

- *Implement systems for real-time data processing to provide the latest statistics.*
- *Combine data from various sources to provide a comprehensive view of the situation.*

### **3. Data Presentation**

- *Develop interactive dashboards that display key metrics such as new cases, total cases, recoveries, deaths, and vaccination rates.*
- *Use geographical information systems (GIS) to create real-time maps showing the spread of the virus.*
- *Utilize visual tools to show trends over time and project future scenarios.*

## **6. Assumptions and Improvements**

### **1. Data Accuracy**

- *Assume that the data provided by official sources is accurate and reliable.*
- *Assume that data sources are updated in a timely manner to reflect the current situation.*

### **2. Consistent Definitions**

- *Assume that all data sources use consistent definitions for metrics such as "confirmed cases," "recoveries," and "deaths."*
- *Assume stable internet connectivity for real-time data collection and updates.*

### **3. User Compliance**

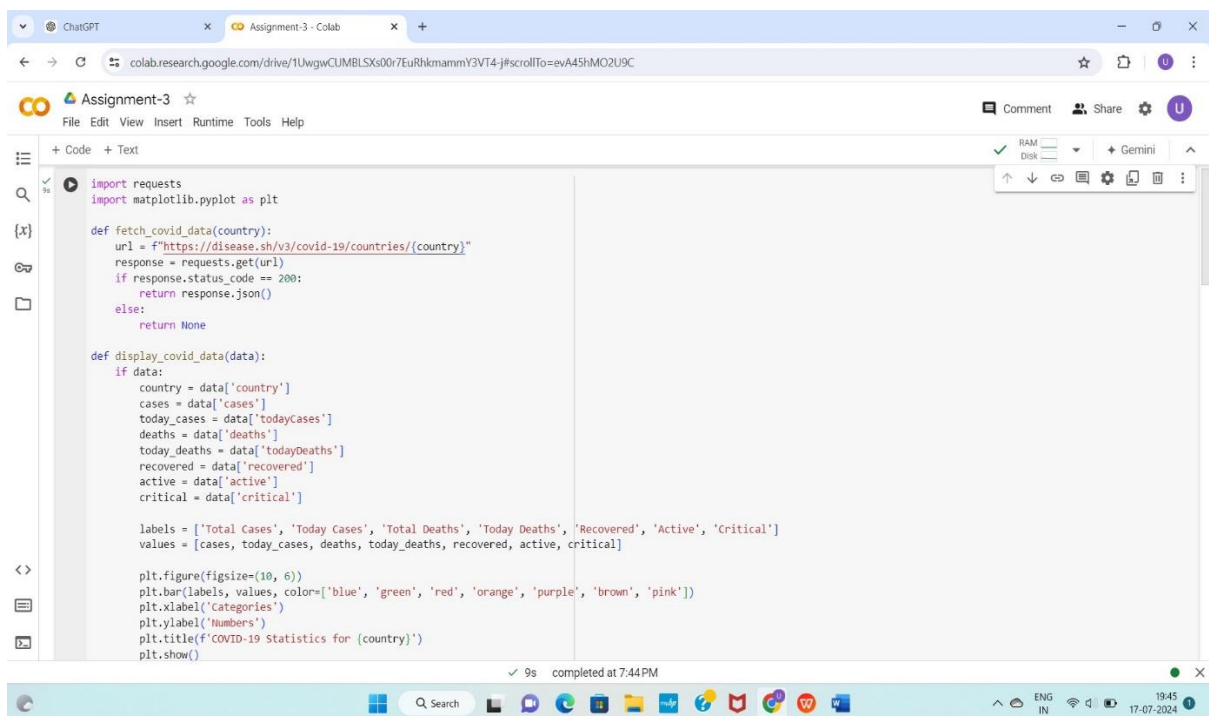
- *Assume users will interpret the data correctly and follow guidelines based on the provided statistics.*

### **4. Public Awareness and Education**

- *Provide educational resources to help the public understand the data and its implications.*
- *Use infographics and multimedia content to make complex data more accessible and understandable.*

## 5. Data Security and Privacy

- *Ensure data is collected and processed in compliance with data privacy regulations.*
- *Implement strong security measures to protect sensitive health data from unauthorized access and breaches.*



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'ChatGPT' and 'Assignment-3 - Colab'. The address bar shows the Colab URL. The notebook title is 'Assignment-3'. The code editor contains the following Python code:

```
import requests
import matplotlib.pyplot as plt

def fetch_covid_data(country):
    url = f"https://disease.sh/v3/covid-19/countries/{country}"
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        return None

def display_covid_data(data):
    if data:
        country = data['country']
        cases = data['cases']
        today_cases = data['todayCases']
        deaths = data['deaths']
        today_deaths = data['todayDeaths']
        recovered = data['recovered']
        active = data['active']
        critical = data['critical']

        labels = ['Total Cases', 'Today Cases', 'Total Deaths', 'Today Deaths', 'Recovered', 'Active', 'Critical']
        values = [cases, today_cases, deaths, today_deaths, recovered, active, critical]

        plt.figure(figsize=(10, 6))
        plt.bar(labels, values, color=['blue', 'green', 'red', 'orange', 'purple', 'brown', 'pink'])
        plt.xlabel('Categories')
        plt.ylabel('Numbers')
        plt.title(f'COVID-19 Statistics for {country}')
        plt.show()
```

The code defines two functions: `fetch_covid_data` to retrieve data from a specific country's API endpoint, and `display_covid_data` to process the data and create a bar chart. The bar chart displays seven categories: Total Cases, Today Cases, Total Deaths, Today Deaths, Recovered, Active, and Critical. The status at the bottom indicates the code was completed at 7:44 PM.