Assignment: Python Programming for DL

Name: Murali sai

Register Number: 192372309

Department: CSE(AI)

Date of Submission: 17-07-24

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., 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.

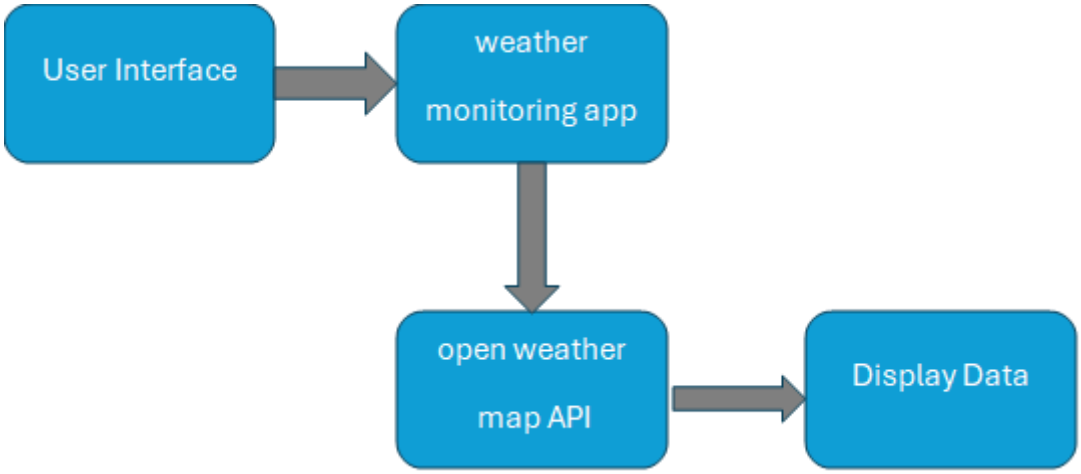
• Documentation of the API integration and the methods used to fetch and display weather data.

• 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.

Implementation:

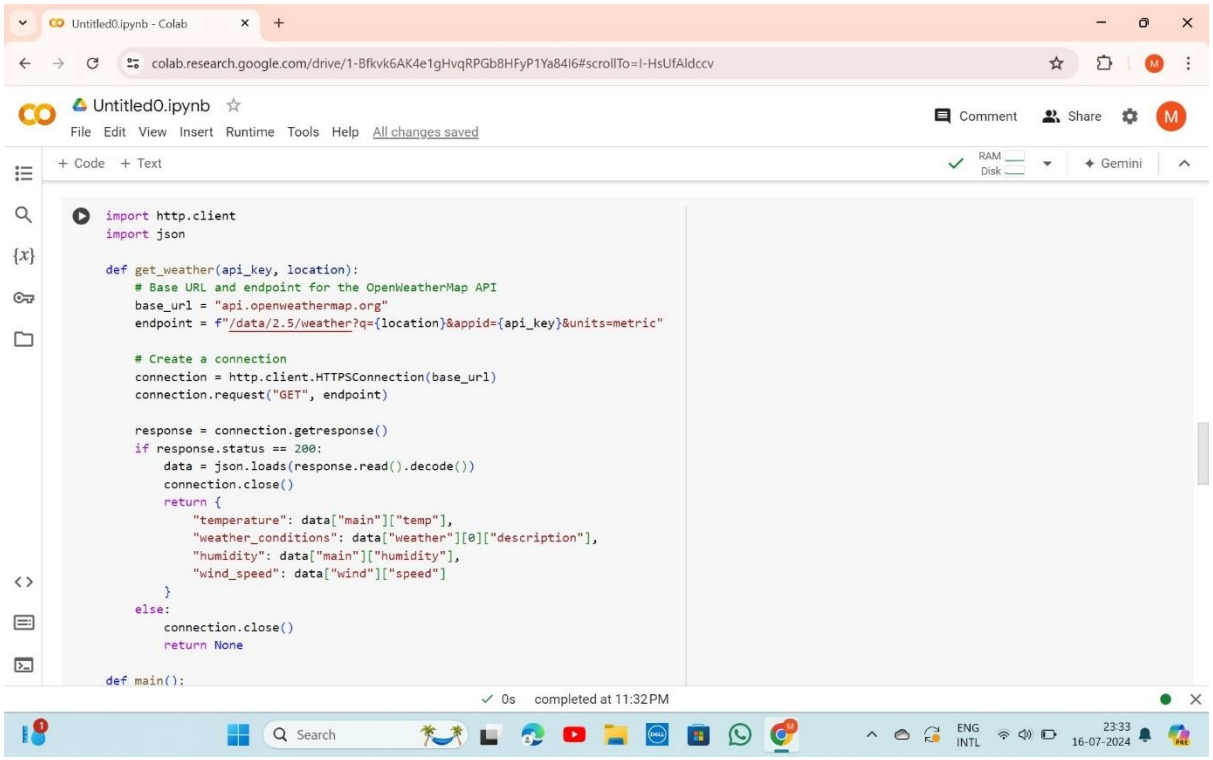
```
import http.client
import json

def get_weather(api_key, location):
```

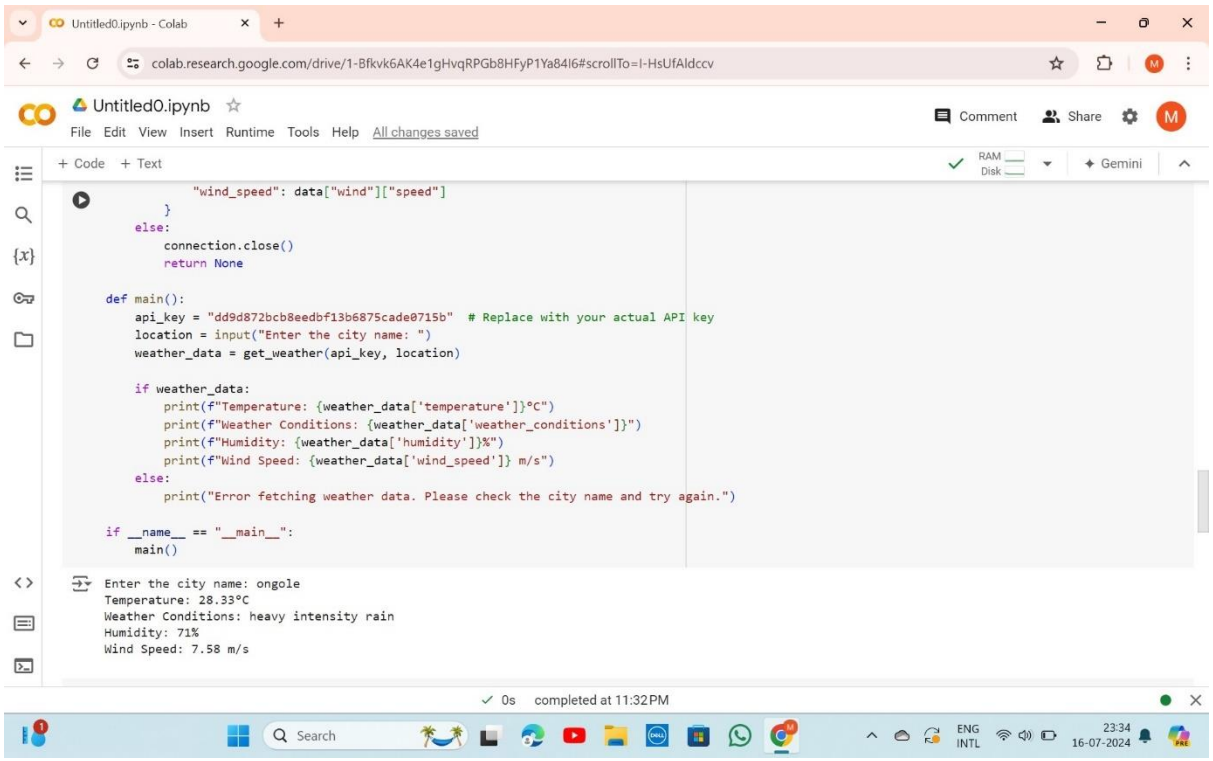
```
    # Base URL and endpoint for the OpenWeatherMap API
    base_url = "api.openweathermap.org"
    endpoint =
f"/data/2.5/weather?q={location}&appid={api_key}&units=metric"

    # Create a connection
    connection = http.client.HTTPSConnection(base_url)
    connection.request("GET", endpoint)

    response = connection.getresponse()
    if response.status == 200:
        data = json.loads(response.read().decode())
        connection.close()
        return {
            "temperature": data["main"]["temp"],
            "weather_conditions": data["weather"][0]["description"],
            "humidity": data["main"]["humidity"],
            "wind_speed": data["wind"]["speed"]
        }
    else:
        connection.close()
        return None

def main():
    api_key = "dd9d872bcb8eedbf13b6875cade0715b"  # Replace with your
actual API key
    location = input("Enter the city name: ")
    weather_data = get_weather(api_key, location)

    if weather_data:
        print(f"Temperature: {weather_data['temperature']}°C")
        print(f"Weather Conditions:
{weather_data['weather_conditions']}")
        print(f"Humidity: {weather_data['humidity']}%")
        print(f"Wind Speed: {weather_data['wind_speed']} m/s")
    else:
        print("Error fetching weather data. Please check the city name
and try again.")

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

output:

- Enter the city name: ongole
- Temperature: 29.24°C
- Weather Conditions: overcast clouds
- Humidity: 63%
- Wind Speed: 6.82 m/s

User Input:


Untitled0.ipynb - Colab
colab.research.google.com/drive/1-Bfkvk6AK4e1gHvqRPGb8HFyP1Ya84I6#scrollTo=I-HsUfAldccv
Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
import http.client
import json
def get_weather(api_key, location):
# Base URL and endpoint for the OpenWeatherMap API
base_url = "api.openweathermap.org"
endpoint = f"/data/2.5/weather?q={location}&appid={api_key}&units=metric"
# Create a connection
connection = http.client.HTTPSConnection(base_url)
connection.request("GET", endpoint)
response = connection.getresponse()
if response.status == 200:
data = json.loads(response.read().decode())
connection.close()
return {
"temperature": data["main"]["temp"],
"weather_conditions": data["weather"][0]["description"],
"humidity": data["main"]["humidity"],
"wind_speed": data["wind"]["speed"]
}
else:
connection.close()
return None
def main():
0s completed at 11:32PM
Search
ENG
INTL
23:33
16-07-2024


Untitled0.ipynb - Colab
colab.research.google.com/drive/1-Bfkvk6AK4e1gHvqRPGb8HFyP1Ya84I6#scrollTo=I-HsUfAldccv
Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
"wind_speed": data["wind"]["speed"]
}
else:
connection.close()
return None
def main():
api_key = "dd9d872bcb8eedbf13b6875cade0715b" # Replace with your actual API key
location = input("Enter the city name: ")
weather_data = get_weather(api_key, location)
if weather_data:
print(f"Temperature: {weather_data['temperature']}°C")
print(f"Weather Conditions: {weather_data['weather_conditions']}")
print(f"Humidity: {weather_data['humidity']}%")
print(f"Wind Speed: {weather_data['wind_speed']} m/s")
else:
print("Error fetching weather data. Please check the city name and try again.")
if __name__ == "__main__":
main()
Enter the city name: ongole
Temperature: 28.33°C
Weather Conditions: heavy intensity rain
Humidity: 71%
Wind Speed: 7.58 m/s
0s completed at 11:32PM
Search
ENG
INTL
23:34
16-07-2024

Documentation:

1. **API 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.

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).

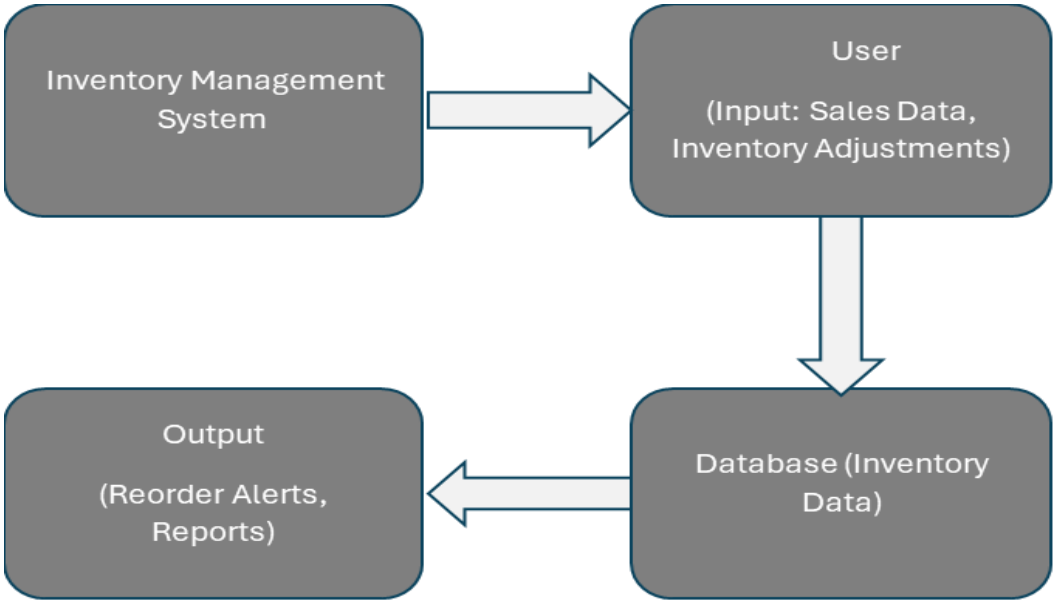
• User Interface: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.

• Assumptions and Improvements: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

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.

Implementation:

```
import pandas as pd

class InventoryManagement:
    def __init__(self):
```

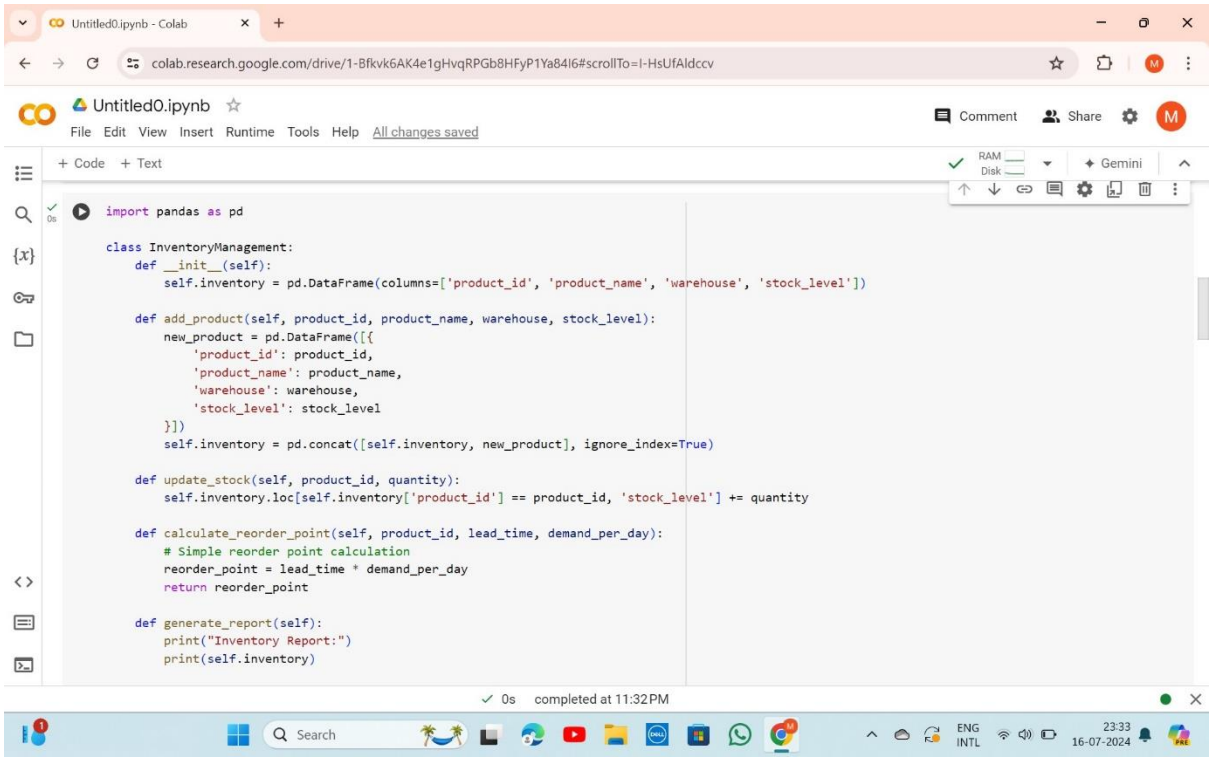
```
        self.inventory = pd.DataFrame(columns=['product_id',
'product_name', 'warehouse', 'stock_level'])

    def add_product(self, product_id, product_name, warehouse,
stock_level):
        new_product = pd.DataFrame([{
            'product_id': product_id,
            'product_name': product_name,
            'warehouse': warehouse,
            'stock_level': stock_level
        }])
        self.inventory = pd.concat([self.inventory, new_product],
ignore_index=True)

    def update_stock(self, product_id, quantity):
        self.inventory.loc[self.inventory['product_id'] == product_id,
'stock_level'] += quantity

    def calculate_reorder_point(self, product_id, lead_time,
demand_per_day):
        # Simple reorder point calculation
        reorder_point = lead_time * demand_per_day
        return reorder_point

    def generate_report(self):
        print("Inventory Report:")
        print(self.inventory)

    def get_stock_level(self, product_id):
        product = self.inventory[self.inventory['product_id'] ==
product_id]
        if not product.empty:
            return product.iloc[0]['stock_level']
        else:
            return None

def main():
    inventory = InventoryManagement()

    # Adding products
    inventory.add_product(1, 'Widget A', 'Warehouse 1', 100)
    inventory.add_product(2, 'Widget B', 'Warehouse 1', 150)

    # Updating stock levels
    inventory.update_stock(1, -10)  # Sold 10 units of Widget A
    inventory.update_stock(2, 50)   # Received 50 units of Widget B

    # Calculating reorder point
```

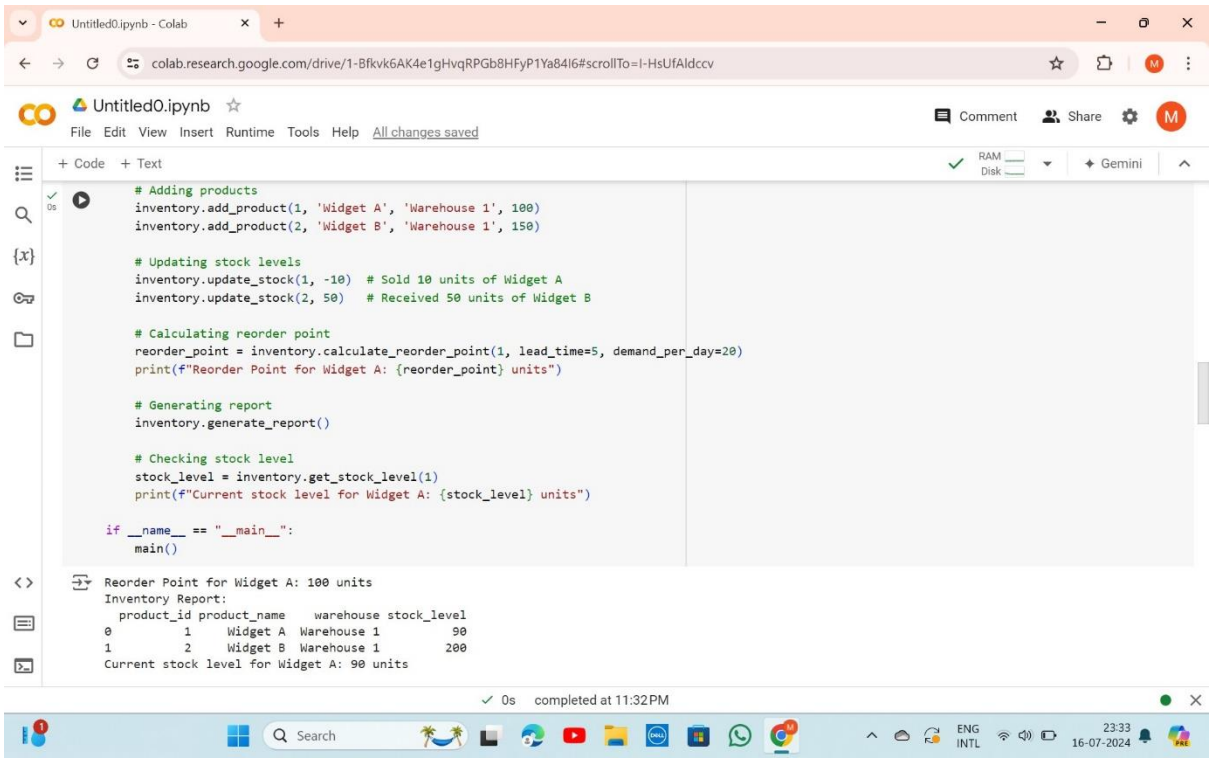
```
    reorder_point = inventory.calculate_reorder_point(1, lead_time=5,
demand_per_day=20)
    print(f"Reorder Point for Widget A: {reorder_point} units")

    # Generating report
    inventory.generate_report()

    # Checking stock level
    stock_level = inventory.get_stock_level(1)
    print(f"Current stock level for Widget A: {stock_level} units")

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

Output:

```
Reorder Point for Widget A: 100 units
Inventory Report:
  product_id product_name    warehouse stock_level
0          1     Widget A  Warehouse 1          90
1          2     Widget B  Warehouse 1         200
Current stock level for Widget A: 90 units
```

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.

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.

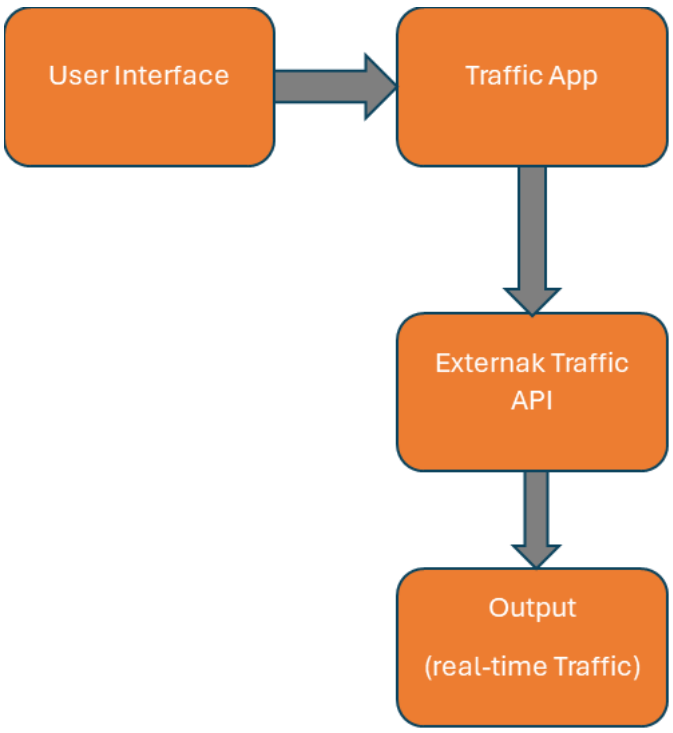
• Documentation of the API integration and the methods used to fetch and display traffic data.

• Explanation of any assumptions made and potential improvements

Solution:

Real-Time Traffic Monitoring System

Data Flow Diagram:



Pseudocode:

1. Get user input for starting point and destination.
2. Send a request to the traffic API with the locations.
3. Receive and parse the traffic data from the API.
4. Display traffic conditions, estimated travel time, and alternative routes.

Implementation:

```
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 = "62e6b236-5eab-42c9-8cc1-a71d01536cc0"  # 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:

```
Enter the starting point: chennai
Enter the destination: pondicherry
```

Error fetching traffic data. Please check the inputs and try again.

User input:

Untitled0.ipynb - Colab

colab.research.google.com/drive/1-Bfkvk6AK4e1gHvqRPGb8HFyP1Ya84I6#scrollTo=I-HsUfAldccv

Untitled0.ipynb

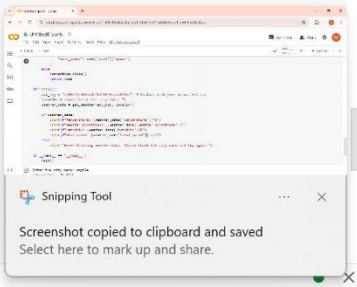
File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Gemini

```
[ ] 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 = "62e6b236-5eab-42c9-8cc1-a71d01536cc0"  # 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)
```



Snipping Tool

Screenshot copied to clipboard and saved
Select here to mark up and share.

✓ 0s completed at 11:32PM

Search ENG INTL 23:34 16-07-2024

Untitled0.ipynb - Colab

colab.research.google.com/drive/1-Bfkvk6AK4e1gHvqRPGb8HFyP1Ya84I6#scrollTo=_GohVaM6hmxO

Untitled0.ipynb

File Edit View Insert Runtime Tools Help

Comment Share

+ Code + Text

RAM Disk Gemini

```
            return None

    def main():
        api_key = "62e6b236-5eab-42c9-8cc1-a71d01536cc0"  # 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()
```

```
Enter the starting point: chennai
Enter the destination: pondicherry
Error fetching traffic data. Please check the inputs and try again.
```

✓ 14s completed at 11:35PM

Search ENG INTL 23:35 16-07-2024

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.

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.

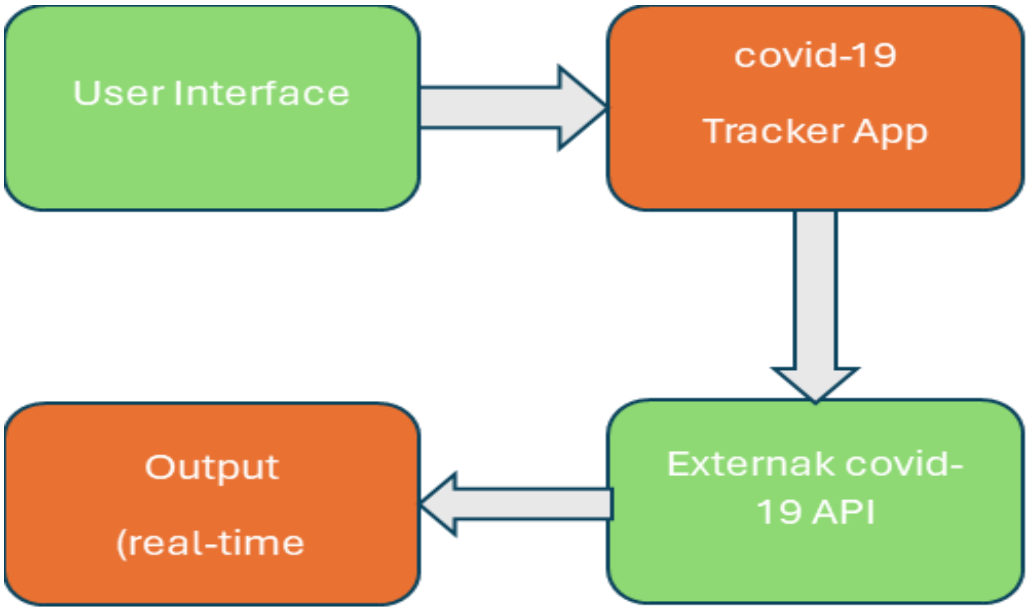
• Documentation of the API integration and the methods used to fetch and display COVID19 data.

• 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.

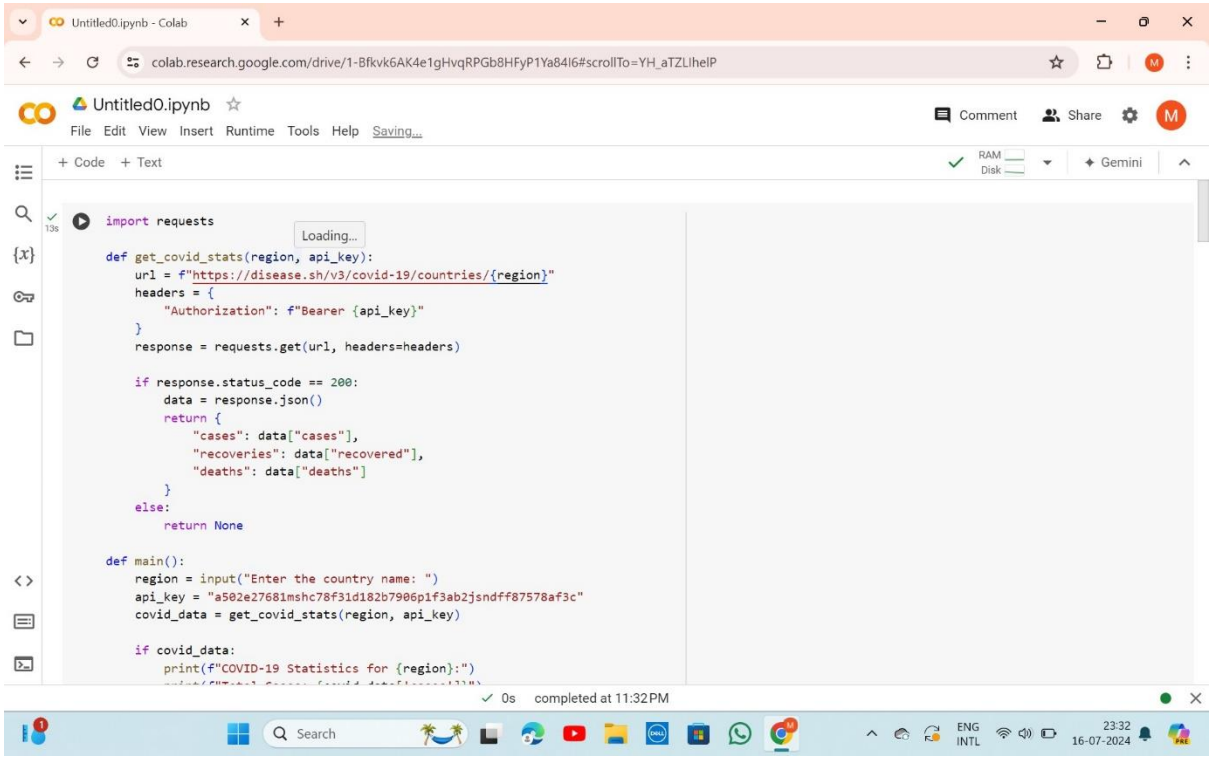
Implementation:

```
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 = "a502e27681mshc78f31d182b7906p1f3ab2jsndff87578af3c"
    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()
```

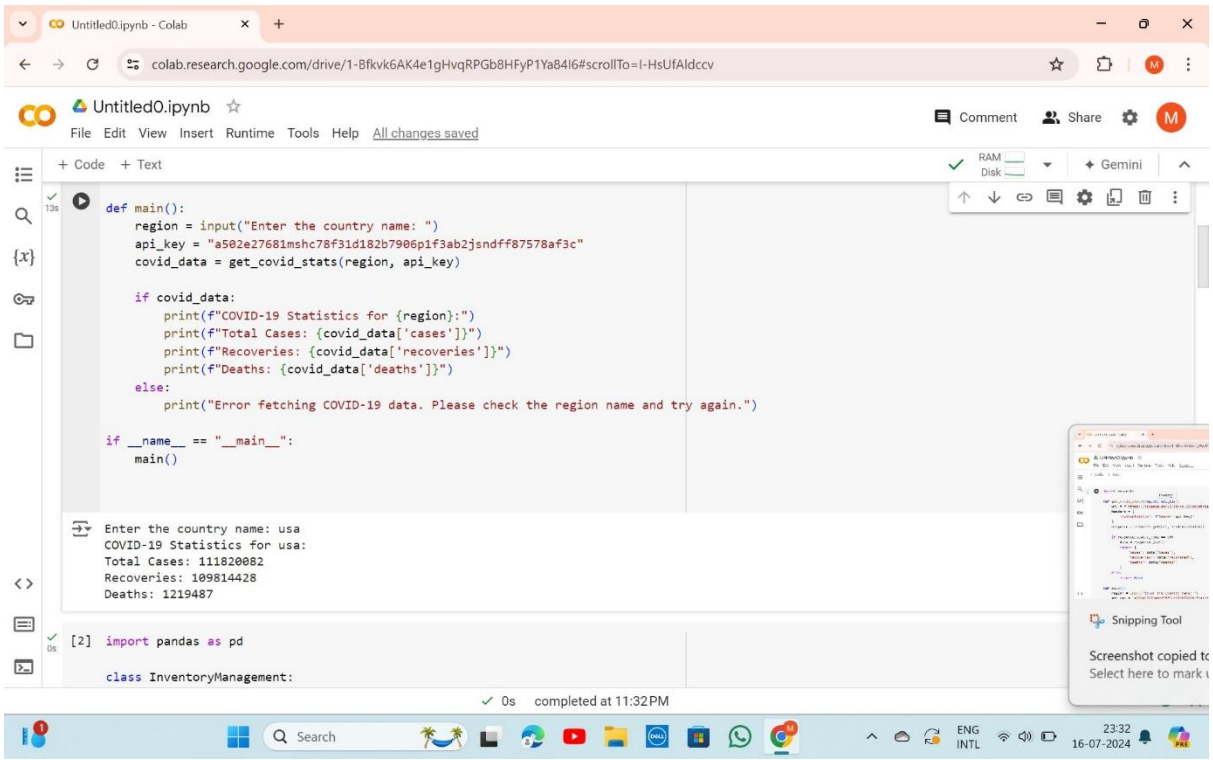
Output:

```
Enter the country name: USA
COVID-19 Statistics for USA:
Total Cases: 111820082
Recoveries: 109814428
Deaths: 1219487
```

User Input:


Untitled0.ipynb - Colab
colab.research.google.com/drive/1-Bfkvk6AK4e1gHvqRPGb8HFyP1Ya84I6#scrollTo=YH_aTZLIhelP
Untitled0.ipynb
File Edit View Insert Runtime Tools Help Saving...
Comment
Share
+ Code + Text
RAM
Disk
Gemini
Loading...
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 = "a502e27681mshc78f31d182b7906p1f3ab2jsndff87578af3c"
covid_data = get_covid_stats(region, api_key)
if covid_data:
print(f"COVID-19 Statistics for {region}:")
0s completed at 11:32PM
Search
ENG
INTL
23:32
16-07-2024

D


Untitled0.ipynb - Colab
colab.research.google.com/drive/1-Bfkvk6AK4e1gHvqRPGb8HFyP1Ya84I6#scrollTo=I-HsUfAldccv
Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
def main():
region = input("Enter the country name: ")
api_key = "a502e27681mshc78f31d182b7906p1f3ab2jsndff87578af3c"
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()
Enter the country name: usa
COVID-19 Statistics for usa:
Total Cases: 111820082
Recoveries: 109814428
Deaths: 1219487
[2] import pandas as pd
class InventoryManagement:
Snipping Tool
Screenshot copied to
Select here to mark
0s completed at 11:32PM
Search
ENG
INTL
23:32
16-07-2024

Documentation:

1. API Integration: Using disease.sh API for real-time COVID-19 statistics.
2. Methods: The get _covid _stats function handles the API request and response. The main function manages user input and displays statistics.
3. Assumptions: The user provides a valid country name.
4. Improvements: Enhance error handling, provide historical data, and integrate with vaccination statistics.