JAYANT RAUTELA
ASSIGNMENT

Problem No.1 a PERSONAL ACCONT MANAGER
Programming lang.: python

```
class Expense:
    def __init__(self, cost, date, tag):
        self.cost = cost
        self.date = date
        self.tag = tag

class Account Manager:
    def __init__(self):
        self.expenses = []

    def add_expense(self, cost, date, tag):
        expense = Expense(cost, date, tag)
        self.expenses.append(expense)
        print("Expense added successfully.")

    def display_expenses(self):
        if not self.expenses:
            print("No expenses recorded yet.")
        else:
            print("Expenses:")
            for idx, expense in enumerate (self. expenses, start=1):
                print(f"{idx}. Cost: {expense.cost}, Date: {expense.date}, Tag:
{expense.tag}")

def main():
    account_manager = AccountManager()

    while True:
        print("\nPersonal Account Manager")
        print("1. Add Expense")
        print("2. Display Expenses")
        print("3. Quit")
        choice = input("Enter your choice: ")

        if choice == "1":
            cost = float(input("Enter expense cost: "))
            date = input("Enter date (YYYY-MM-DD): ")
            tag = input("Enter tag: ")
            account_manager.add_expense(cost, date, tag)
        elif choice == "2":
            account_manager. display expenses()
        elif choice == "3":
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

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

PROBLEM 10: DATA ANALYTSTS Program on Air Quality Index

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset from Kaggle
# Replace 'dataset.csv' with the actual filename and path of your dataset
data = pd.read_('dataset')

# Preprocess the data (cleaning and organizing)
# You might need to remove NaN values, handle missing data, etc.

# Analysis: Top 10 most polluted cities and states for each year
def top_10_polluted_cities(data, year):
    # Filter data for the specific year
    data_year = data[data['Year'] == year]
    # Group by city and calculate mean AQI
    city_aqi = data_year.groupby('City')['AQI'].mean().reset_index()
    # Sort by AQI in descending order
    top_10_cities = city_aqi.sort_values(by='AQI', ascending=False).head(10)
    return top_10_cities

def top_10_polluted_states(data, year):
    # Similar approach as above but with states instead of cities
    pass

# Analysis: Month-wise AQI for Delhi in 2021
def monthly_aqi_delhi_2021(data):
    # Filter data for Delhi and 2021
    delhi_2021 = data[(data['City'] == 'Delhi') & (data['Year'] == 2021)]
    # Group by month and calculate mean AQI
    monthly_aqi = delhi_2021.groupby('Month')['AQI'].mean().reset_index()
    return monthly_aqi

# Visualize the results if needed
def visualize_top_10_cities(top_10_cities):
    # Use Matplotlib to plot the data
    pass

def visualize_top_10_states(top_10_states):
    pass

def visualize_monthly_aqi(monthly_aqi):
    # Plotting the month-wise AQI for Delhi in 2021
    plt.plot(monthly_aqi['Month'], monthly_aqi['AQI'], marker='o')
```

```
    plt.xlabel('Month')
    plt.ylabel('AQI')
    plt.title('Month-wise Air Quality Index (AQI) for Delhi in 2021')
    plt.xticks(range(1, 13))
    plt.grid(True)
    plt.show()

# Main function
def main():
    # Assuming 'data' is the loaded dataset
    # Replace with actual dataset variable
    year = 2021
    top_10_cities_2021 = top_10_polluted_cities(data, year)
    top_10_states_2021 = top_10_polluted_states(data, year)
    monthly_aqi_delhi_2021_data = monthly_aqi_delhi_2021(data)

    # Visualize results if needed
    visualize_top_10_cities(top_10_cities_2021)
    visualize_top_10_states(top_10_states_2021)
    visualize_monthly_aqi(monthly_aqi_delhi_2021_data)

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

PROBLEM No. **3 MEDICINE REMINDER using python**

```
import datetime
import time
import winsound

def set_alarm(alarm_time):
    while True:
        current_time = datetime.datetime.now().strftime("%H:%M")
        if current_time == alarm_time:
            print("Time to take your medicine!")
            # Play sound
            winsound.Beep(1000, 1000)  # Adjust frequency and duration as needed
            break
        time.sleep(60)  # Check every minute

def main():
    print("Welcome to Medicine Reminder!")
    print("Please enter the time you want to set the alarm (format: HH:MM)")
    alarm_time = input("Enter the time: ")

    try:
        datetime.datetime.strptime(alarm_time, "%H:%M")
    except ValueError:
        print("Invalid time format! Please use HH:MM format.")
        return

    print(f"Alarm set for {alarm_time}.")
    set_alarm(alarm_time)
```

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

PROBLEM No.9 The TASK ORGANISER using Python

```python
def main():
    tasks = {}

    while True:
        print("\nTask Organizer")
        print("1. Add Task")
        print("2. View Tasks")
        print("3. Mark Task as Completed")
        print("4. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            add_task(tasks)
        elif choice == '2':
            view_tasks(tasks)
        elif choice == '3':
            mark_completed(tasks)
        elif choice == '4':
            print("Exiting Task Organizer. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

def add_task(tasks):
    task_name = input("Enter the name of the task: ")
    task_priority = input("Enter the priority of the task: ")
    tasks[task_name] = {'priority': task_priority, 'completed': False}
    print("Task added successfully.")

def view_tasks(tasks):
    if not tasks:
        print("No tasks added yet.")
    else:
        print("\nTasks:")
        for i, (task, info) in enumerate(tasks.items(), 1):
            status = "Completed" if info['completed'] else "Not Completed"
            print(f"{i}. {task} - Priority: {info['priority']} - Status: {status}")

def mark_completed(tasks):
    if not tasks:
        print("No tasks added yet.")
    else:
        view_tasks(tasks)
        task_index = int(input("Enter the index of the task to mark as completed: ")) - 1
        if task_index >= 0 and task_index < len(tasks):
            task_name = list(tasks.keys())[task_index]
            tasks[task_name]['completed'] = True
            print(f"{task_name} marked as completed.")
```

```
        else:
            print("Invalid task index.")

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

PROBLEM 12: ENERGY CONSUMPTION FORECASTING using c++

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Function to calculate mean
double mean(const vector<double>& data) {
    double sum = 0.0;
    for (double value : data) {
        sum += value;
    }
    return sum / data.size();
}

// Function to calculate covariance
double covariance(const vector<double>& x, const vector<double>& y) {
    double x_mean = mean(x);
    double y_mean = mean(y);
    double cov = 0.0;
    for (size_t i = 0; i < x.size(); ++i) {
        cov += (x[i] - x_mean) * (y[i] - y_mean);
    }
    return cov / x.size();
}

// Function to calculate variance
double variance(const vector<double>& data) {
    double mean_value = mean(data);
    double var = 0.0;
    for (double value : data) {
        var += pow(value - mean_value, 2);
    }
    return var / data.size();
}

// Function to perform linear regression
pair<double, double> linear_regression(const vector<double>& x, const vector<double>& y) {
    double cov = covariance(x, y);
    double var_x = variance(x);

    // Calculate slope (m) and intercept (c) of the line equation y = mx + c
    double slope = cov / var_x;
    double intercept = mean(y) - slope * mean(x);
```

```
    return make_pair(slope, intercept);
}

// Function to forecast energy consumption
double forecast_energy_consumption(double slope, double intercept, double future_time) {
    return slope * future_time + intercept;
}

int main() {
    // Historical data
    vector<double> time = {1, 2, 3, 4, 5}; // Time in months
    vector<double> energy_consumption = {100, 150, 200, 250, 300}; // Energy consumption
in kWh

    // Perform linear regression
    pair<double, double> regression_params = linear_regression(time, energy_consumption);
    double slope = regression_params.first;
    double intercept = regression_params.second;

    // Forecast energy consumption for future time
    double future_time = 6; // Future time in months
    double forecasted_consumption = forecast_energy_consumption(slope, intercept,
future_time);

    cout << "Forecasted energy consumption after " << future_time << " months: " <<
forecasted_consumption << " kWh" << endl;

    return 0;
}
```

PROGRAM 7: Python program to make an application Geolocation: Displaying User or Device Position on Maps with https://eonet.sci.gsfc.nasa.gov/api/v2.1/events

```
import requests
import folium

def get_events():
    url = 'https://eonet.sci.gsfc.nasa.gov/api/v2.1/events'
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        print("Failed to fetch data from NASA EONET API")
        return None

def display_events_on_map(events):
    map_center = [0, 0]  # Center of the map
    zoom_level = 2  # Zoom level of the map
    map = folium.Map(location=map_center, zoom_start=zoom_level)

    if events is not None and 'events' in events:
        for event in events['events']:
```

```
            if 'geometries' in event and len(event['geometries']) > 0:
                geometry = event['geometries'][0]
                if 'coordinates' in geometry:
                    coordinates = geometry['coordinates']
                    lat, lon = coordinates[:2]
                    folium.Marker(location=[lat, lon], popup=event['title']).add_to(map)
    else:
        folium.Marker(location=map_center, popup="No events found").add_to(map)

    map.save('events_map.html')
    print("Map saved as 'events_map.html'")

if __name__ == "__main__":
    events_data = get_events()
    display_events_on_map(events_data)
```