Application 1 : Rule Engine with AST

```
class Node:
    def __init__(self, node_type, left=None, right=None, value=None):
        self.node_type = node_type
        self.left = left
        self.right = right
        self.value = value
def create_rule(rule_string):
    root_node = Node("operator",
                     left=Node("operand", value=("age", ">", 30)),
                     right=Node("operand", value=("salary", ">", 50000)),
                     value="AND")
    return root_node
def evaluate_rule(node, data):
    if node.node_type == "operator":
        if node.value == "AND":
            return evaluate_rule(node.left, data) and evaluate_rule(node.right, data)
        elif node.value == "OR":
            return evaluate_rule(node.left, data) or evaluate_rule(node.right, data)

    elif node.node_type == "operand":
        attribute, operator, value = node.value
        actual_value = data.get(attribute)
```

```
 if operator == ">":
        return actual_value > value
    elif operator == "<":
        return actual_value < value
    elif operator == "==":
        return actual_value == value
  return False

data = {"age": 35, "department": "Sales", "salary": 60000, "experience": 3}
rule1 = create_rule("age > 30 AND salary > 50000")
rule2 = create_rule("age > 25 AND department == 'Sales'")
rule_asts = [rule1, rule2]
combined_rules = combine_rules(rule_asts)
result = evaluate_rule(combined_rules, data)
print("Eligibility Result:", result)
```

Application 2 : Real-Time Data Processing System for

Weather Monitoring with Rollups and Aggregates

```
import requests
import time

API_KEY = 'YOUR_API_KEY'
cities = ["Delhi", "Mumbai", "Chennai", "Bangalore", "Kolkata", "Hyderabad"]
interval = 300  # 5 minutes

def get_weather_data(city):
```

```
    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"
    response = requests.get(url)
    data = response.json()
    return data

while True:
    for city in cities:
        weather_data = get_weather_data(city)
        process_weather_data(weather_data)
    time.sleep(interval)
def kelvin_to_celsius(kelvin):
    return kelvin - 273.15
dedaily_data = {
    "average_temp": [],
    "max_temp": float('-inf'),
    "min_temp": float('inf'),
    "conditions": []
}

def process_weather_data(data):
    temp = kelvin_to_celsius(data['main']['temp'])
    feels_like = kelvin_to_celsius(data['main']['feels_like'])

    daily_data["average_temp"].append(temp)
    daily_data["max_temp"] = max(daily_data["max_temp"], temp)
    daily_data["min_temp"] = min(daily_data["min_temp"], temp)
    daily_data["conditions"].append(data['weather'][0]['main'])
import statistics

def calculate_daily_summary(daily_data):
    avg_temp = sum(daily_data["average_temp"]) / len(daily_data["average_temp"])
```

```
    max_temp = daily_data["max_temp"]
    min_temp = daily_data["min_temp"]
    dominant_condition = statistics.mode(daily_data["conditions"])  # Most frequent condition
    return {"avg_temp": avg_temp, "max_temp": max_temp, "min_temp": min_temp,
"dominant_condition": dominant_condition}
alert_threshold = {
    "temperature": 35,
    "consecutive_exceed": 2
}
consecutive_high_temp = 0


def check_alert_conditions(temp):
    global consecutive_high_temp
    if temp > alert_threshold["temperature"]:
        consecutive_high_temp += 1
        if consecutive_high_temp >= alert_threshold["consecutive_exceed"]:
            trigger_alert(temp)
    else:
        consecutive_high_temp = 0


def trigger_alert(temp):
    print(f"ALERT: Temperature reached {temp}°C for {alert_threshold['consecutive_exceed']}
consecutive updates.")
CREATE TABLE weather_data (
    id INTEGER PRIMARY KEY,
    city TEXT,
    timestamp INTEGER,
    temp REAL,
    feels_like REAL,
    main_condition TEXT
);
```

```
CREATE TABLE daily_summary (
   id INTEGER PRIMARY KEY,
   city TEXT,
   date TEXT,
   avg_temp REAL,
   max_temp REAL,
   min_temp REAL,
   dominant_condition TEXT
);
import sqlite3


def store_weather_data(city, timestamp, temp, feels_like, main_condition):
   conn = sqlite3.connect('weather.db')
   cursor = conn.cursor()
   cursor.execute("INSERT INTO weather_data (city, timestamp, temp, feels_like, main_condition)
VALUES (?, ?, ?, ?, ?)",
                  (city, timestamp, temp, feels_like, main_condition))
   conn.commit()
   conn.close()
```

```
import matplotlib.pyplot as plt

def visualize_daily_summary(city, dates, avg_temps, max_temps, min_temps):
    plt.figure(figsize=(10, 5))
    plt.plot(dates, avg_temps, label='Avg Temp')
    plt.plot(dates, max_temps, label='Max Temp')
    plt.plot(dates, min_temps, label='Min Temp')




plt.title(f"Daily Temperature Trends for {city}")
    plt.xlabel("Date")
    plt.ylabel("Temperature (°C)")
    plt.legend()
    plt.show()
```