**Develop weather modeling using the quadratic model using Waterfall model**



A weather modeling system using a quadratic model can be developed using the Waterfall model by following a structured, sequential approach. This involves defining requirements, designing the system, implementing the quadratic model, testing, and finally deploying and maintaining the system.

Here's a breakdown of the process using the Waterfall model:

1. Requirements Analysis and Specification:

* **Define the scope:**

Clearly define the objectives of the weather model. For example, predicting temperature, humidity, or rainfall.

* **Gather inputs:**

Identify the necessary weather parameters (temperature, humidity, wind speed, etc.) and their data sources.

* **Determine accuracy requirements:**

Define the desired level of accuracy for the predictions.

* **Specify output format:**

Determine how the model's output will be presented (e.g., hourly forecasts, daily summaries).

* **Document requirements:**

Create a Software Requirements Specification (SRS) document outlining all the above.

2. Design:

* **System Architecture:**

Design the overall structure of the weather modeling system, including data input, processing, and output components.

* **Quadratic Model Design:**

Develop the specific mathematical equations for the quadratic model. This includes defining the coefficients and how they relate to the input parameters.

* **Data Handling:**

Design how the input data will be processed (cleaning, scaling, etc.) and how the model's output will be stored.

* **User Interface:**

Design how the user will interact with the system (e.g., a graphical interface for input and output).

3. Implementation:

* **Coding:** Translate the design into actual code using a suitable programming language (e.g., Python, R).
* **Unit Testing:** Test individual components or modules of the code to ensure they function correctly.
* **Integration:** Combine the different modules into a cohesive system.

4. Testing and Integration:

* **System Testing:** Test the entire system to ensure it meets the requirements defined in the SRS.
* **Verification:** Verify that the model's predictions are accurate and reliable. This involves comparing the model's output with real-world weather data.
* **User Acceptance Testing (UAT):** Allow users to test the system and provide feedback.

5. Deployment and Maintenance:

* **Deployment:**

Deploy the system to the intended environment (e.g., a server, a cloud platform).

* **Maintenance:**

Regularly monitor the system's performance and make necessary updates or bug fixes.

Example:

Let's say we want to predict temperature using a quadratic model.

* **Inputs:** Current temperature (T\\_current), humidity (H), and wind speed (W).
* **Quadratic Model:** T\\_predicted = a \\* T\\_current^2 + b \\* T\\_current + c \\* H + d \\* W + e (where a, b, c, d, and e are coefficients)
* **Design:** The system would take the input parameters, apply the quadratic formula, and output the predicted temperature.

Advantages of using the Waterfall model for this project:

* **Structured and organized:**

The sequential nature of the Waterfall model provides a clear roadmap for development.

* **Well-suited for well-defined requirements:**

If the requirements for the weather model are well-understood and unlikely to change significantly, the Waterfall model can be effective.

* **Easy to manage:**

The Waterfall model is relatively easy to understand and manage, especially for smaller projects.

Disadvantages of using the Waterfall model for this project:

* **Inflexible:** It can be challenging to incorporate changes to requirements or design after a phase has been completed.
* **Limited user feedback:** User feedback is primarily gathered at the end of the development cycle.
* **Risk of delays:** If issues are discovered during testing, they can be costly and time-consuming to fix.

import numpy as np

# Step 1: Input Data

time = np.array([0, 4, 8, 12, 16, 20]) # Time in hours

temperature = np.array([15, 18, 24, 29, 25, 19]) # Temperature in °C

# Step 2: Fit the quadratic model T(t) = a\*t^2 + b\*t + c

coefficients = np.polyfit(time, temperature, 2)

a, b, c = coefficients

print(f"\nDeveloped Quadratic Model:\nT(t) = {a:.4f}t² + {b:.4f}t + {c:.4f}\n")

# Step 3: Predict temperature for every hour from 0 to 24

t\_values = np.arange(0, 25, 1) # Time from 0 to 24 hours

predicted\_temp = a \* t\_values\*\*2 + b \* t\_values + c

print("Predicted Temperature (°C) for 24 Hours:\n")

for t, temp in zip(t\_values, predicted\_temp):

print(f"At {t:02d}:00 hrs -> {temp:.2f} °C")

# Step 4: Plot only if matplotlib is installed

try:

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.scatter(time, temperature, color='blue', label='Original Data', zorder=5)

plt.plot(t\_values, predicted\_temp, color='red', linestyle='--', label='Quadratic Model Prediction')

plt.title('Temperature Prediction using Quadratic Model')

plt.xlabel('Time (Hours)')

plt.ylabel('Temperature (°C)')

plt.xticks(np.arange(0, 25, 2))

plt.grid(True)

plt.legend()

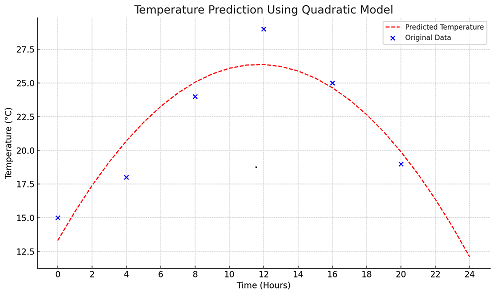
plt.tight\_layout()

plt.show()

except ImportError:

print("\nNOTE: 'matplotlib' is not installed. Skipping graph display.")

print("To install it, run: pip install matplotlib")



Here is the sample graph showing the **predicted temperature curve** (in red dashed line) and the **original data points** (in blue dots).

* **X-axis**: Time in hours (0–24)
* **Y-axis**: Temperature in °C

The quadratic model fits the observed data and predicts a smooth variation of temperature throughout the day. Let me know if you'd like to export this or simulate for different seasons.