Project Spec – Weather App

General Description of the Weather Prediction App (Revised)

The Weather Prediction App is a Python-based tool designed to provide users with real-time and forecasted weather information. The app enables users to input city names and receive weather data, including hourly and daily forecasts, presented in both textual and graphical formats. This information helps users make informed decisions about travel, outdoor activities, and general day-to-day planning.

The app utilizes the **Open-Meteo API** to fetch weather data, which includes temperature, humidity, wind speed, and precipitation. The app also incorporates the **Nominatim geocoding API** to convert user-provided city names into geographic coordinates (latitude and longitude), eliminating the need for manual location input.

Initially built as a command-line interface (CLI) app, the core functionality includes fetching weather data, processing it, and presenting it in a user-friendly format. Future plans involve adding a graphical user interface (GUI) using a framework like Tkinter for a more intuitive user experience.

Revisions Made:

- Replaced manual location input (latitude/longitude) with automatic geocoding via the Nominatim API.
- Removed references to an API key, as the Open-Meteo API does not require one.
- Added real-time weather data retrieval with both textual (table) and visual (plot) outputs.

Vignettes

Vignette 1: Enter Location (Updated)

The user starts the app to check the weather forecast for a specific city. After launching the app, they are prompted to enter a city and state name, such as "Ames, Iowa." The app processes this input using the Nominatim geocoding API, converting the city and state into geographic coordinates (latitude and longitude). The user receives confirmation that the weather data for the specified location is being retrieved.

Revisions:

- Implemented geocoding using the Nominatim API to handle user-provided city/state inputs.

Technical Details:

- The app uses a weather API to locate the city based on user input.
- Error handling must be implemented in case the user enters an invalid or unrecognized location.
- The API request includes the city name, but could also optionally include state or country to avoid ambiguity.
- If using a CLI, the user should receive feedback for successful location matching or error messages if the location is not found.

Vignette 2: Enter Number of Days to Forecast

After entering the location, the user is prompted to specify how many days of weather forecast they would like to see. The user types "7" to see a week-long forecast for New York. The app acknowledges this input and displays a message indicating that it is retrieving weather data for the next 7 days. The response includes daily summaries with key metrics such as maximum and minimum temperatures, humidity, and an overall forecast summary (e.g., "partly cloudy" or "rainy").

Technical Details:

- The app sends another request to the API, specifying the number of days (limited by the API's capabilities, usually between 1 to 16 days).
- The request should include both the location and the forecast length parameters.
- A loop might be used to format the output for each day, listing all key metrics for easy reading in the CLI.

Vignette 3: Select Hourly Weather Parameters

The user wishes to know specific details for a particular day in the forecast. They are given the option to select which weather parameters to view on an hourly basis, such as temperature, precipitation, and wind speed. The user chooses temperature and precipitation for a more detailed analysis. The app then retrieves the selected parameters, showing a breakdown of temperature and expected rainfall for each hour of the selected day.

Technical Details:

- User input is used to select from available weather parameters (e.g., temperature, wind speed, precipitation).
- The API request includes parameters for hourly weather, filtered based on user preference.
- Data processing is done to display only the relevant weather parameters in an easy-to-read format.

Vignette 4: Show Interactive Plots of Parameters

After viewing the hourly details, the user is interested in visualizing this information. They select an option to create an interactive plot of temperature changes throughout the day. The app generates a simple line plot that shows temperature fluctuations from morning to evening, with labels for hours and temperature values. Additionally, the user has the option to interact with the plot by hovering over data points to see exact values at different times of the day.

Technical Details:

- The app requires libraries like matplotlib or plotly to generate plots.
- The data retrieved from the API is used as input to plot hourly temperature or other selected parameters.
- Interactive plots can be implemented using plotly to allow the user to explore the data more deeply.

Vignette 5: Create a Table with Min, Mean, and Max for Each Day

To get a better overview of the forecast, the user selects an option to create a table summarizing the weather over the specified number of days. The app generates a table that displays the minimum, maximum, and mean temperature for each of the next 7 days in New York. This summary allows the user to quickly see trends, such as expected warming or cooling over the week.

Technical Details:

- The daily weather data from the API is aggregated to calculate the minimum, maximum, and mean temperatures.
- The pandas library can be used to structure the data and create the summary table.
- The table is displayed in a readable format within the CLI, and can be saved as a CSV file for future reference.

Vignette 6: Highlighting Parts of the Plots Based on Thresholds

The user wants to easily identify potential hot days. They decide to highlight parts of the temperature plot where the temperature exceeds 80°F. The app takes the user's input and updates the plot, marking all areas where the temperature crosses the 80°F threshold in red. This makes it easy for the user to identify the hottest periods in the forecast visually.

Technical Details:

- The threshold value (e.g., 80°F) is set by the user, and the app uses this value to determine which parts of the plot to highlight.
- Conditional formatting is applied to the plot using a library like matplotlib.
- The implementation involves segmenting the data based on the condition and plotting those segments with a different color.

Technical Flow Overview

Blocks and Grouping (Revised)

- **Geocoding Block (Revised)**:
    - Handles converting the user's city and state input into lat. and long. Using the Nominatim API.
- **Data Handling and Processing Blocks**:
    - API Request Handler
    - Response Parser
    - Forecast Calculation
- **Output Blocks**:
    - Plot Generator
    - Table Formatter
    - Threshold Highlighter

Updated Modules (Revised):

1. **Geocoding Module**:
    - Converts city/state input into geographic coordinates via the Nominatim API.
2. **API Request Handler**:
    - Fetches weather data using the Open-Meteo API without requiring an API key.
3. **Data Processing and Visualization**:
    - Processes weather data and displays results in both text (summary table) and visual (line plot) formats.

Data Types in Data Flow

1. **User Inputs**: String (location), Integer (number of days), List of Strings (selected parameters)
2. **API Request and Response**: Dictionary (for API parameters), JSON (API response)
3. **Processed Data**: List (weather parameters per hour/day), Dictionary (summary statistics)
4. **Visual Outputs**: Plot objects (matplotlib/plotly), pandas DataFrame for tables

User Interaction Notes

- **Location Input**: Text input from user (CLI prompt or GUI text field)
- **Forecast Length Selection**: Numerical input (CLI prompt or GUI dropdown)
- **Parameter Selection**: Selection from a list (CLI prompt or GUI checkboxes)
- **Output Presentation**: Display of text (tables), visual plots (interactive if using a GUI)

Diagram (Revised)

Included "Geocoding Nominatim API"