Student Name: Kalyan Gopinath

Student Number: 230656208

CM2005 – Object Oriented Programming

Report

1. Introduction

This report documents the development of a CLI(Command Line Interface) based application for visualizing and predicting weather data (temperature) across various European countries for the period of 1980–2019. Consisting of four key tasks:

1. **Computing Candlestick Data**
2. **Creating a Text-Based Plot**
3. **Filtering Data and Plotting**
4. **Predicting Data and Plotting**

The dataset (hourly temperatures for multiple countries) is sourced from Open Power System Data. The following sections detail the approach, the code implementation logic (with references), and the outputs corresponding to each task.

---

2. Task 1: Compute Candlestick Data

2.1 As stated in the Assignment

"To complete this task, you need to be able to compute candlestick data from the temperature data for a particular location in Europe between 1980–2019. [...] The Open value is the average mean temperature per time frame (year) in the previous time frame, the Close value is the average mean temperature for the current time frame, the High is the highest temperature for that time frame, and the Low is the lowest temperature for that time frame."

2.2 Approaching task 1

1. **Data Structuring**: Defining a custom `Candlestick` class (`Candlestick.h`) containing:
    - `std::string date;` – representing the year (e.g., `"1980"`).
    - `double open;` – average temperature of the previous year.
    - `double close;` – average temperature of the current year.

- `double high;` – highest temperature in the current year.
- `double low;` – lowest temperature in the current year.

2. **Core Function**: A function `computeCandlesticks(...)` (in **`ComputeCandlesticks.cpp`**) processes the temperature entries. These entries are `(timestamp, temperature)` pairs read in from the CSV file for a specific country column (e.g., `"GB_temperature"`).
   - First grouping the data by year (`YYYY`).
   - For each year from `1980` to `2019`, we:
     - Calculate the year's **average** (for the Close).
     - Use the previous year's Close as the **Open** for the current year.
     - Determine the **High** and **Low** from the max/min temperature in that year's data.
3. **Reference to the code Implementation**
   - **`Candlestick.h`**: Defines the `Candlestick` class.
   - **`ComputeCandlesticks.h`** and **`ComputeCandlesticks.cpp`**: Contain the declaration and implementation of `computeCandlesticks(...)`.
   - Code excerpt (from `ComputeCandlesticks.cpp`):

```
double close = std::accumulate(temperatures.begin(),
temperatures.end(), 0.0) / temperatures.size();
double high  = *std::max_element(temperatures.begin(),
temperatures.end());
double low   = *std::min_element(temperatures.begin(),
temperatures.end());
candlesticks.emplace_back(yearStr, open, close, high, low);
```

2.3 Example of the Output

When running this function, console output (for debugging) may show:

```
Debug: Computed 40 candlesticks.
Year: 1980, Open: 0, Close: 2.357, High: 18.920, Low: -3.120
Year: 1981, Open: 2.357, Close: 2.487, High: 19.300, Low: -4.200
...
Year: 2019, Open: 2.775, Close: 3.032, High: 20.100, Low: -2.820
```

This confirms the Candlestick data structure is populated as required.

---

3. Task 2: Create a Text-Based Plot of the Candlestick Data

3.1 As stated in the Assignment

"The second task involves creating a text-based plot of the candlestick data. [...] Use characters such as '-' to represent the top of a box and '|' to represent the stalk."

3.2 Approaching task 2

1. **Terminal-Based Plot**: I have developed a function `plotCandlesticks(const std::vector<Candlestick>&, int scaleHeight)` in **`PlotCandlesticks.cpp`**.
2. **Normalization**: Each candle's `open`, `close`, `high`, and `low` is mapped to a row index in a fixed vertical scale (default 20 rows).
3. **Plot Construction**: For each row from the top (highest temperature) to the bottom (lowest temperature):
    - We print vertical bars (|) to indicate the range between `lowPos` and `highPos`.
    - The code uses symbols like +, -, = to distinguish `open`/`close` levels.
4. **Pagination**: If there are more candlesticks than can be displayed horizontally, the plotting function prompts the user to press Enter to see the next page.

3.3 Reference to the code Implementation

- **`PlotCandlesticks.h`** and **`PlotCandlesticks.cpp`**: Define and implement `plotCandlesticks(...)`.
- **`Utils.h`**: Contains helper functions like `normalize(...)` which converts a temperature to a row index.

3.4 Example of the Output

Below is an illustrative snippet for a limited dataset, showing the text-based representation:

```
 20.0 | |     |     |
 18.0 | |     |     |
 16.0 | |     |     |
 14.0 | |     +     |
 12.0 | |     |     |
 10.0 | |     |     |
  8.0 | |     -     |
  6.0 | |     |     |
  4.0 | |     |     +
  2.0 | |     |     |
      --------------------
        1980  1981  1982
```

Each column (from left to right) represents one year (e.g., 1980, 1981, 1982).

---

4. Task 3: Filtering Data and Plotting Using Text

4.1 As stated in the Assignment

"Provide at least two filter data options (by date range, country, and temperature data range) and plot the text-based graph."

4.2 Approaching task 3

1. **Filtering Mechanisms**:

- **Filter by Date Range**: Prompt the user for a start year and end year and keep only the candlesticks within this range.
- **Filter by Closing Temperature Range**: Prompt the user for a minimum and maximum "closing temperature" threshold.

2. **Implementation**:
   - The filtering functions are in **`DataFilter.cpp`**:

     ```
     std::vector<Candlestick> filterByYearRange(...);
     std::vector<Candlestick> filterByClosingTemperatureRange(...);
     ```

   - Therefore, prompting the user in `main.cpp` (function `filterCandlesticks(...)`) to decide which filters to apply.
3. **Final Plot**: After filtering, the code reuses the same `plotCandlesticks(...)` function (Section 3) to generate the text-based plot, but only for the subset of data that passed the filter criteria.

4.3 Reference to the code Implementation

- **`DataFilter.h`** and **`DataFilter.cpp`**
- **`main.cpp`**: `filterCandlesticks(...)` integrates user input with the filtering functions.

4.4 Sample Usage

During runtime, users see prompts like:

```
Do you want to filter by date range? (y/n): y
Enter start year (1980-2019): 1990
Enter end year (1980-2019): 1995

Do you want to filter by closing temperature range? (y/n): y
Enter minimum closing temperature: 2.0
Enter maximum closing temperature: 5.0
```

The app then displays and plots only the candlesticks that match years 1990–1995 with a closing temperature between 2.0 and 5.0.

---

5. Task 4: Predicting Data and Plotting

5.1 As stated in the Assignment

"Predict temperature changes between the date range of your choice and a country. [...] Support the statements by developing appropriate prediction functions and include a brief description of the method used."

5.2 Approaching task 4

1. **Method Chosen**: **Simple Linear Regression** based on historical (1980–2019) data.
    - Treats each year (`YYYY`) as `x` and its average closing temperature for that year as `y`.
    - Compute slope `m` and intercept `c` using the standard linear regression formula.
2. **Prediction Function**:
    - Implemented in **`TemperaturePredictor.cpp`**:
        - `calculateLinearRegression(...)`: calculates slope and intercept.
        - `predictTemperatures(int startYear, int endYear)`: for each year in `[startYear, endYear]`:
            - Compute `predictedClose = m * year + c`.
            - For simplicity, set `open = close = predictedClose`, `high = close + 1`, `low = close - 1`.
3. **Plotting Predictions**: The returned `std::vector<Candlestick>` from `predictTemperatures(...)` is again passed to `plotCandlesticks(...)`, generating a text-based "forecast" plot.

5.3 Justification of the Model

A **linear regression** is a straightforward approach that highlights whether there is an overall warming (positive slope) or cooling (negative slope) trend over time. Although more complex models (e.g., polynomial regression, ARIMA, or neural networks) could be used, linear regression suffices as an initial proof-of-concept and satisfies the assignment requirement to "calculate these values from historical data provided."

5.4 Code Fragment

From **`TemperaturePredictor.cpp`**:

```
void TemperaturePredictor::calculateLinearRegression(double& m, double& c)
{
    int n = data.size();
    // sums, sums of squares, etc. computed here
    m = (n * sum_xy - sum_x * sum_y) / denominator;
    c = (sum_y * sum_x2 - sum_x * sum_xy) / denominator;
}
```

Then used in:

```
std::vector<Candlestick> TemperaturePredictor::predictTemperatures(int
startYear, int endYear) {
    calculateLinearRegression(m, c);
    for (int year = startYear; year <= endYear; ++year) {
        double predictedClose = m * year + c;
        predictions.emplace_back(std::to_string(year),
                                 predictedClose,   // open
                                 predictedClose,   // close
                                 predictedClose+1, // high
                                 predictedClose-1  // low
        );
    }
    return predictions;
}
```

6. Code Structure and Execution

1. **`main.cpp`**
   - Handles the **Menu** system:
     - **Select Country and Display Table** (reads CSV, computes candlesticks, displays in tabular form).
     - **Plot Candlestick Data** (calls `plotCandlesticks(...)`).
     - **Filter Plot Data** (calls the filtering functions, then plots).
     - **Predict Temperature Changes** (uses `TemperaturePredictor`, then plots predictions).
     - **Exit**.
2. **`CSVReader.{h,cpp}`**
   - Responsible for reading the CSV file (e.g., `"weather_data.csv"`) for a specified temperature column (`"<country>_temperature"`).
3. **`Candlestick.{h,cpp}`**, **`ComputeCandlesticks.{h,cpp}`**
   - Data structure and computation for candlestick data.
4. **`PlotCandlesticks.{h,cpp}`**
   - Renders a text-based candlestick plot in the console.
5. **`DataFilter.{h,cpp}`**
   - Filtering by year range or closing temperature range.
6. **`TemperaturePredictor.{h,cpp}`**
   - Simple linear regression approach to forecasting future candlestick data.
7. **`Utils.h`**
   - Helper functions for normalization and clamping values for the textual plot.

---

7. Demonstration and Results

- **System Demonstration Video**: A maximum 3-minute video (required by the assignment) shows:
  1. **Selecting a Country** (e.g., "GB Temperature") and computing candlestick data.
  2. **Displaying the Table** of candlesticks (1980–2019).
  3. **Plotting** the resulting candlesticks (text-based).
  4. **Filtering** by date (e.g., 1990–2000) and/or closing temperature range (e.g., 2.0–6.0).
  5. **Predicting** future data (e.g., 2020–2025) and plotting the predictions.
- **Screenshots** can be captured at each stage (particularly the text-based plot) to showcase functionality, as required by the assignment.

---

8. Conclusion

This report demonstrates the successful development of a **command-line weather analysis tool** fulfilling the four assignment tasks:

- **Task 1**: We computed annual candlestick data from hourly temperature data by grouping and taking average, min, and max.
- **Task 2**: We created a **text-based candlestick plot** for the computed data using ASCII characters.
- **Task 3**: We provided **at least two filtering options**—date range and closing temperature range—before plotting.
- **Task 4**: We implemented a **simple linear regression** predictor to forecast future temperatures, integrating the results back into the same candlestick data structure for plotting.

All code has been clearly separated into header (`.h`) and source (`.cpp`) files, with references included in this report. The design emphasizes reusability, readability, and modularity, aligning with object-oriented programming principles.

References

1. **Open Power System Data** - Weather Data: https://data.open-power-system-data.org/weather_data/2020-09-16
2. **C++ Reference** (for STL algorithms): https://en.cppreference.com/