CM2005 Object Oriented Programming

Midterm assignment

1 INTRODUCTION

In this report, I will describe how I used the technical analysis toolkit for weather data in this report. Calculating candlestick data, producing a text-based plot of the data, filtering data and using text to plot, and predicting data and plotting it in a graph were the four main objectives that the project focused on. I have included a screenshot and a description of the output with my explanations of the reasoning and techniques I used to complete each task.

2 Compute Candlestick Data (Task 1)

I was told to use the provided dataset (weather_data_EU_1980-2019_temp_only.csv) to calculate the open, closing, high, and low values for temperatures in European nations. To manage the computation, I wrote a function in the computation_weather_data class called compute_candlestick_yearly(). The application will group temperature data by year and compute the open, close, high, and low values when the user enters a country code. Candlestick objects will be used to store the results.

2.1 How the Task Was Carried Out

To compute candlestick data, I have first extracted the temperature data based on the user provided country code. Then, I group the data by year.

Group Temperature by Year Code (CSVReader.cpp):

```cpp
std::map<int, std::vector<double>> CSVReader::group_temp_year(
    const std::vector<std::string>& dataset, int col_Index) {
    // Store temperatures grouped by year
    std::map<int, std::vector<double>> yearly_temps;

    for (const auto& line : dataset) {
        // Tokenise using ","
        auto tokens = CSVReader::tokenise(line, ',');
        if (tokens.size() <= col_Index) continue;

        // Extract the year from the timpstamp
        int year = std::stoi(tokens[0].substr(0, 4));

        // Convert the temperature value to number
        double temperature = std::stod(tokens[col_Index]);

        // Push the temperature to the particular year
        yearly_temps[year].push_back(temperature);
    }

    return yearly_temps;
}
```

For each year, the following calculations were performed:

- **Open**: The average temperature of the previous year.
- **Close**: The average temperature of the current year.
- **High**: The highest temperature recorded for that year.
- **Low**: The lowest temperature recorded for that year.

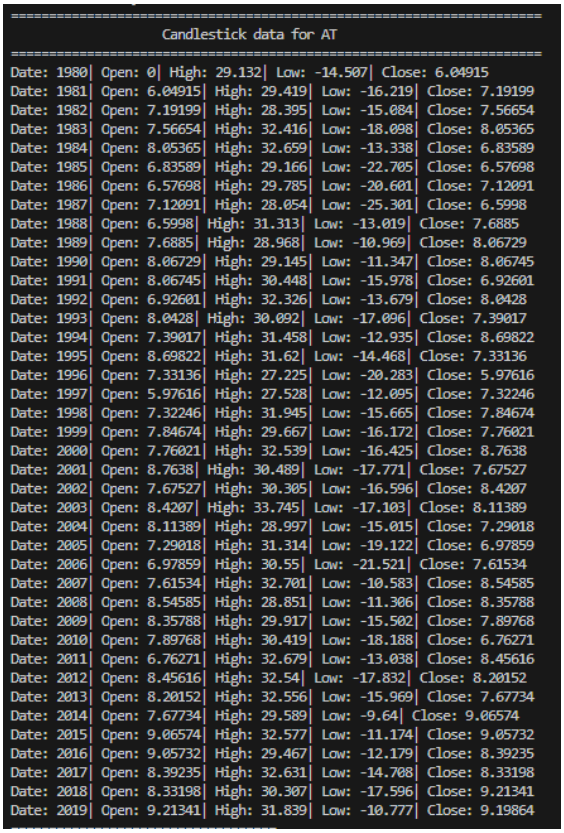
Computation Code (Candlestick.cpp):

```cpp
Candlestick Candlestick::compute(const std::string& date, const std::vector<double>& temperature, double previousClose) {
    // Open value: Average temperature from the previous time frame
    double open = previousClose;

    // High value: Highest temperature
    double high = *std::max_element(temperature.begin(), temperature.end());

    // Low value: Lowest temperature
    double low = *std::min_element(temperature.begin(), temperature.end());

    // Close value: Average temperature for the current time frame
    double close = std::accumulate(temperature.begin(), temperature.end(), 0.0) / temperature.size();

    return Candlestick(date, open, high, low, close);
};
```

A structured format was used to store the candlestick data, which was then output to the console.

2.2 Output:

Computed Candlestick for AT:



```
============================================================
                Candlestick data for AT
============================================================
Date: 1980| Open: 0| High: 29.132| Low: -14.507| Close: 6.04915
Date: 1981| Open: 6.04915| High: 29.419| Low: -16.219| Close: 7.19199
Date: 1982| Open: 7.19199| High: 28.395| Low: -15.084| Close: 7.56654
Date: 1983| Open: 7.56654| High: 32.416| Low: -18.098| Close: 8.05365
Date: 1984| Open: 8.05365| High: 32.659| Low: -13.338| Close: 6.83589
Date: 1985| Open: 6.83589| High: 29.166| Low: -22.705| Close: 6.57698
Date: 1986| Open: 6.57698| High: 29.785| Low: -20.601| Close: 7.12091
Date: 1987| Open: 7.12091| High: 28.054| Low: -25.301| Close: 6.5998
Date: 1988| Open: 6.5998| High: 31.313| Low: -13.019| Close: 7.6885
Date: 1989| Open: 7.6885| High: 28.968| Low: -10.969| Close: 8.06729
Date: 1990| Open: 8.06729| High: 29.145| Low: -11.347| Close: 8.06745
Date: 1991| Open: 8.06745| High: 30.448| Low: -15.978| Close: 6.92601
Date: 1992| Open: 6.92601| High: 32.326| Low: -13.679| Close: 8.0428
Date: 1993| Open: 8.0428| High: 30.092| Low: -17.096| Close: 7.39017
Date: 1994| Open: 7.39017| High: 31.458| Low: -12.935| Close: 8.69822
Date: 1995| Open: 8.69822| High: 31.62| Low: -14.468| Close: 7.33136
Date: 1996| Open: 7.33136| High: 27.225| Low: -20.283| Close: 5.97616
Date: 1997| Open: 5.97616| High: 27.528| Low: -12.095| Close: 7.32246
Date: 1998| Open: 7.32246| High: 31.945| Low: -15.665| Close: 7.84674
Date: 1999| Open: 7.84674| High: 29.667| Low: -16.172| Close: 7.76021
Date: 2000| Open: 7.76021| High: 32.539| Low: -16.425| Close: 8.7638
Date: 2001| Open: 8.7638| High: 30.489| Low: -17.771| Close: 7.67527
Date: 2002| Open: 7.67527| High: 30.305| Low: -16.596| Close: 8.4207
Date: 2003| Open: 8.4207| High: 33.745| Low: -17.103| Close: 8.11389
Date: 2004| Open: 8.11389| High: 28.997| Low: -15.015| Close: 7.29018
Date: 2005| Open: 7.29018| High: 31.314| Low: -19.122| Close: 6.97859
Date: 2006| Open: 6.97859| High: 30.55| Low: -21.521| Close: 7.61534
Date: 2007| Open: 7.61534| High: 32.701| Low: -10.583| Close: 8.54585
Date: 2008| Open: 8.54585| High: 28.851| Low: -11.306| Close: 8.35788
Date: 2009| Open: 8.35788| High: 29.917| Low: -15.502| Close: 7.89768
Date: 2010| Open: 7.89768| High: 30.419| Low: -18.188| Close: 6.76271
Date: 2011| Open: 6.76271| High: 32.679| Low: -13.038| Close: 8.45616
Date: 2012| Open: 8.45616| High: 32.54| Low: -17.832| Close: 8.20152
Date: 2013| Open: 8.20152| High: 32.556| Low: -15.969| Close: 7.67734
Date: 2014| Open: 7.67734| High: 29.589| Low: -9.64| Close: 9.06574
Date: 2015| Open: 9.06574| High: 32.577| Low: -11.174| Close: 9.05732
Date: 2016| Open: 9.05732| High: 29.467| Low: -12.179| Close: 8.39235
Date: 2017| Open: 8.39235| High: 32.631| Low: -14.708| Close: 8.33198
Date: 2018| Open: 8.33198| High: 30.307| Low: -17.596| Close: 9.21341
Date: 2019| Open: 9.21341| High: 31.839| Low: -10.777| Close: 9.19864
------------------------------
```

3 Create a Text-Based Plot (Task 2)

I made an easy text-based graphic using the candlestick data for the task 2. Before I started working on it, I researched the candlestick plot. Before starting, I created the characters, and the example graph I wanted to see as an output using a text editor. I have pasted my work below.

Workings:

```
Characters:
C - represent close value
O - represent open value
H - represent high value
L - represent low value
X - the connector between all the symbols

Sample Graph:
10|
9 |
8 |  H
7 |  X
6 |  O
5 |  X
4 |  X
3 |  X
2 |  X
1 |  C
0 |  X
-1|  X
-2|  X
-3|  X
-4|  L
-5|
--------------
    Year
```

Next, I made a function in the computation_weather_data class called text_plot_yearly(). The function enabled users to create plots for each country separately and enter multiple country codes separated by commas. The plot used H to represent the high value, L to represent the low value, C to represent the close value, O to represent the open value, and X act as the connector between these symbols, as can be observed from my previous workings.

3.2 How the Task Was Carried Out

The function first extracted the relevant candlestick data after verifying user input for country codes. By iterating over each year and using ASCII characters to visualize the high-low and open-close ranges, a text-based representation was produced using this data. The steps listed below are carried out for every country code:

1. **Input Parsing**: The input is split into individual country codes and trimmed for extra spaces.

   Code taken from CountryCode_processing() (CSVReader.cpp):

```
// Split input into different country codes
std::vector<std::string> countryCodes;
std::string delimiter = ",";
int pos = 0;

// Extract country codes
while ((pos = input.find(delimiter)) != std::string::npos) {
    countryCodes.push_back(input.substr(0, pos));
    input.erase(0, pos + delimiter.length());
}
countryCodes.push_back(input);

// Process each country code
for (const std::string &countryCode : countryCodes)
{
    // Trim extra spaces from the country code
    std::string trimmedCountryCode = countryCode;
    trimmedCountryCode.erase(trimmedCountryCode.find_last_not_of(" \t\n\r\f\v") + 1);
```

2. **Data Extraction**: For each country code, the temperature data column is identified in the dataset.

   Code taken from CountryCode_processing() (CSVReader.cpp):

```
std::string col_Name = countryCode + "_temperature";
int col_Index = CSVReader::Col_Index(col_Name);
```

3. **Candlestick Data Generation**: Grouped yearly temperature function to group the year data together and compute candlestick function is used to compute candlestick data, which includes open, high, low, and close values.

   Code taken from text_plot_yearly() (WeatherData.cpp):

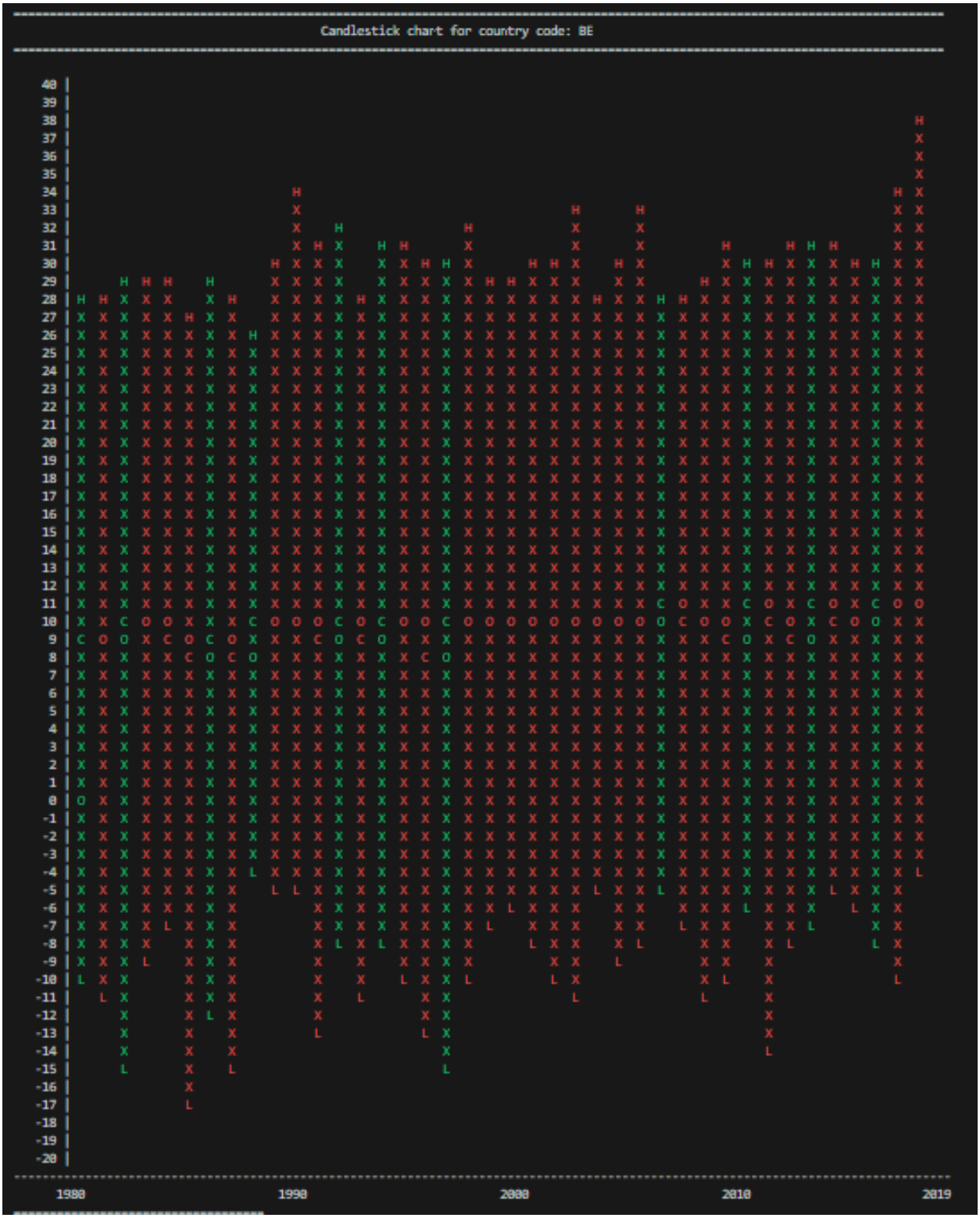
```
auto Temps_yearly = CSVReader::group_temp_year(dataset, col_index);
auto candlesticks = CSVReader::computeCandlesticks(Temps_yearly);
```

4. **Chart Generation**: A text-based candlestick chart is created using normalized y-axis values and visual symbols for open, close, high, and low temperatures. If the open value is more than close value, the plot is coloured as red. If the close value is more than open value the plot is coloured as green.

3.2 OUPUT:

The output shows text-based plot for the country BE from 1980 to 2019.

Text-Based plot for BE:


Candlestick chart for country code: BE
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-11
-12
-13
-14
-15
-16
-17
-18
-19
-20
1980
1990
2000
2010
2019

4 Filter Data and Plot (Task 3)

I had to plot a text-based graph and give at least two filter data options. I chose country and year range as my two filter choices. To process these filters and produce the plots, I wrote a function named text_plot_Filters().

4.2 How the Task Was Carried Out

The function first asks the user for the year range, the specified nations, and the filter criteria. To view the plot, the user can enter a minimum year range of 1980 and a maximum year range of 2019. The plot would be empty if the user entered the incorrect range.

Code taken from text_plot_Filters() (WeatherData.cpp):

```
// User input for country codes
std::cout << "Enter country codes (comma-separated): ";
std::string input;
std::cin.ignore();
std::getline(std::cin, input);

// User input for year range
std::string yearRange;
std::cout << "Enter the year range (e.g., 1980-1983): ";
std::cin >> yearRange;
```

Then, the filters will be processed, the program will extract the relevant set of data and plotted it using the text-based method described in Task 2.

Code taken from text_plot_Filters() (WeatherData.cpp):

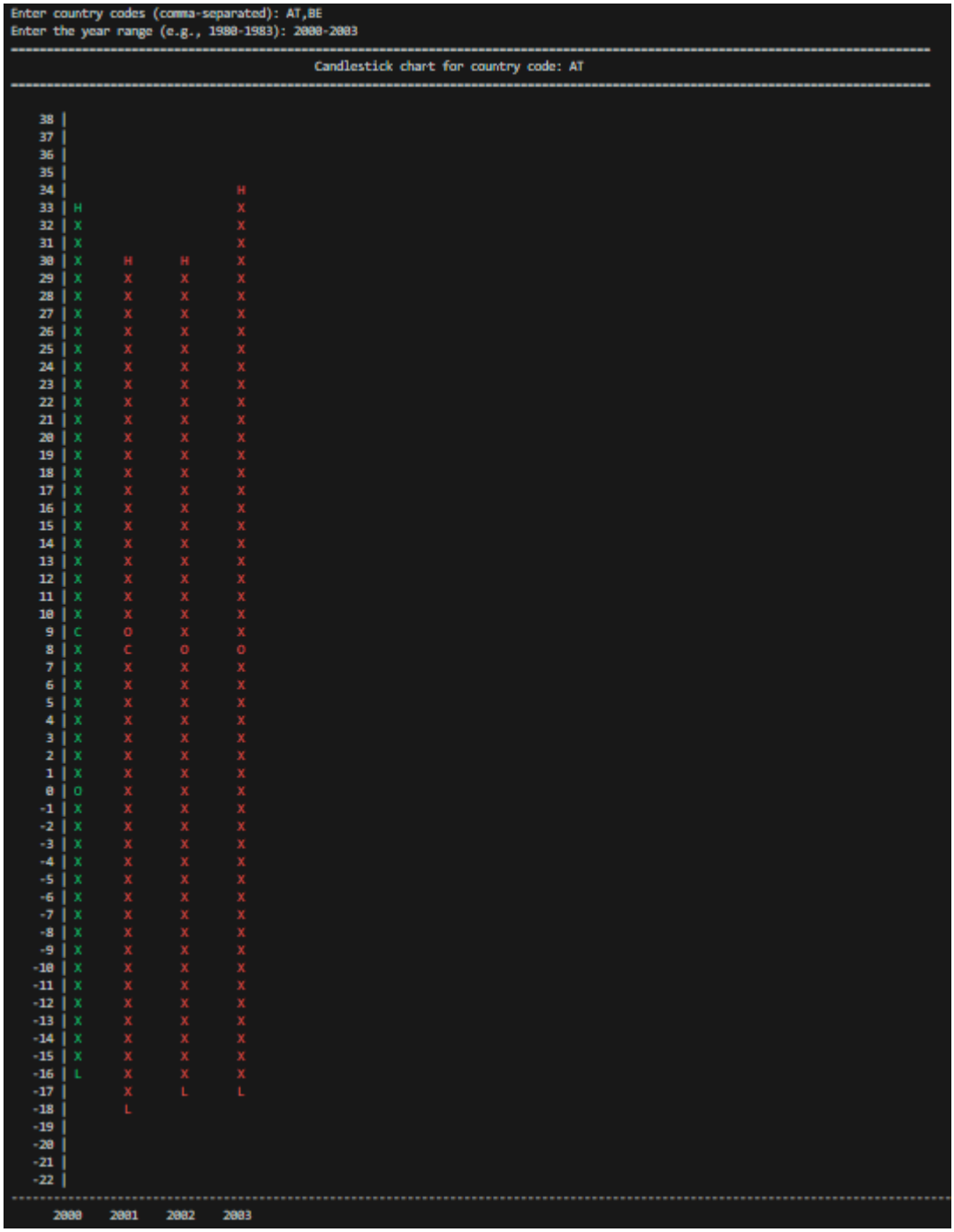
```
std::map<std::string, int> code_to_index = CSVReader::CountryCode_processing(input);

// Iterate over processed country codes and their column indices
for (const auto& pair : code_to_index) {
    const std::string& countryCode = pair.first;
    int col_index = pair.second;

    // Group temperatures by year within the range
    auto Temps_yearly = CSVReader::group_temp_year_range(dataset, col_index, startYear, endYear);

    // Compute candlesticks from grouped temperatures
    auto candlesticks = CSVReader::computeCandlesticks(Temps_yearly);
```

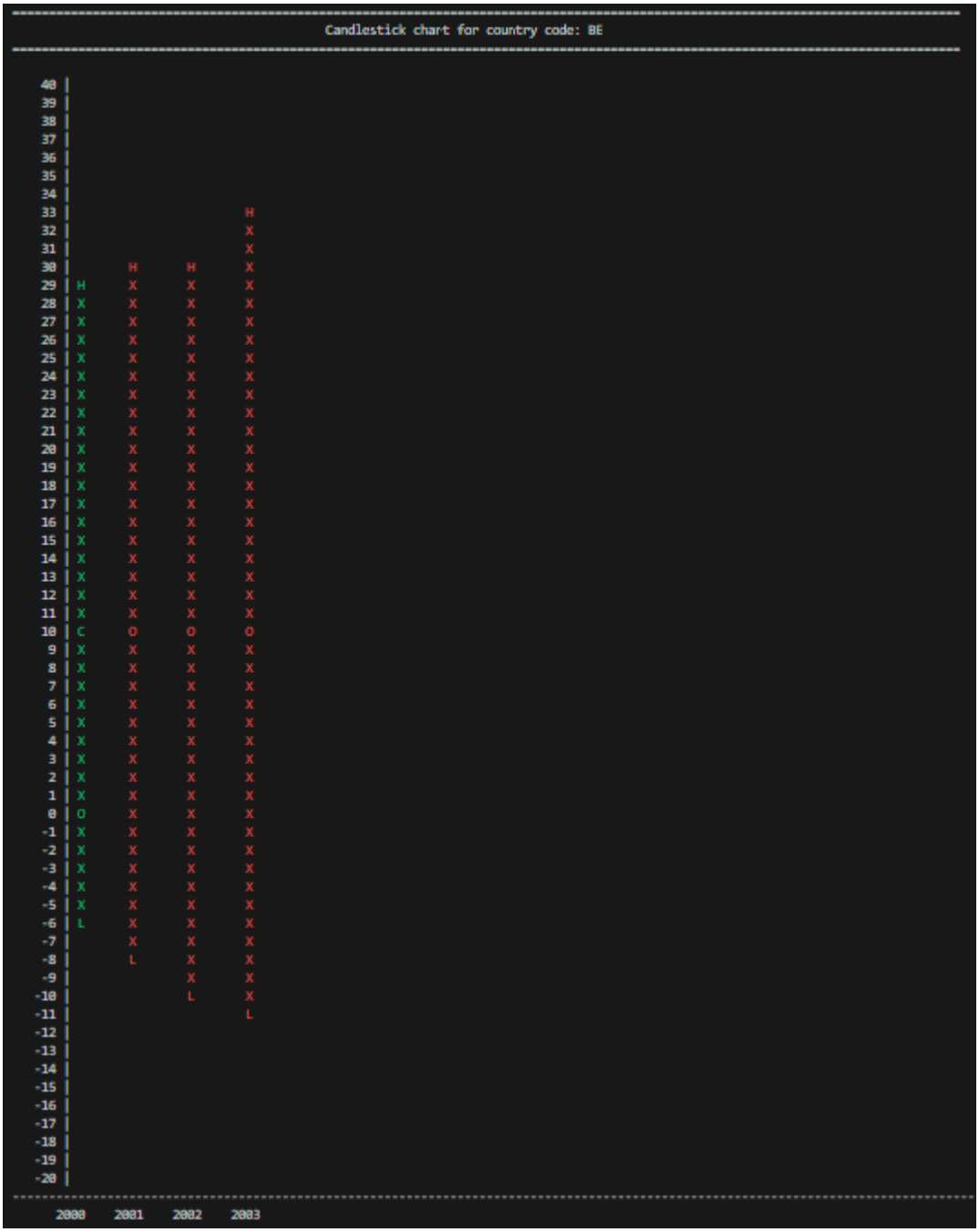
Users got to focus on data parts for improved analysis and visualisation.

4.2 OUTPUT:

The output shows text-based plot for country code of AT and BE for year range from 2000 to 2003 only.

Text-Based plot for AT & BE between 2000-2003:




Candlestick chart for country code: BE
40 |
39 |
38 |
37 |
36 |
35 |
34 |
33 | H
32 | X
31 | X
30 | H H X
29 | H X X X
28 | X X X X
27 | X X X X
26 | X X X X
25 | X X X X
24 | X X X X
23 | X X X X
22 | X X X X
21 | X X X X
20 | X X X X
19 | X X X X
18 | X X X X
17 | X X X X
16 | X X X X
15 | X X X X
14 | X X X X
13 | X X X X
12 | X X X X
11 | X X X X
10 | C O O O
9 | X X X X
8 | X X X X
7 | X X X X
6 | X X X X
5 | X X X X
4 | X X X X
3 | X X X X
2 | X X X X
1 | X X X X
0 | O X X X
-1 | X X X X
-2 | X X X X
-3 | X X X X
-4 | X X X X
-5 | X X X X
-6 | L X X X
-7 | X X X
-8 | L X X
-9 | X X
-10 | L X
-11 | L
-12 |
-13 |
-14 |
-15 |
-16 |
-17 |
-18 |
-19 |
-20 |
2000 2001 2002 2003

5 Predict Future Data (Task 4)

Task 4 wanted me to predict changes in temperature between a country and the year range of my choice. To save all of the processes for this section, I have written a function in the computation_weather_data class called Future_Prediciton(). Users were able to provide the number of years for predictions along with a country code. The software computed future averages by analysing historical data. As an output, predictions were shown.

5.2 How the Task Was Carried Out

The user will be asked for the country code and the number of years to predict at the beginning of the process. The user must enter 5 in the number of years they expect if they wish to make predictions for 2020–2024. If the user fails to understand, an example is provided.

Code taken from Future_Prediction() (WeatherData.cpp):

```
std::cout << "Enter the country code for future temperature prediction: ";
std::string countryCode;
std::cin >> countryCode;
```

```
std::cout << "Enter the number of years to predict (e.g., 6 for 2020 to 2025): ";
int predictionYears;
std::cin >> predictionYears;
```

The function validates the country code exists and the number of years was greater than zero.

Code taken from Future_Prediction () (WeatherData.cpp):

```
if (predictionYears <= 0) {
    std::cout << "Error: The number of years to predict must be greater than 0." << std::endl;
    return;
}
```

Next, data from 2015 to 2019 is extracted from the dataset as these are the last 5 years, and the yearly averages, highs and lows were calculated.

Code taken from Future_Prediction () (WeatherData.cpp):

```
std::vector<Candlestick> pastCandlesticks;
double totalHighDiff = 0, totalLowDiff = 0;
int count = 0;

// Generate candlesticks for 2015 to 2019
for (int i = 0; i < years.size(); i++) {
    if (years[i] >= 2015 && years[i] <= 2019) {
        // Temperature data for the year
        auto& temps = yearlyTemperatures[years[i]];
        double previousClose = (i == 0) ? 0.0 : averageTemperatures[years[i - 1]];
        Candlestick candle = Candlestick::compute(std::to_string(years[i]), temps, previousClose);

        // Difference in highs
        totalHighDiff += (yearlyHighs[years[i]] - yearlyHighs[years[i - 1]]);
        // Difference in low
        totalLowDiff += (yearlyLows[years[i]] - yearlyLows[years[i - 1]]);
        count++;

        pastCandlesticks.push_back(candle);
    }
}
```

A linear regression model was used to analyse the data values from 2015 to 2019 in order to predict future temperatures. The link between years and average temperatures was used by this model to compute a trendline. Future year temperatures could be predicted using the intercept and the line's slope (m).

Code taken from Future_Prediction () (WeatherData.cpp):

```
double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;
int n = years.size();

for (int i = 0; i < n; i++) {
    int x = years[i];
    // Average temperature for the year
    double y = averageTemperatures[years[i]];

    sumX += x;
    sumY += y;
    sumXY += x * y;
    sumX2 += x * x;
}

// Calculate the slope
double m = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
```

Using the averages and the regression model, predictions were calculated for the future years.

Code taken from Future_Prediction () (WeatherData.cpp):

```
std::vector<Candlestick> futureCandlesticks;
double previousClose = pastCandlesticks.back().close;

// Generate predicted candlesticks
for (int i = 1; i <= predictionYears; i++) {
    int futureYear = 2019 + i;
    // Predict average temperature
    double predictedTemperature = averageTemperatures[2019] + m * i;

    double open = previousClose;
    double close = predictedTemperature;

    // Predict high and low temperature
    double high = yearlyHighs[2019] + avgHighDiff * i;
    double low = yearlyLows[2019] + avgLowDiff * i;

    Candlestick futureCandle(std::to_string(futureYear), open, high, low, close);
    futureCandlesticks.push_back(futureCandle);

    previousClose = close;
}
```

Lastly, I plot a combined candlestick chart showing years from 2015 to 2019 and the future years. This chart visually represents temperature trends, making it easy to identify patterns and interpret predictions.

For 2015 to 2019 years:

- If open is more than close, the graph will be plotted as red.
- If close is more than open, the graph will be plotted as green.

For Future years:

- If open is more than close, the graph will be plotted as yellow.
- If close is more than open, the graph will be plotted as blue.

5.3 Model Justification and Description

The linear regression model and historical candlestick analysis are used to predict future temperature trends for a specific country code in weather data. Average, high, and low temperatures are computed for every year. In order to predict future average temperatures, a linear regression model uses these data points to calculate the temperature's rate of change over time. The model calculates the average changes in highs and lows during the last five years, from 2015 to 2019, in order to improve the precision of high and low temperature predictions. The open, close, high, and low temperatures for each year, as well as future candlestick data, are then predicted using these trends.

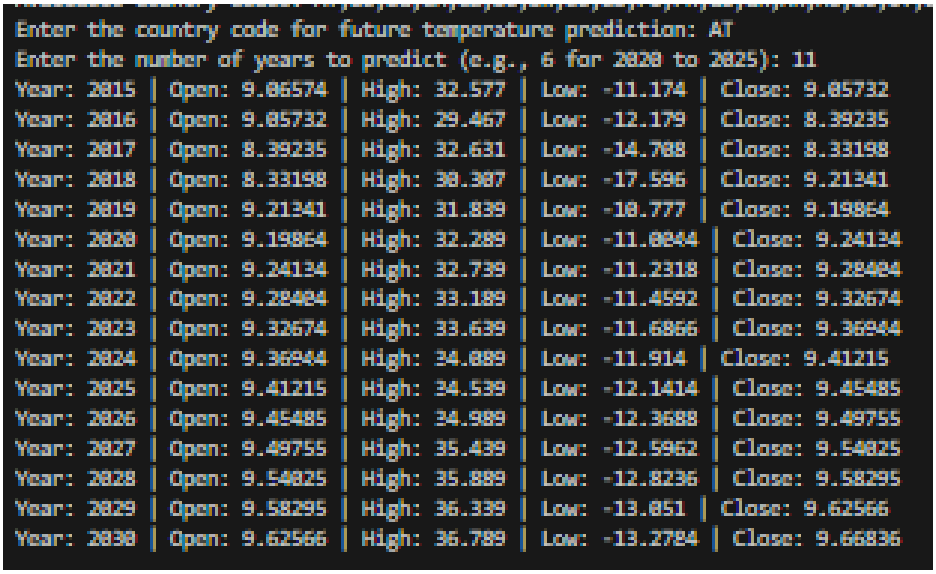
5.4 Originality and Challenge of Implementation

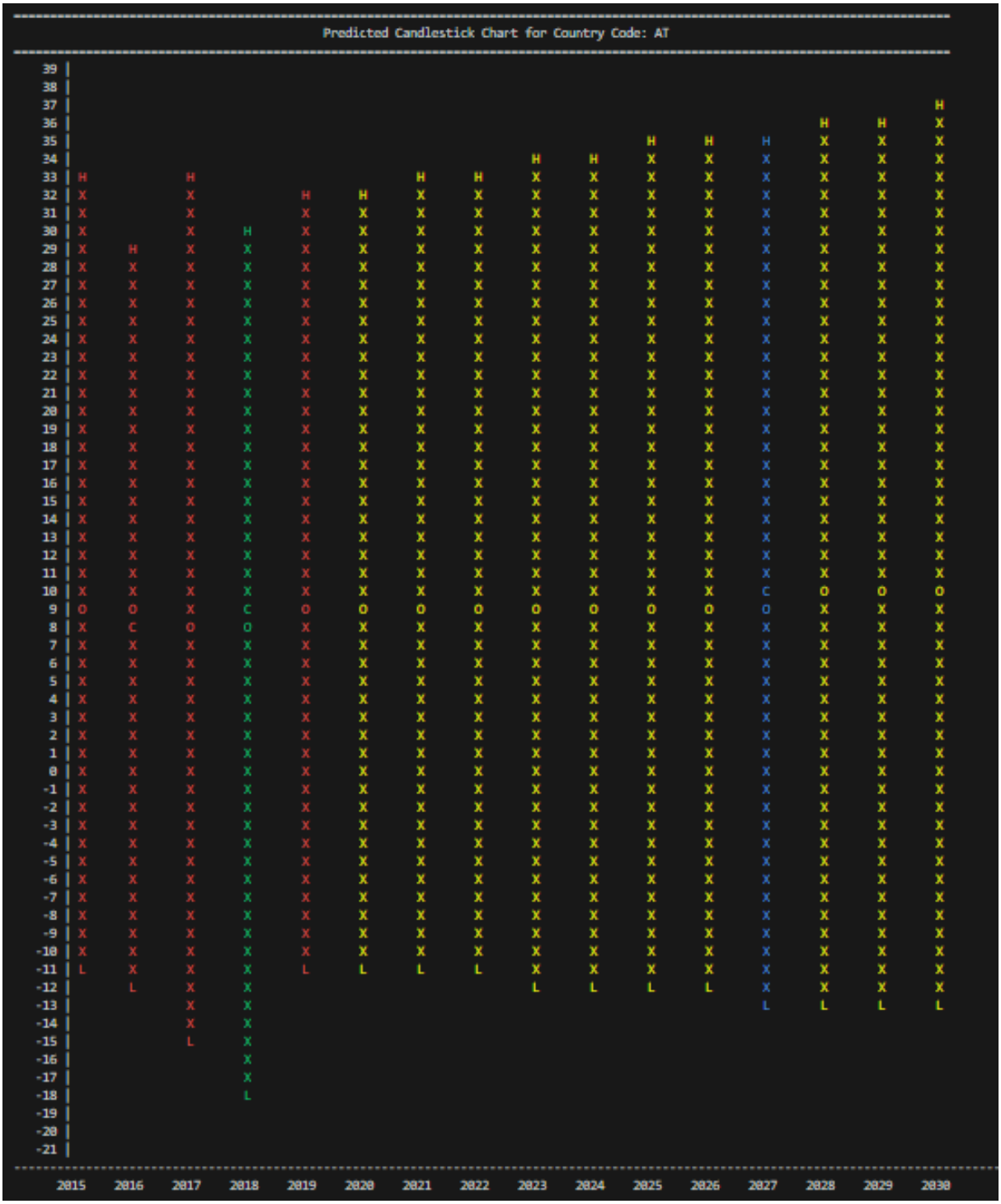
Problem-solving and creative thinking were required for this project. Adding predictions for the future and creating candlestick charts for temperature data required applying straightforward techniques to show complex trends. Making accurate and understandable predictions for users was the biggest problem. This demonstrates how the concept is both unique and useful.

5.4 Output:

The sample output shows the future prediction for AT country for 11 years range from 2020 to 2030. The first output shows the values in text format. The second output shows the values in a chart. As mentioned earlier, the current years will be displayed as red if open value is more than close or green if close value is more than open. The future years will be displayed as yellow if open value is more than close or blue if close value is more than open.

Future prediction for AT (11 years):





6 Conclusion

In conclusion, the goal of this project was to create a set of tools for weather analysis. The first task was computing data from candlesticks. The second task involved using text-based graphs for visualization. Adding filter options for the text-based graphs was task three. Predicting the future for a few chosen years was task four. Every one of the

four tasks was finished successfully. The prediction model that was selected worked well, and the implementation was both new and difficult. With supporting screenshots, this report provides an overview of how each activity was carried out and explains why it was selected.

7 Overall Experience

I had a positive experience throughout the project. The process was smooth, and challenges were resolved effectively. I learned a lot throughout the process. It was a great experience and would be useful for my future.

8 References

Vidhya, 2024. Linear Regression: A Comprehensive Guide [online]. Available at: https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/ [Accessed 13 December 2024]

Mitchell, C., 2024. Understanding Basic Candlestick Charts [online]. Available at: https://www.investopedia.com/trading/candlestick-charting-what-is-it/ [Accessed 18 December 2024]

Qu, K., 2024. Research on linear regression algorithm[online]. Available at: https://www.matec-conferences.org/articles/matecconf/pdf/2024/07/matecconf_icpcm2023_01046.pdf [Accessed 23 December 2024]