# Report

## Reading In The Data

**CsvReader class, TemperatureRow class - page 1 - 12 in the Code PDF**

I used the same layout and same Tokenize function for the CsvReader class as the Merkelrex project. I added a private static method to create a path from a csv file name. The csv file should be put into a folder called "data" next to the project files.

Each row in the data is converted into a TemperatureRow type that contains a timestamp and a vector of temperatures that have been converted to the double type. The data is read from the second row to leave off the header row and I've added a map of countries (keys) to their index 0-28 (values) which can be used to call a temperature from the vector of temperatures in the TemperatureRow.

The WeatherAnalyzerMain class init method calls the CsvReader::readcsv method to read in each row as a TemperatureRow into a vector of TemperatureRows.

## Task 1 - Compute the candlestick data

I computed the candlestick data for a chosen country for every year. To filter the data I iterate over the rows vector containing TemperatureRow objects using the TemperatureRow::getTempsByYear() method. This can be found on page 8 of the code document inside the TemperatureRow class. It uses the same concept as the Merkelrex app to create a currentYear from the year of the first row. If the currentYear equals the year in the TemperatureRow timestamp, the temperature at the chosen country column is added to a vector (temps). When the currentYear no longer matches, the temps vector is added to a map where the key is the currentYear and the value is a vector of every temperature for that country, for that year. The final output of this function - a map of each year as keys and every temperature for the chosen country, for that year in a vector as the value. Totaling 40 keys (years 1980 to 2019).

This output is used in the Statistics::calculateCandlesticks() method in the Statistics class. This can be found on page 18 of the code PDF. This method outputs a vector of Candlestick objects (page 12 code PDF) containing a year, open, close, high, and low. I compute the first year's candlestick separately because there is no previous year's close - the close for this year is the mean for this year. Then I iterate over the map from the second element - calculate the open as the previous year's close, the close as the current year's mean and take the highest and lowest values. These calculations are extracted into a private method of the Statistics class Statistics::getMeanHighLow().

To print the calculated candlesticks the user chooses option "3". Which does the following:

First: Calls the WeatherAnalyzerMain::getCandlestickData() method (page 35). This method calls 3 methods and returns a vector of candlestick objects:

1. WeatherAnalyzerMain::getUserInput() - gets the user inputs for country
2. TemperatureRow::getTempsByYear - described above
3. Statistics::calculateCandlesticks - described above

Second: Iterates over the vector of candlestick objects and prints the data

## Task 2 - Create a text based plot of the candlestick data

To create the candlestick plot choose option “4” or choose “Yes” after choosing option “3”. If you choose “yes” after option 3 the WeatherAnalyzerMain::plotCandlestickChart(std::vector<Candlestick> candlesticks) method is called (page 36). This function calls the Candlestick::getCandlestickChart(candlesticks) (page 14 code PDF) which builds a map where the keys are the y-axis temperatures in descending order and the values are each string line that will be printed representing the candlesticks for each year. To fit the y-axis on the screen I have rounded each temperature to the nearest integer. This method iterates through each year and its candlestick data points, rounds the data points and then iterates through the map of y-axis keys and builds a string using ANSI color codes,  unicode characters and/or empty spaces to represent the graph for each year. The returned map is printed to the screen.

If option “4” is chosen, the  WeatherAnalyzerMain::plotCandlestickChart() method is overloaded, this time it takes no parameters but makes a call to the WeatherAnalyzerMain::getCandlestickData() used in task 1 to get the data, then provides the user with an option to filter the year, and calls the WeatherAnalyzerMain::plotCandlestickChart(std::vector<Candlestick> candlesticks) method from above with the filtered years.

## Task 3 - Filter Data and Plotting using text

Choosing option “5” will print scatter plot data for the highs and lows for a particular country. It can be filtered by year (the same as the candlestick data) or by a day of the year. The WeatherAnalyzerMain::printScatterPlotData() method (page 38) is called which calls the WeatherAnalyzerMain::getScatterPlotData() method. This method calls 3 methods and returns a vector of ScatterPlotHighLow objects:

1. WeatherAnalyzerMain::getUserInput() - gets the user inputs for country and period filter (year or day of the year - if day of the year gets the month and day)
2. TemperatureRow::getTempsByYear - described above
3. Statistics::calculateCandlesticks - described above

The data is then printed to the screen.

Choosing option “6” prints a scatter plot. It calls the WeatherAnalyzerMain::plotScatterPlot() method which calls the same WeatherAnalyzerMain::getScatterPlotData() method from above, then filters the data by asking the user for a year range, and whether to plot high or low temperatures. The ScatterPlotHighLow::calculateScatterPlotHighs method builds the plot for highs and the ScatterPlotHighLow::calculateScatterPlotLows method builds the plot for lows. These are found on page 24-26 of the code PDF. The y-axis are calculated in the same way as the candlestick charts.

## Task 4 - Predicting data and plotting

I predict data from the scatter plot data in task 3. The user can predict a high or low temperature for any country, as either a yearly temperature or for a particular day. My strategy was to calculate the correlation coefficient strength between the years on the x-axis and the high or low temperatures on the y-axis. If the correlation is stronger - above 0.7 or below -0.7, then simple linear regression is used. A user is asked for the year they would like to predict the temperature for, and the year is used as the x value in the regression line formula.

If the correlation is weaker, a mean average is calculated from the historical data points and used as the prediction.

The WeatherAnalyzerMain::getPredictionTemp() method on page 40 calls the required methods and prints the prediction. First, it calls the WeatherAnalyzerMain::getScatterPlotData() method which asks the user to choose a country and a period filter. Then it asks the user to predict for high or low using the getUserHighLow() helper method on page 44. It then calls the Statistics::getCorrelationCoefficient() method on page 21. If the correlation is strong then the Statistics::getLinearRegressionPrediction() method on page 22 is called and the output is printed. Else the mean is predicted and printed.

## Originality and challenge of implementation

When the user enters option "2" they can retrieve any temperature from the data. The TemperatureRow::getRowIndex() method on page 7 searches the TemperatureRow objects in the rows vector. It searches with binary search for the timestamp using a timestamp built from user inputs and returns a row index. The WeatherAnalyzerMain::getTemperature() (page 34) retrieves the temperature using the row index and country column index and prints it out. I've also added an option to filter by day for the candlestick data and an option to print the chart immediately after viewing the data.