REPORT

ASSIGNMENT 1

SEPTEMBER 10, 2024

SHAHRON SHAJI
51781227

Table of Contents

1. BACKGROUND

In this assignment, we analyse the Global Historical Climatology Network daily (GHCNd) dataset using PySpark, focusing on key climatic variables such as temperature, precipitation, snowfall, and snow depth. The GHCNd dataset, maintained by the National Centres for Environmental Information (NCEI), offers a comprehensive archive of historical climate data collected from over 100,000 land surface stations across 180 countries. Our goal is to leverage PySpark's capabilities to explore patterns and trends in these variables across different regions and time periods, contributing to a better understanding of climate variability and change.

To achieve this, we preprocess the dataset to handle missing values, inconsistencies, and varying record lengths across different stations. We then utilize PySpark's data processing tools to perform exploratory data analysis, allowing us to identify trends and anomalies in temperature and precipitation data. The analysis involves working with large-scale data in a distributed computing environment, which provides the computational power needed for processing such vast amounts of information efficiently.

One of the main challenges encountered was managing the large volume of data and dealing with the variability in the periods of record across different stations. Understanding PySpark's distributed computing model was also crucial to ensure efficient data manipulation and analysis. The NCEI documentation and PySpark were invaluable resources in overcoming these difficulties, guiding us in both the handling of the dataset and the implementation of our analysis workflows.

2. PROCESSING

2.1. Q1

The directory of the GHCNd data stored in Hadoop Distributed File System (HDFS) at `hdfs:///data/ghcnd/` consist of multiple datasets, which includes daily observations, station metadata, state codes, country codes and inventory data. The `daily` dataset, which contains the climate records of every day, is the largest in size and covers various years, while the other datasets provide supplementary information, such as station details and geographic identifiers.

The directory structure can be visualized as a tree with one subdirectory and 4 other files. The main components are:

- ***daily/*** : This subdirectory contains 263 `.csv.gz` files where each file name is an year indicating the daily data of that particular year. The year starts from 1750, then a few years are skipped till 1763 and contains till 2024.
- ***ghncd-countries.txt*** : This text file consists of the code mapping for countries.
- ***ghncd-inventory.txt*** : This text file consists information about the availability of specific variables at each station over time.
- ***ghncd-states.txt*** : This text file consists of the code mapping for states.
- ***ghncd-stations.txt*** : This text file provides the metadata about each climate station, including location and identifiers.

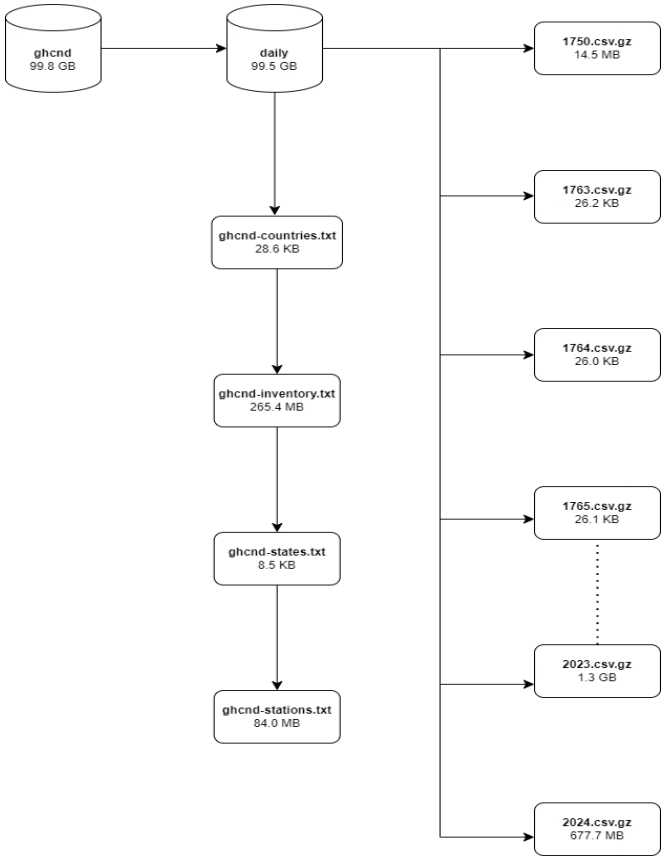

Fig 2.1. : Visualisation of directory with size

On going through the daily dataset, it can be observed that it spans through multiple decades, with the total size of the dataset increasing significantly over the years. There is a total of 263 years' worth data in the daily subdirectory and the size of each year's data varies, with more recent years generally having more entries, which can be due to the increase in the number of stations and probably better reporting practices. Fig 2.2. shows the visualization of data size change over time which indicates a growing volume of climate data being collected.

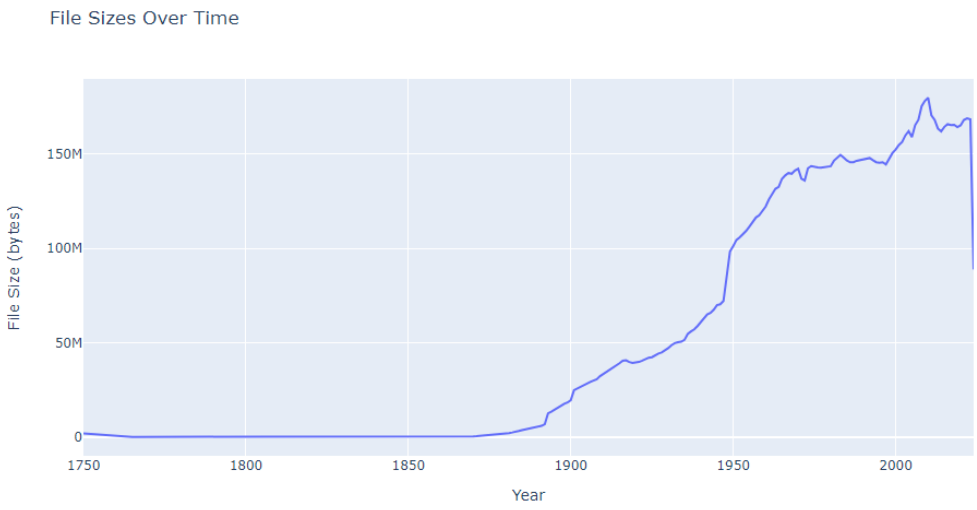

Fig 2.2.: File sizes over time

Name	Expanded Size	Compressed Size
ghcnd	99.8 GB	12.5 GB
daily	99.5 GB	12.4 GB
ghcnd-countries	28.6 KB	3.6 KB
ghcnd-inventory	265.4 MB	33.2 MB
ghcnd-states	8.5 KB	1.1 KB
ghcnd-stations	84.0 MB	10.5 MB

Table 2.1. : Main components and its sizes

The size of ghcnd dataset is 99.8 Gb, with the daily dataset comprising the majority which is 99.5 Gb. The table clearly shows that the daily dataset has a huge difference in size from the others. The stations, states, countries, and inventory datasets are relatively small, which works as a supplementary metadata to support the analysis of the daily dataset.

2.2. Q2

A proper schema was defined and applied to ensure a proper data handling, using a pretty straight forward method *StructField*. For the daily dataset, the following schema was defined:

- **ID**: *StringType()*
- **DATE**: *StringType()*
- **ELEMENT**: *StringType()*
- **VALUE**: *FloatType()*
- **MEASUREMENT_FLAG**: *StringType()*
- **QUALITY_FLAG***: StringType()*
- **SOURCE_FLAG**: *StringType()*
- **OBSERVATION_FLAG**: *StringType()*

These data types were selected to match the expected formats of each field. Initially, DATE column was created with *DateType()* to match the prerequisites of Apache Spark documentations, but this resulted in giving NaN values for all of date and ended up in using *StringType()*. After defining the schema, a sample of 1000 rows were loaded from 2023.csv.gz file using this schema.

Next, each of the auxiliary datasets, stations, states, countries, and inventory were loaded, but these required extra processing details to overcome the fixed-width text formatting. *spark.read.text* method combined with *pyspark.sql.functions.substring* was used to parse and extract the relevant fields.

A summary table of each metadata table is as follows:

Metadata Tables	Number of Rows
inv	756342
state	74
country	219
station	127994

Table 2.2. : Table of metadata with number of rows

To deal with the overlap between datasets, inv and station datasets were compared which revealed that some **ID**s in the inv table are not found in the station table. This indicates that there are no discrepancies in data or no missing entries in data. This was identified using *leftanti* join of inv with station.

3. STATIONS

3.1. Q1

To improve the station dataset with more information, several data transformations were performed on it and then joined with country, state and inventory data. This process helped to better understand the data coverage and completeness across different stations worldwide.

The two-character country code from each station ID in the station table were extracted using *pyspark.sql.functions.substring* method and then added it to the new column named COUNTRY_CODE using *withColumn* method. Using this COUNTRY_CODE as a reference, the country table was then joined to station table forming the combined_station dataset. The extra column called CODE which replicated the same data as COUNTRY_CODE was then dropped from the table to avoid duplication.

A normal left join was done for achieving this combination of combined_station table and state table, even state code was only available for stations within the United States. This join enriched the combined_station dataset with state names, but only for stations located in the U.S. Once the data was joined, the CODE column was dropped inorder to avoid redundancy. As the state names were in the column NAME and this was renamed to STATE_NAME for easy identification in the future.

To determine the first and last year that each station was active, the inv dataset was used. We extracted the minimum of FIRST_YEAR and the maximum of LAST_YEAR for each station using aggregation functions in the *groupBy* method. Once that was done, the column ID was renamed to STATION_IDDD inorder to recognise it in the future to drop it once combined with combined_station table.

To determine how many different elements has each station collected overall, *countDistinct* function is used along with *groupBy* method to get the result. Once again, ID was renamed to STATION_IDDD inorder to recognise it in the future to drop it.

To count the core elements, first the core elements were defined which were assumed as TMAX, TMIN, PRCP, SNOW and SNWD. The approach used was a straightforward method of marking each element of a station as whether core element or not and counting them. If it is a core element, the element will be placed as Core or otherwise placed as Other in a new column called ELEMENT_TYPE. After that ELEMENT_TYPE is pivoted as CORE and OTHER The total count of Core elements of each station will be placed in CORE and that of Other will be placed in OTHER. Furthermore NA values were filled with 0 and the ID was renamed to STATION_IDDD.

Once this was done, a simple *filter* method for CORE equal to 5 will give the number of stations that collected all five core elements. Inorder to calculate the number of stations that only collected precipitation and no other element, first ELEMENT value with PRCP is filtered out and then this is joined with the table obtained for counting the elements. This essentially helps us to filter out those with ELEMENT_COUNT only equal to one.

From this analysis, we found:

- The number of stations that collected all five core elements = 20482
- The number of stations that collected only precipitation (PRCP) = 16308

Once all the data was obtained, the next job was to integrate the obtained data with the combined_station table to obtain the enriched station data that will contain all station metadata, including their activity years, number of elements, recorded and specific element details. During each

stage of combining, STATION_IDDD was dropped from the table to avoid redundancy. This enriched dataset was then saved to the home directory in *parquet* format which is a columnar storage file format oprimized for performance and efficiency. It supports compression and is well-suited for large datasets.

Column Name	Description
ID	Unique Identifier for each station
LATITUDE	Latitude coordinate of the station
LONGITUDE	Longitude coordinate of the station
ELEVATION	Elevation of the station in meters
STATE	Two-character state code (if applicable, primarily for US-based stations)
STATION_NAME	Name of the station
GSN_FLAG	Indicator of whether the station is part of the Global Summary of the Day (GSN)
HCN_CRN_FLAG	Indicator of whether the station is part of the Historical Climatology Network (HCN) or Climate Reference Network (CRN)
WMO_ID	World Meteorological Organization identifier (if available)
COUNTRY_CODE	Two-character country code derived from the station ID
COUNTRY_NAME	Name of the country derived from the countries table
STATE_NAME	Name of the state derived from the states table (if applicable)
FIRST_YEAR_ACTIVE	First year of station activity recording any element
LAST_YEAR_ACTIVE	Last year of station activity recording any element
ELEMENT_COUNT	Total number of different elements data collected by the station
CORE	Number of core elements collected by the station
OTHER	Number of non-core elements collected by the station

Table 3.1.: Enriched Stations Table Schema

3.2. Q2

The methods used to join and find the missing values are pretty straightforward processes of using *left join* method and then finding the distinct null values from the ID column. It was relatively faster command to run for the daily_table with just 1000 rows but took a good amount of time for the entire daily observations in the complete daily dataset.

Performing a full *left join* between the daily dataset, which contains millions of rows spanning multiple years, and the stations dataset is computationally expensive. The cost of this operation involves multiple factors:

- Data Size
- Memory and Compute Resources
- Shuffle Operations

Given these factors, a full *left join* between daily_table and combined_station would be highly expensive in terms of both computational resources and time.

An efficient alternative approach would be to use *broadcast* join strategy combined with a distributed set difference. A possible method is to collect all unique station IDs from the stations dataset into a distributed data structure, such as a set, using Spark's collect function. Then broadcast the collected set of station IDs across all nodes. For each record in the daily dataset, check whether its station ID exists in the broadcasted set of IDs. If not, the record is from a station missing in stations. This approach avoids the need for a full table scan and join, significantly reducing the computational cost.

After implementing the initial join approach, a certain number of stations in the daily_table dataset were founded not listed in the combined_station dataset.

- For daily_table with a limit of 1000 rows, the number of stations that are not in daily_table is 127424.
- For entire daily observations, the number of stations that are not in daily_table is 8

4. DAILY

4.1. Q1

To determine the number of blocks required for the daily climate summaries for the years 2023 and 2024, we used the default HDFS block size of 134,217,728 bytes (approximately 128 MB) which was obtained from using the bash command *hdfs getconf -confKey "dfs.blocksize"*. Using these values to calculate the number of blocks, the following values were obtained:

- Number of blocks for 2024 daily data = 1
- Number of blocks for 2023 daily data = 2

The individual block sizes for 2023 daily data is 134,217,728 bytes for block 1 and 34,139,574 bytes for block 2.

When counting the number of observations in the 2023 and 2024 datasets, Spark executes a series of jobs, each containing stages that break down the process into smaller tasks.

File Path	File Size (Bytes)	Block Size (Bytes)	Number of Blocks	Number of Partitions	Number of Tasks
hdfs:///data/ghcnd/daily/2023.csv.gz	168,357,302	134,217,728	2	1	2
hdfs:///data/ghcnd/daily/2024.csv.gz	88,831,735	134,217,728	1	1	1

Table 4.1.: Summary of Relevant HDFS Command Outputs

Each job consists of multiple stages. In this case, both jobs (for 2023 and 2024 data) involve two stages:

1. Reading Data from HDFS (Stage 1): This stage reads the compressed CSV files from HDFS. The number of tasks is determined by the number of HDFS blocks, which is based on the file size and the default block size (134,217,728 bytes).

2. Count Rows (Stage 2): This stage performs a count operation on the DataFrame. The tasks are parallelized across the partitions created in the first stage.

A task is the smallest unit of work in Spark. Each task processes data from a single partition. For the 2023 dataset, 2 tasks were executed (corresponding to 2 blocks), and for the 2024 dataset, 1 task was executed (corresponding to 1 block).

When counting the number of observations in the 2023 and 2024 datasets, the number of tasks executed by Spark directly corresponds to the number of HDFS blocks in each input file. For the 2023 data, which consists of a single file (2023.csv.gz) with a size of 168,357,302 bytes, there are 2 HDFS blocks due to the default block size of 134,217,728 bytes, therefore executing 2 tasks. Whereas, the 2024 data file (2024.csv.gz) is smaller, at 88,831,735 bytes, fitting within a single HDFS block, resulting in 1 task being executed. Therefore, the number of tasks in each stage matches the number of blocks for the respective input files.

To restrict the daily dataset to only include data from the years 2014 to 2023 (inclusive), a wildcard pattern in the file path was used. The pattern specifies the files to be loaded, capturing just the required years of data. By using the *spark.read* method with this pattern, the resulting DataFrame, `daily_table_2014_2023`, contains all the observations from the required years.

When dealing with compressed files (like .gz), Spark handles them differently than uncompressed files. Compressed files like .gz are not splittable by nature. This means that Spark cannot split these files into smaller blocks for parallel processing. Spark treats each compressed file as a single block, and therefore, one task will be assigned to read the entire file. The number of tasks executed will generally correspond to the number of compressed files. For example, if you have 10 compressed files (.gz), Spark will create 10 tasks, each responsible for reading one complete file. Because compressed files cannot be split, each file will be read by a single executor, which may reduce parallelism compared to reading multiple uncompressed files where each file could be split into multiple blocks and processed in parallel.

Spark treats each compressed file (e.g., .gz files) as a single partition since they are not splittable. This means that for each .gz file, Spark will create a single task. On loading the daily data for multiple years from 2014 to 2023, and each year has one compressed .gz file, Spark will create 10 tasks (one per file). These tasks will run in parallel up to the number of available cores across your cluster's executors. The number of tasks running in parallel is limited by the number of .gz files and the number of cores available in the cluster. So for 10 files and a cluster with 4 cores, Spark will run 4 tasks in parallel, and the remaining 6 tasks will wait until a core becomes available.

To increase parallelism when loading and applying transformations in Spark, a change in the data storage format is one way. Convert data from compressed .gz files to a splittable format like plain CSV, Parquet, or Avro, which allows Spark to divide files into smaller chunks, increasing the number of tasks that can run in parallel. Additionally, repartitioning the data before storing it in HDFS can further enhance parallelism by creating more partitions that can be processed simultaneously.

An improve parallelism can also be achieved through Spark settings. Increasing the number of `spark.sql.shuffle.partitions` parameter, especially when dealing with large datasets, to allow more tasks to handle data shuffles. Allocating more executors and cores to your Spark job can also boost parallelism by enabling a greater number of tasks to run at once.

5. ANALYSIS

5.1. Q1

The methods used to get the count of these values are very straight forward with simple *filter* method and equating to the required conditions as asked. Only noteable one is the approach used for finding the countries in southern hemisphere which is just finding the countries with negative latitude. Once those are done, it is joined with country table and similar scenario is done for state table. These are then outputted to the output directory.

Category	Count
Total number of stations	127994
Stations that are still active	36516
Stations in GSN	991
Stations in HCN	1218
Stations in CRN	234
Stations in multiple networks	15
Stations in the southern hemisphere	25357
Stations in US Territories (excluding US)	399

Table 5.1.: Summary of the counts

5.2. Q2

The distance function implemented uses the Haversine formula to compute the geographical distance between two stations based on their latitude and longitude coordinates. This formula is well-suited for calculating distances between points on a sphere, such as the Earth, by taking into account its curvature. The function begins by setting the Earth's radius to 6,371 km and converts the latitude and longitude of both points from degrees to radians. It then calculates the differences in the latitude and longitude coordinates. The Haversine formula is applied to compute the central angle between the two points using the equation:

$$\mathrm{a} = \sin^2\left(\frac{\mathrm{diff\_lat}}{2}\right) + \cos(\mathrm{lat1\_rad}) \cdot \cos(\mathrm{lat2\_rad}) \cdot \sin^2\left(\frac{\mathrm{diff\_lon}}{2}\right)$$

Following this, the distance is calculated using:

$$c = 2 \cdot \mathrm{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

Finally, the distance between the two points is obtained with the formula:

$$\mathrm{distance} = R \cdot c$$

where R is the Earth's radius. This function effectively accounts for the Earth's spherical shape by using trigonometric functions that consider its curvature, ensuring an accurate distance measurement between two locations.

A cross join of the station_nz_1 table and station_nz_2 table has been done. Even though these are named as 1 and 2 of station_nz, these are both the same tables. This is done just to avoid the conflicts that will arise from the names and to avoid the error that occurs while exporting. One doing the cross joining and calculating the the respective distances, he two geographically closest stations in New Zealand are **Paraparaumu AWS** and **Wellington Aero AWS** with the distance between them being **50.53** (approx.).

5.3. Q3

To count the total number of rows in the daily_table dataset, we performed a simple row count operation using the *count()* method.

CATEGORY	COUNT
Total number of rows in daily_table	3119374043
PRCP	1073530896
TMAX	457927581
SNWD	299076145
SNOW	356187192
TMIN	456739567
TMAX without TMIN	10567304
Number of stations contributing to TMAX without TMIN	28716

Table 5.2.: Summary of counts in daily_table

From this table, it is evident that the element with the most observations is PRCP (precipitation).

Once the TMAX and TMIN elements were filtered out, it is then joined together using *leftanti.* This will essentially create a table that will only have TMAX values and not TMIN values in the ELEMENT section. Once that was done, a simple *count()* gave the required values. To get the unique stations that are contributing for this, the ID are selected using *select()* and then counted with *count()* combined with *distinct().*

6. VISUALIZATIONS

6.1. Q1

To visualize the daily climate summaries for all the stations in New Zealand for their maximum temperature and minimum temperature per day, the dataset was thoroughly filtered to attain the required plots. First only those records with ID as NZ was filtered and then further filtering for ELEMENT column for TMIN or TMAX. Doing so resulted in the desired dataset and the number of observations for TMIN and TMAX in New Zealand was **487760**. We can also observe that the dataset spans from 1966 to 2024 (inclusive), which means the dataset covers a total of 59 years.

Once that is done the next step is to create a time series plot of TMIN and TMAX for each station in New Zealand, which can provide insights into the temperature trend for each station. The filtered data was converted into a *Pandas DataFrame* inorder to do easy manipulation and visualisation. Then the TMAX and TMIN observations in the ELEMENT column is pivoted using *pivot_table* operation. The data is then grouped by station. The DATE column is then converted into YearMonth column which will consist of Year and month using *dt.to_period('M')* operation. The aim to do so is because, plotting the entire dataset gave a very noisy plot which couldn't really give any meaning to the plot. So once YearMonth was acquired, the average of the entire month was considered as the TMIN and TMAX value for that particular month while grouping the data for a better plot. Then the missing values were handled uing linear interpolation and subplots were created for each stations. The average time series for all stations were also plotted as separate. Seperately, a detailed plot of each station was also done so that the a better visualisation than the subplot can be obtained.

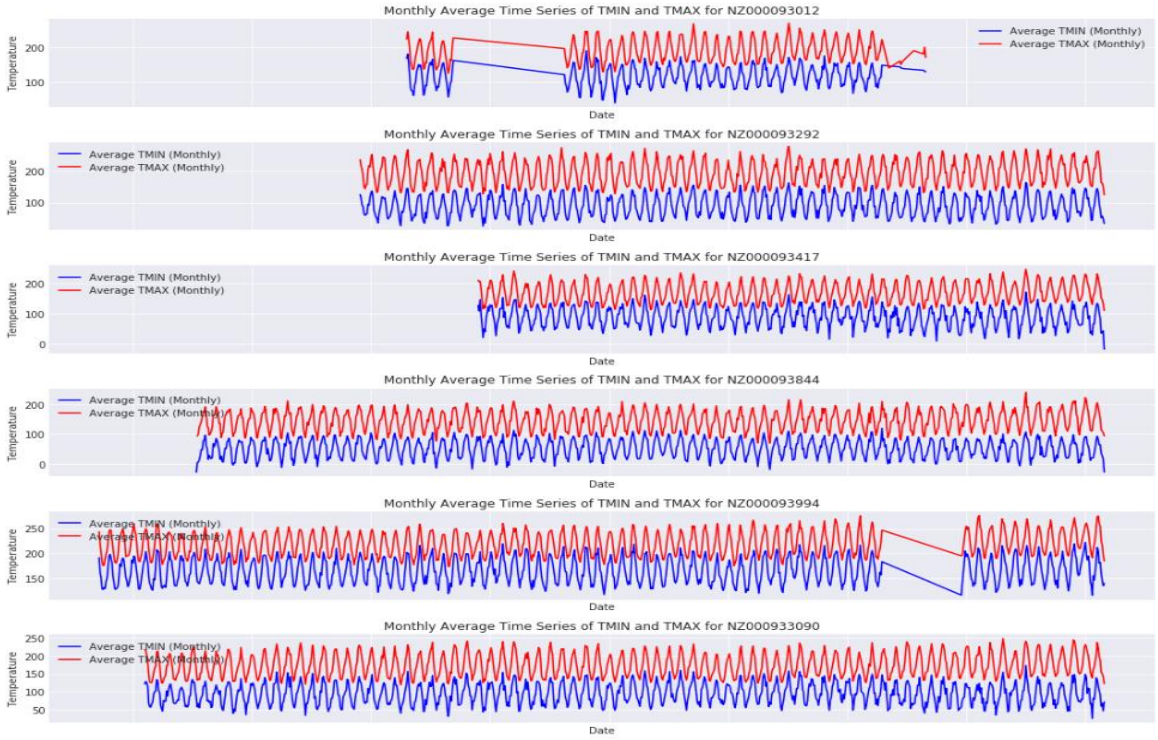

Fig 6.1.: Subplot of Time series of TMAX and TMIN of each stations in New Zealand

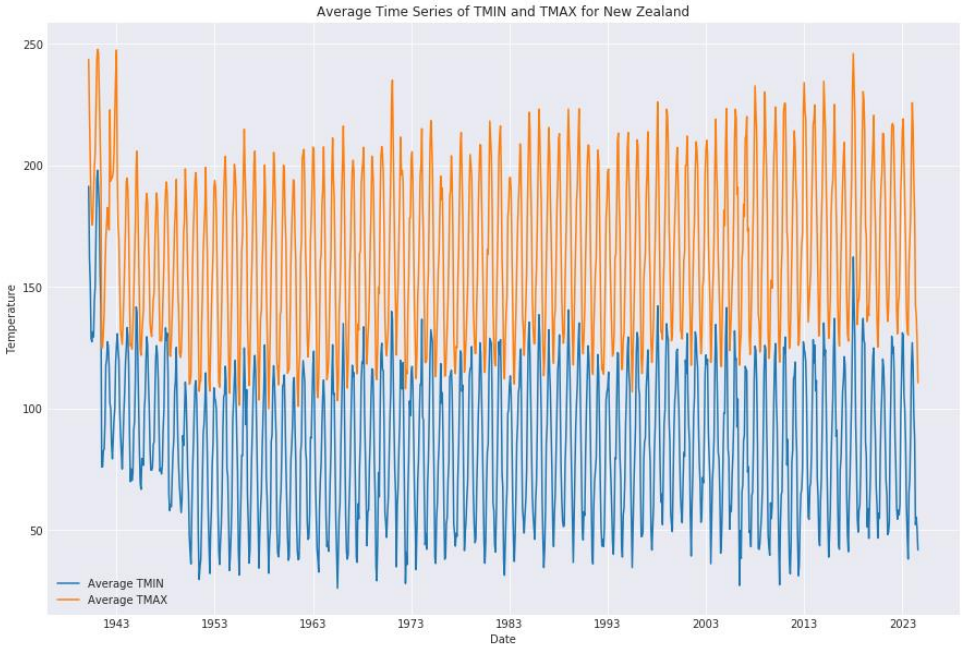

Fig 6.2.: Average Time series of TMAX and TMIN of all stations in New Zealand

The zig-zag plot probably represents the change in temperature over the seasons and also a lot of the stations seems to be relatively newer as the data is not available for a lot of previous years.

6.2. Q2

The analysis of precipitation data was done by grouping observations by year and country to compute the average yearly rainfall for each country. The data was filtered to include only observations where the ELEMENT column was PRCP (precipitation). The average yearly precipitation was then calculated

and descriptive statistics were generated. There was a total of **17548** yearly averages across different countries, with a mean of **43.77 mm per year**. A standard deviation of **88.66 mm** per year, indicating significant variability in yearly precipitation between different countries. The minimum value of **-1.08 mm** is physically implausible to achieve (suggesting errors or missing data) and the maximum value of **4361 mm** represents a huge amount of rainfall in a single year (suggesting to be an outlier or a localized extreme weather event). The country with the highest recorded average rainfall in a single year is Equatorial Guinea in the year 2000. Since this value has a deviation from the mean, it would be a good idea to examine the data quality for that specific year.

The visualisation of the average rainfall in the year 2023 for each country was done using a *choropleth map*. This type of visualization effectively represents geographical data by coloring each country based on its average rainfall, allowing for easy identification of patterns and anomalies across different regions. For this data visualisation, there was a bit of a challenge as the library geopandas wasn't able to be imported to use the 3 character coding system used by *choropleth map*. So inorder to overcome this diversity, the country names were joined with the average yearly rainfall data and then used this information to plot the *choropleth map*. Once that was done, the mean of the data for each country was calculated and used this information to plot the data.

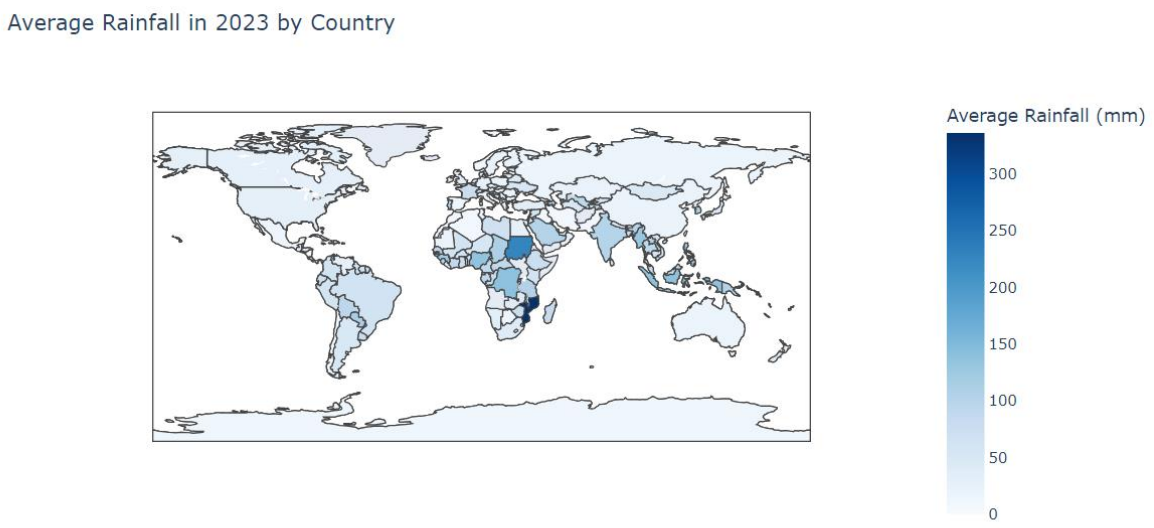

Fig 6.3.: Choropleth map of average rainfall in 2023 for each country

On closely observing the map, I could find some missing plots on North American Countries as well as some over the African countries.

7. CONCLUSION

In conclusion, the analysis of the GHCNd dataset using PySpark effectively identified key patterns and trends in climate variables across different regions and time periods. By leveraging PySpark's capabilities for handling large-scale data, we gained valuable insights into climate variability and change, demonstrating the importance of efficient data processing in environmental research.

8. REFERENCES

1. **National Centers for Environmental Information (NCEI), NOAA**:

   - Documentation on the Global Historical Climatology Network - Daily (GHCNd) dataset.
   - Available at: https://www.ncei.noaa.gov/products/land-based-station/global-historical-climatology-network-daily

2. **Apache Spark Documentation**:

   - Used for understanding PySpark and its data processing capabilities.
   - Available at: https://spark.apache.org/docs/latest/api/python/index.html

3. **ChatGPT**:

   - Used as a source for generating code snippets, explanations, and solutions throughout the assignment.

4. **Haversine Formula for Calculating Distances**:

   - Source: Haversine Formula - Great Circle Distance Between Two Points.
   - Available at: https://en.wikipedia.org/wiki/Haversine_formula
   - Description: Used to compute geographical distances between two points on the Earth's surface, taking into account its spherical shape.

5. **Plotly for Data Visualization**:

   - Source: Plotly Documentation.
   - Available at: https://plotly.com/python/
   - Description: Provides guidance and examples on creating interactive visualizations such as choropleth maps and time series plots.

6. **Pandas for Data Manipulation in Python**:

   - Source: Pandas Documentation.
   - Available at: https://pandas.pydata.org/pandas-docs/stable/
   - Description: Used for data manipulation, such as filtering, pivoting, and aggregating data before plotting.

7. **Matplotlib for Plotting in Python**:

   - Source: Matplotlib: Visualization with Python.
   - Available at: https://matplotlib.org/stable/contents.html
   - Description: A library used for creating static, animated, and interactive visualizations in Python.

8. **PySpark SQL Functions**:

   - Source:PySpark SQL Functions Documentation.
   - Available at: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html
   - Description: Used for applying SQL functions such as `filter`, `groupBy`, and `agg` to manipulate large-scale datasets in Spark.

9. APPENDICES

9.1. SUBPLOT OF TMAX AND TMIN

The following are the subplot of TMAX and TMIN values of all the stations of New Zealand for easy comparison.

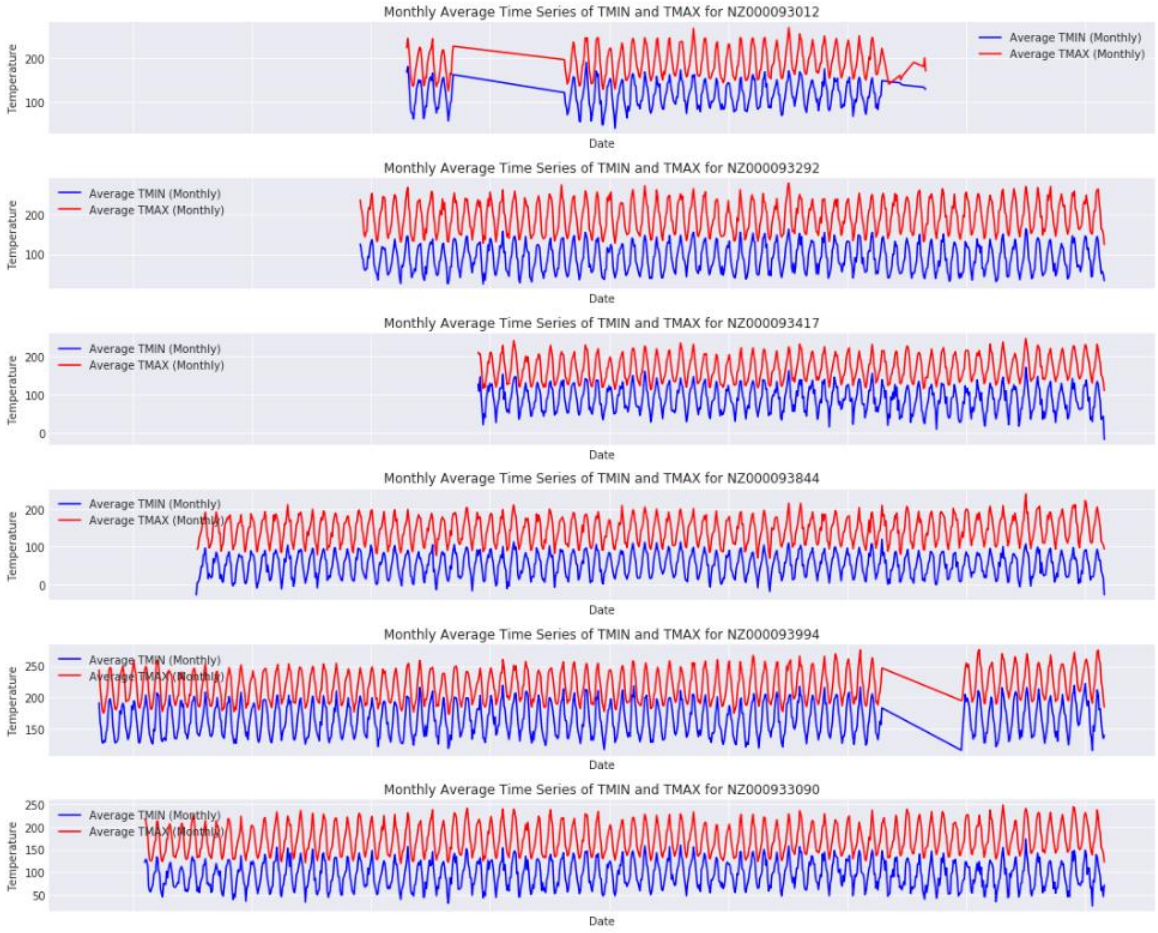

Fig 9.1.: Subplot of Time series of TMAX and TMIN of each stations in New Zealand

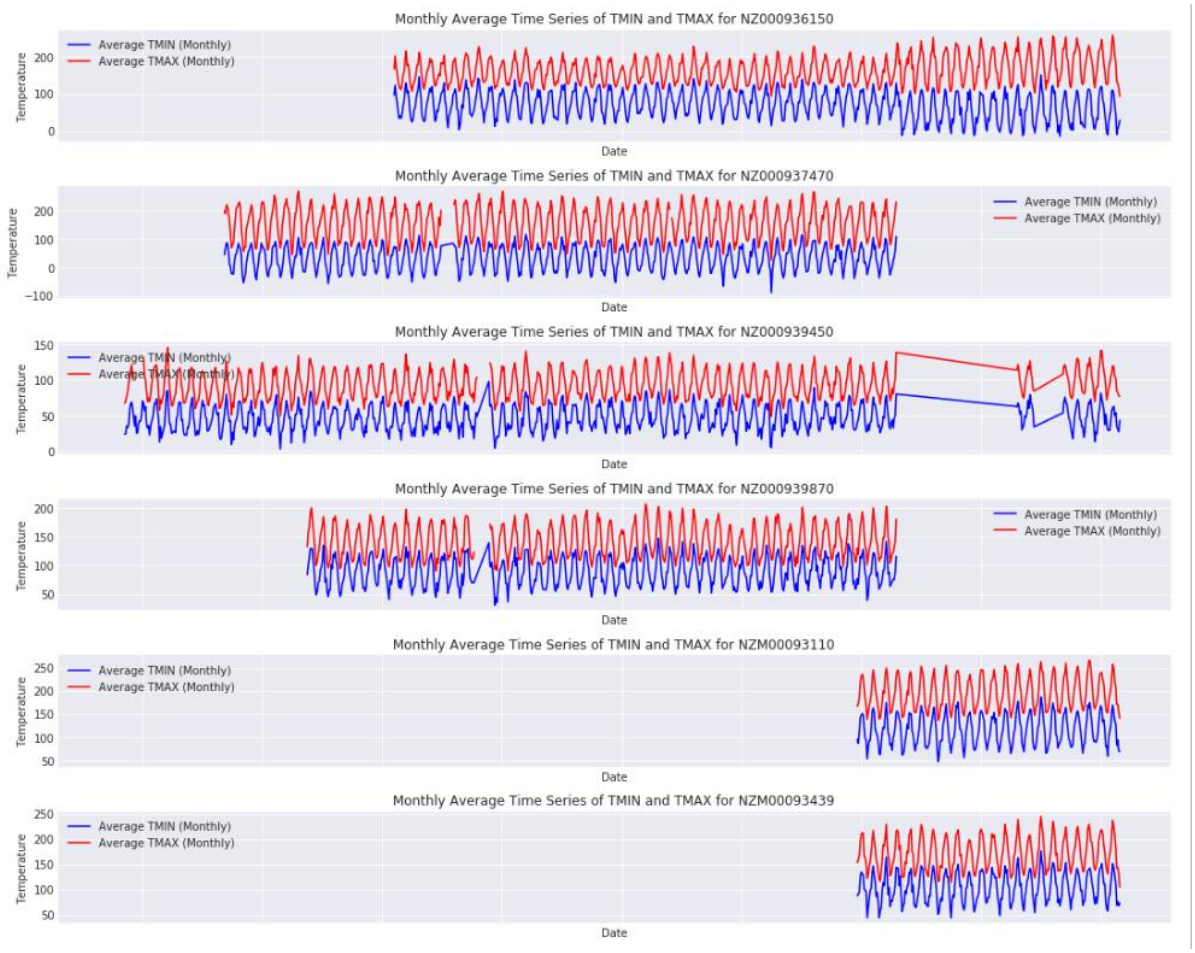

Fig 9.2.: Subplot of Time series of TMAX and TMIN of each stations in New Zealand

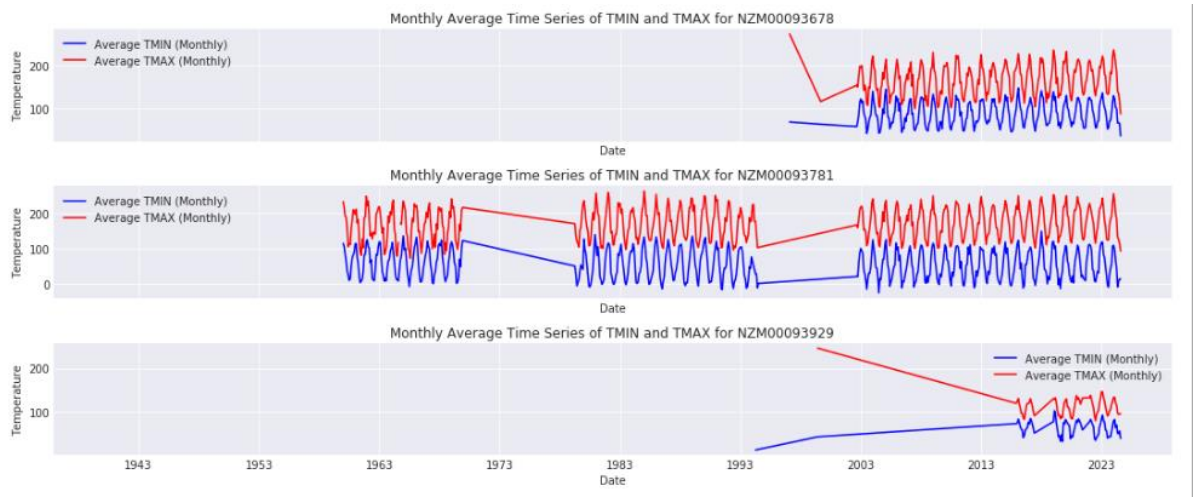

Fig 9.3.: Subplot of Time series of TMAX and TMIN of each stations in New Zealand

9.2. INDIVIDUAL PLOT OF TMAX AND TMIN

The following are the more detailed line plots of all the 15 stations in New Zealand showing the TMAX and TMIN values over the years. Orange represents TMAX and Blue represents TMIN.

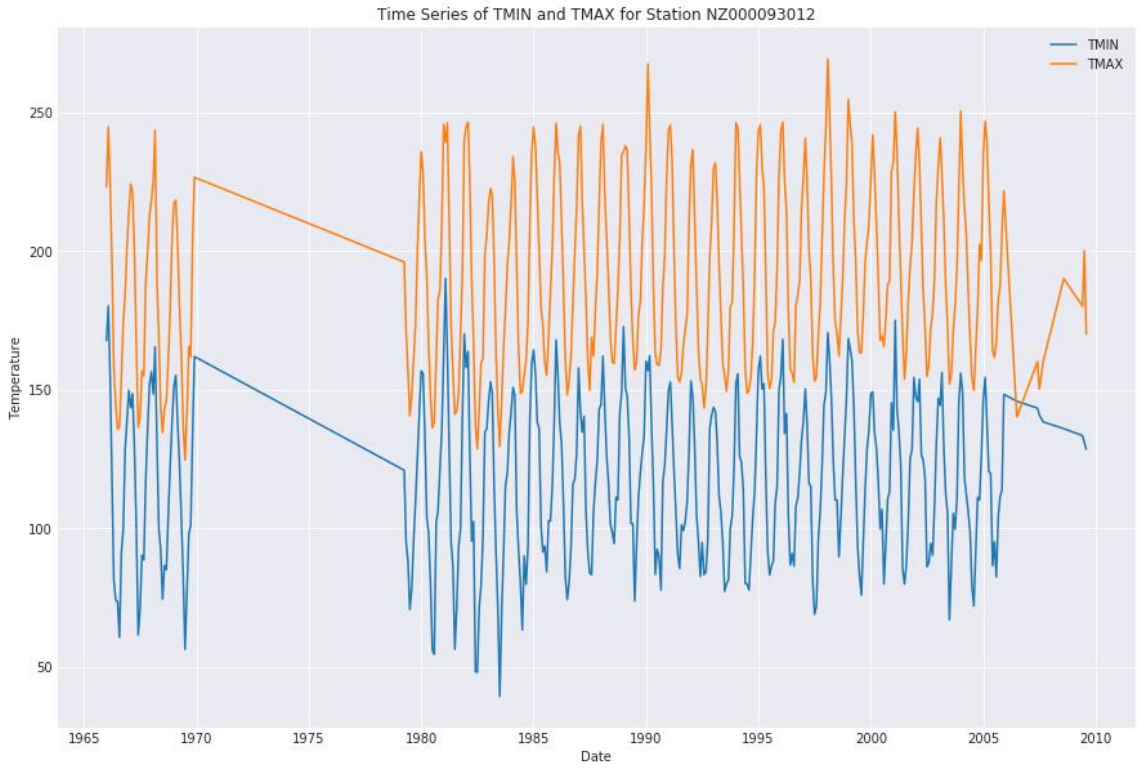

Fig 9.4.: Time series of TMIN and TMAX for Station NZ000093012

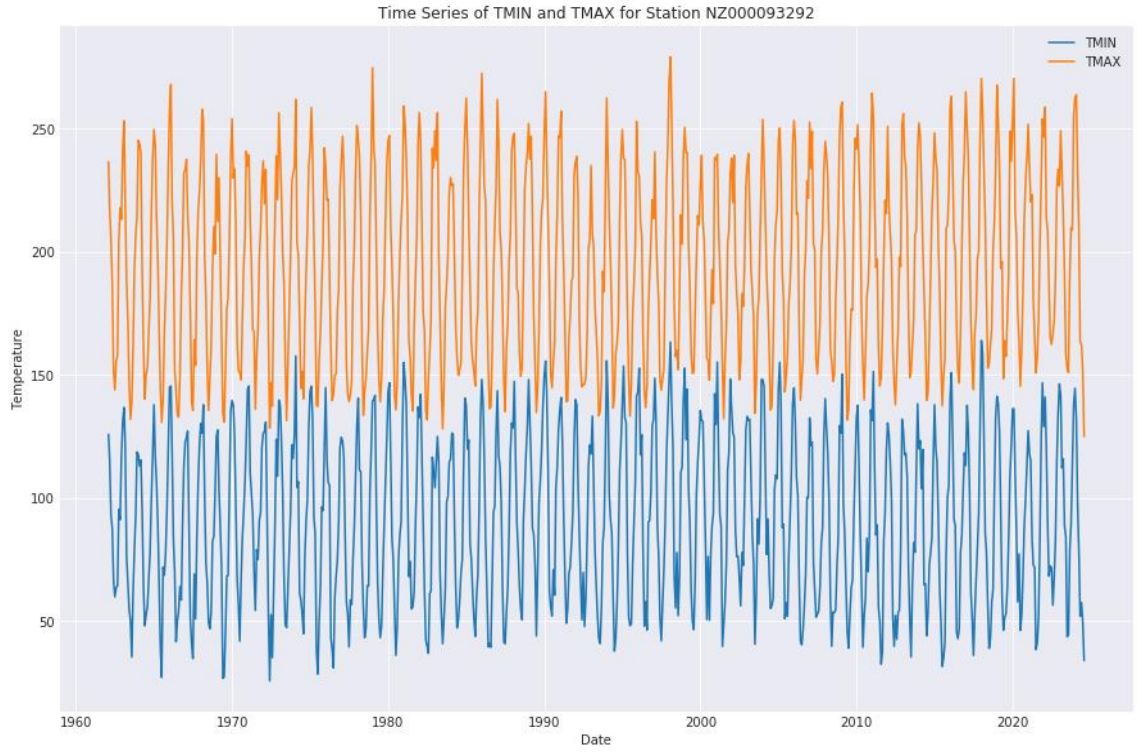

Fig 9.5.: Time series of TMIN and TMAX for Station NZ000093292

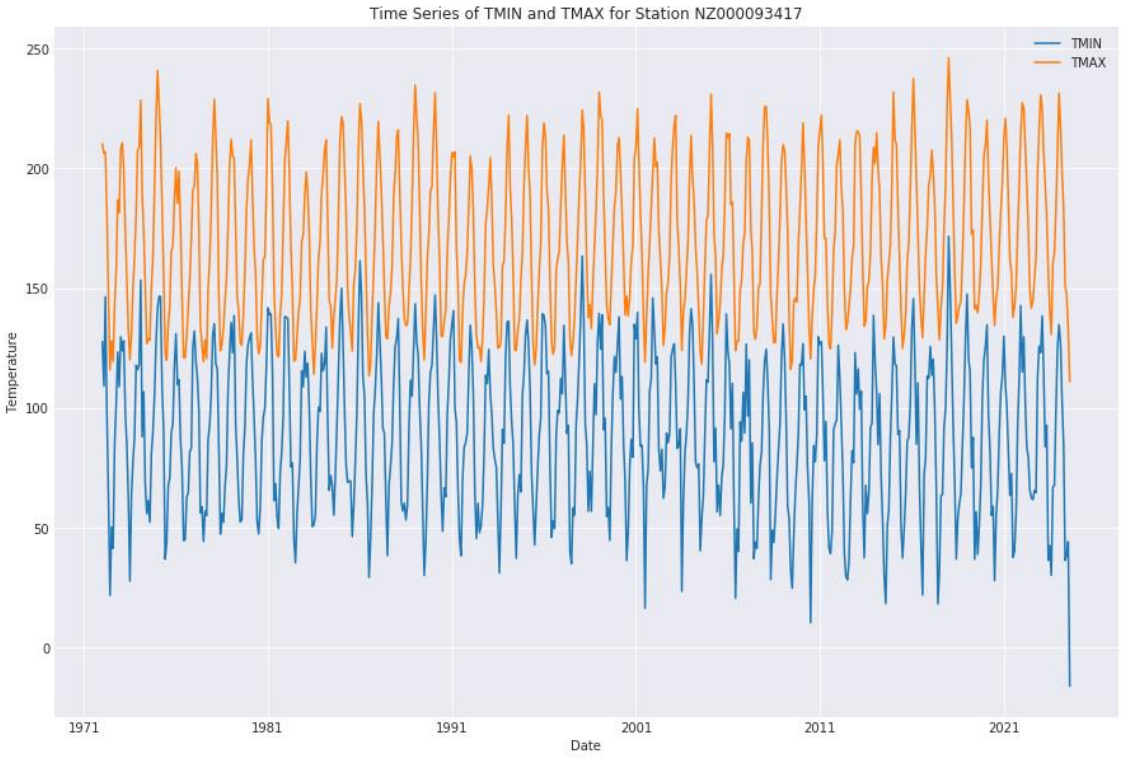

Fig 9.6.: Time series of TMIN and TMAX for Station NZ000093417

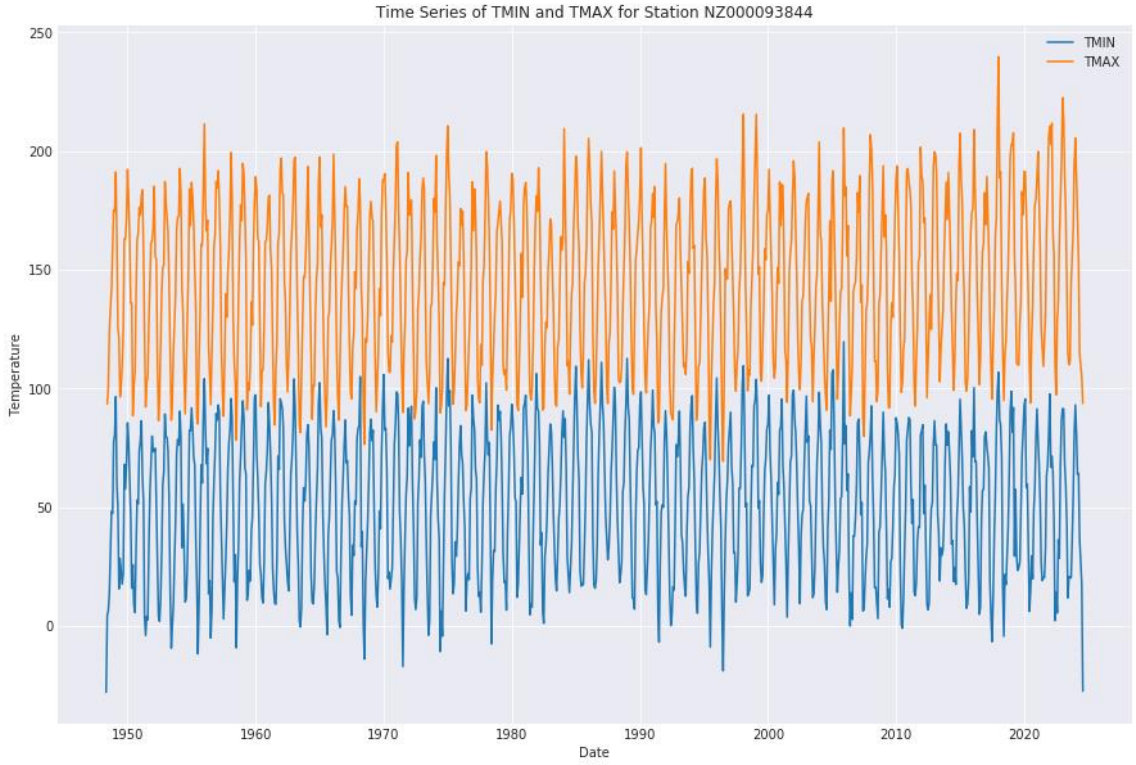

Fig 9.6.: Time series of TMIN and TMAX for Station NZ000093844

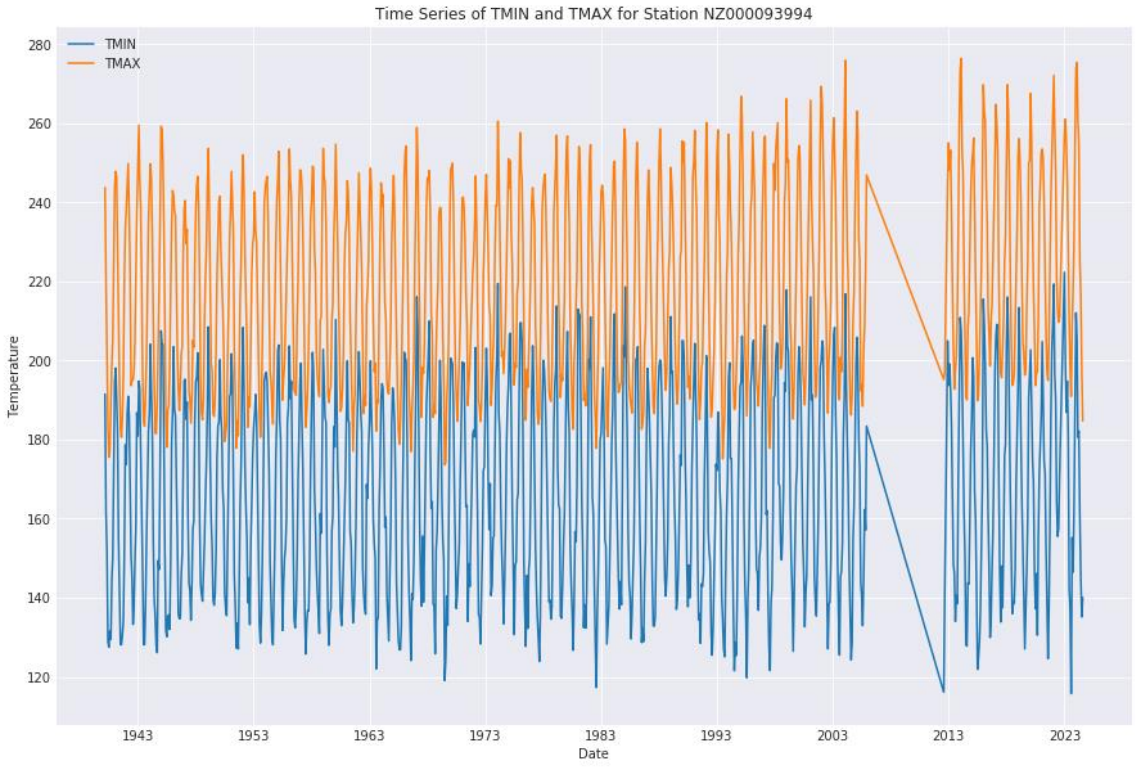

Fig 9.8.: Time series of TMIN and TMAX for Station NZ000093994

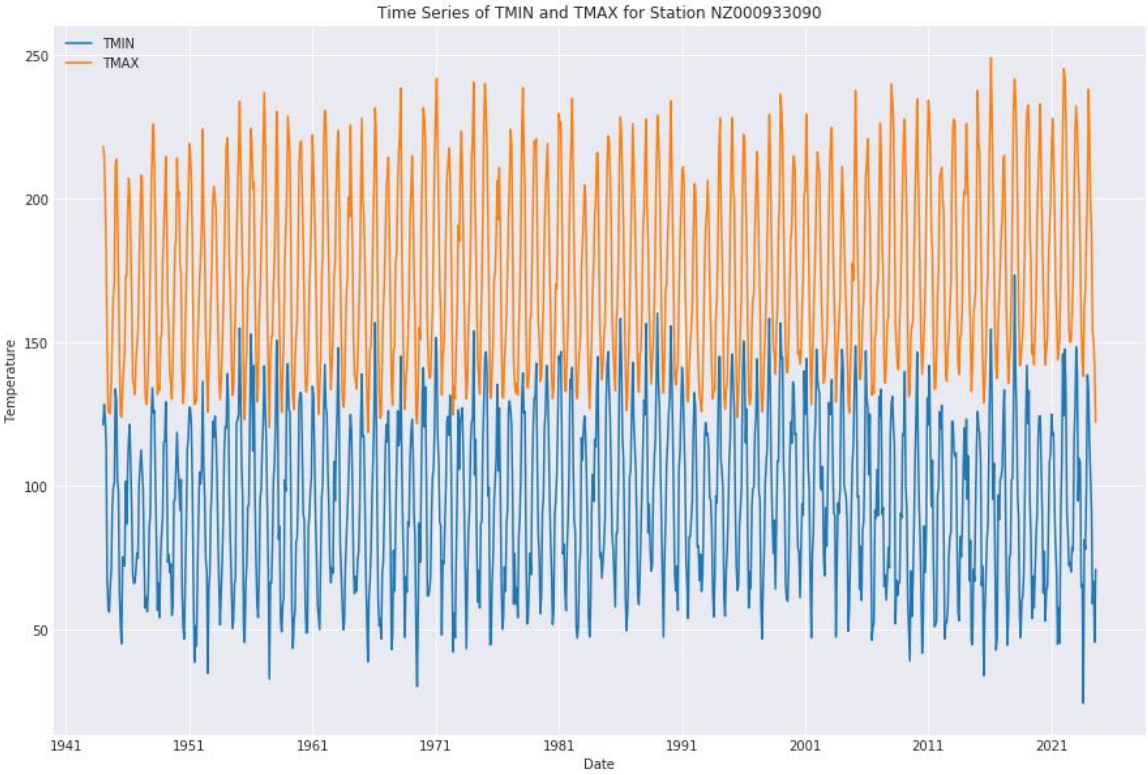

Fig 9.9.: Time series of TMIN and TMAX for Station NZ000933090

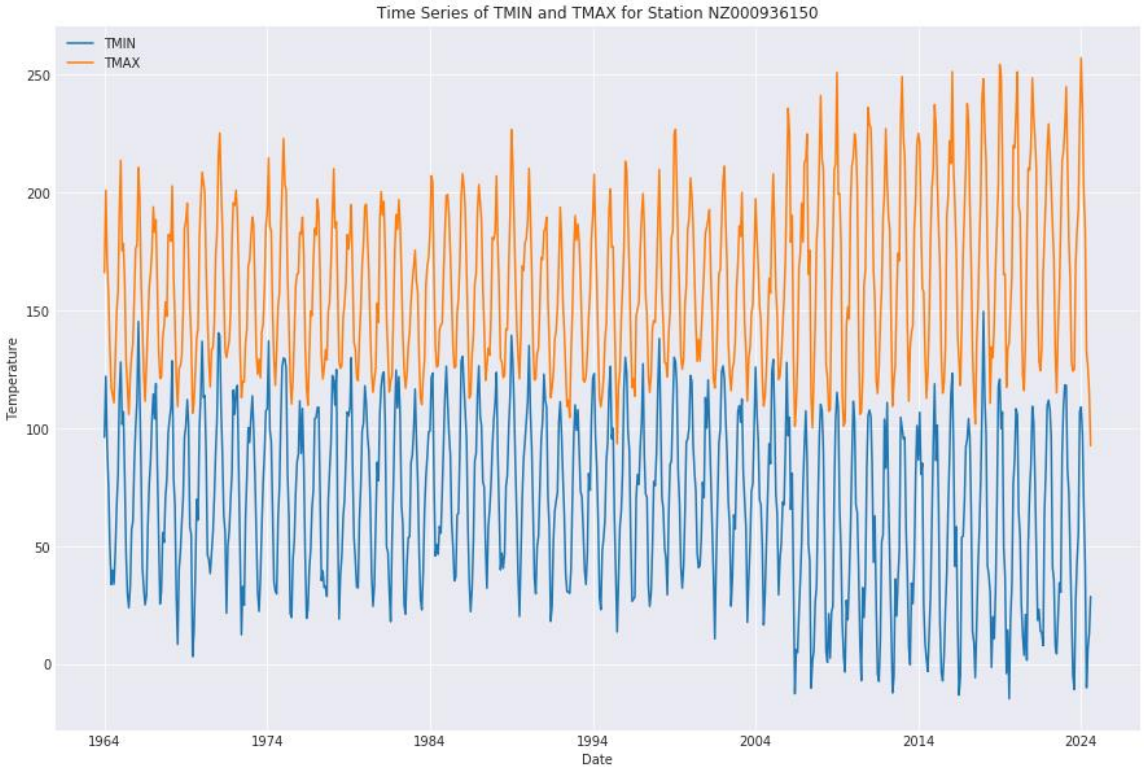

Fig 9.10.: Time series of TMIN and TMAX for Station NZ000936150

Fig 9.11.: Time series of TMIN and TMAX for Station NZ000937470

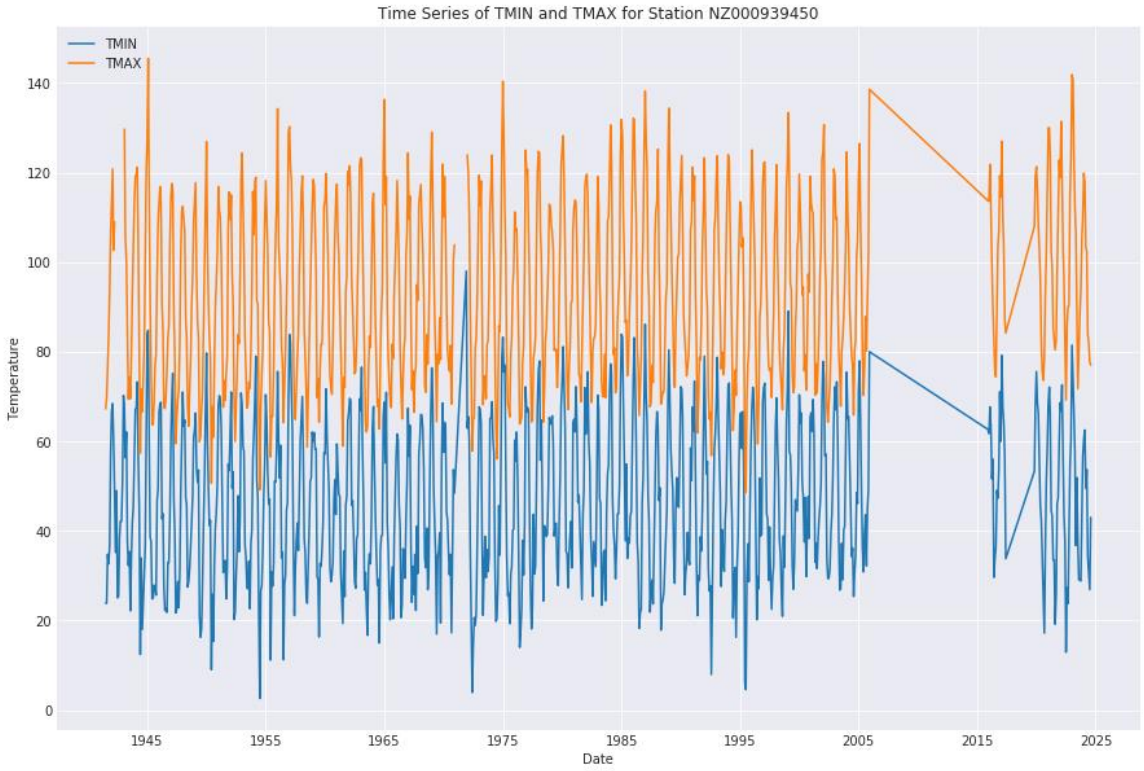

Fig 9.12.: Time series of TMIN and TMAX for Station NZ000939450

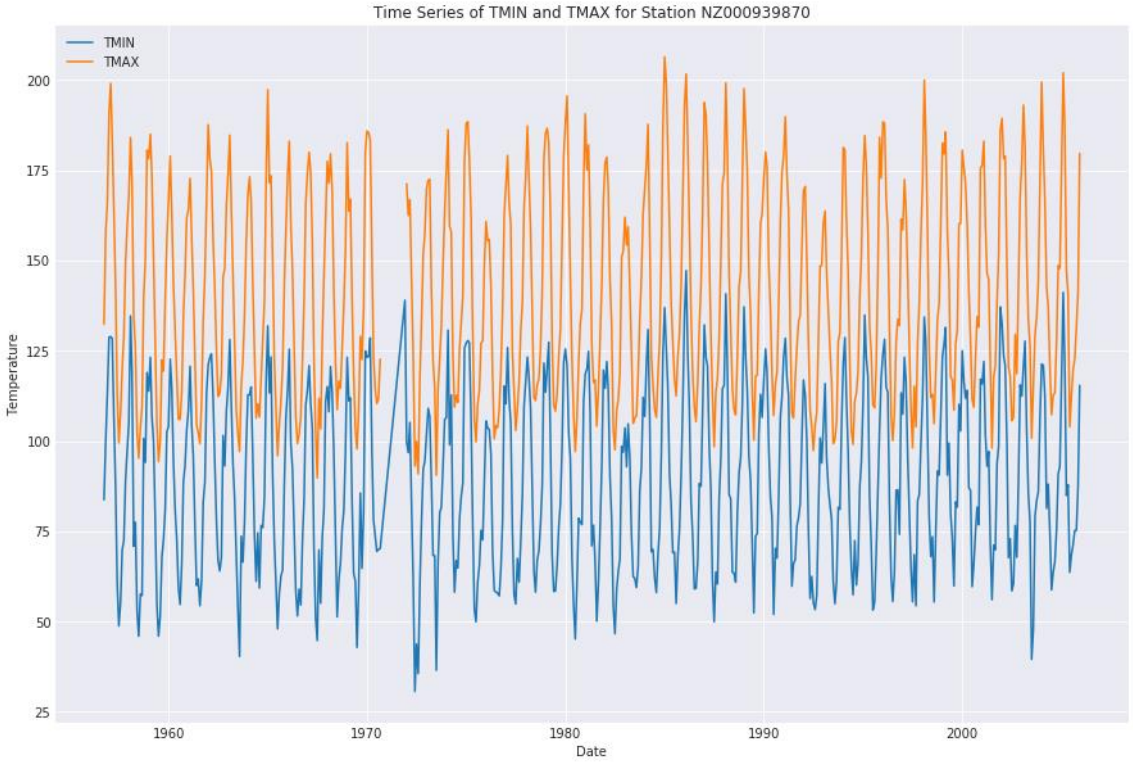

Fig 9.13.: Time series of TMIN and TMAX for Station NZ000939870

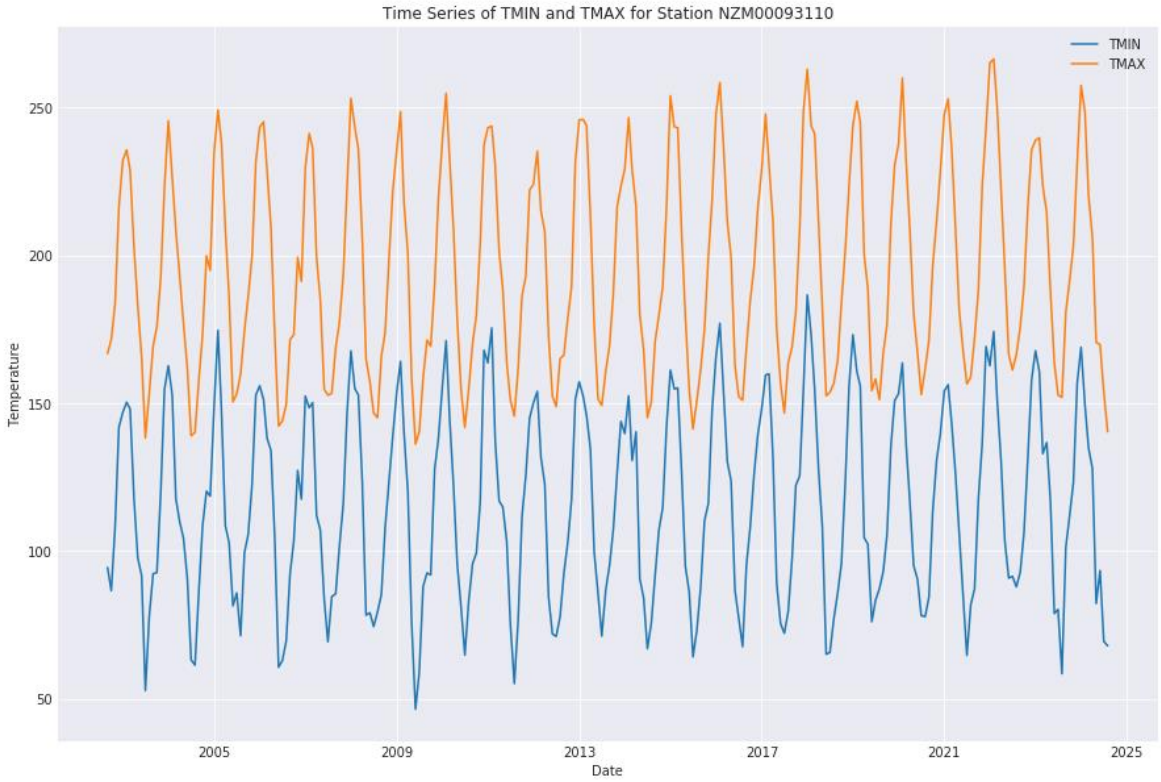

Fig 9.14.: Time series of TMIN and TMAX for Station NZM00093110

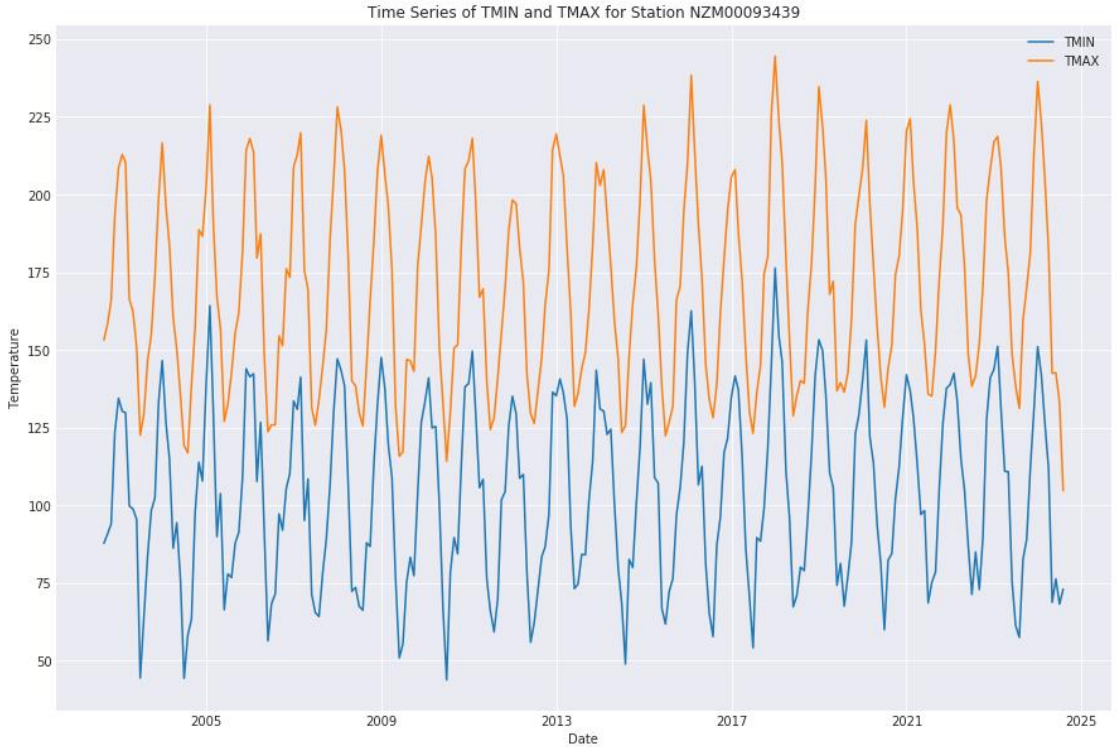

Fig 9.15.: Time series of TMIN and TMAX for Station NZM00093439

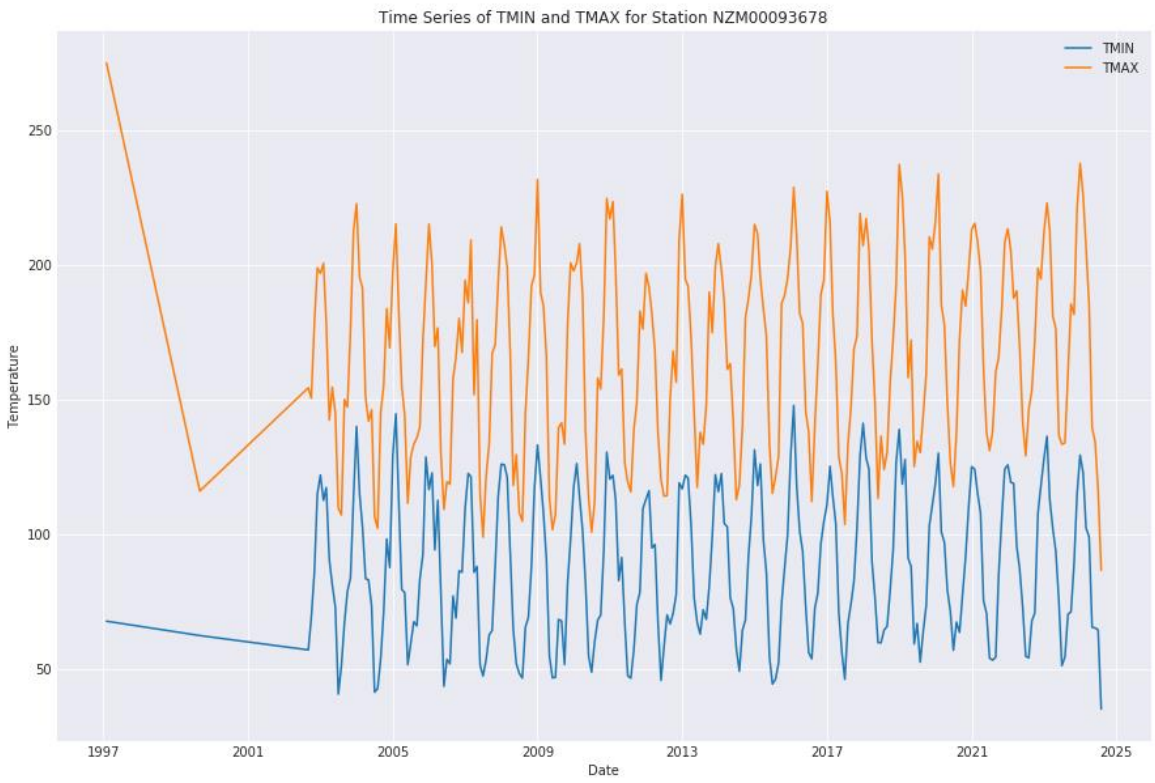

Fig 9.16.: Time series of TMIN and TMAX for Station NZM00093678

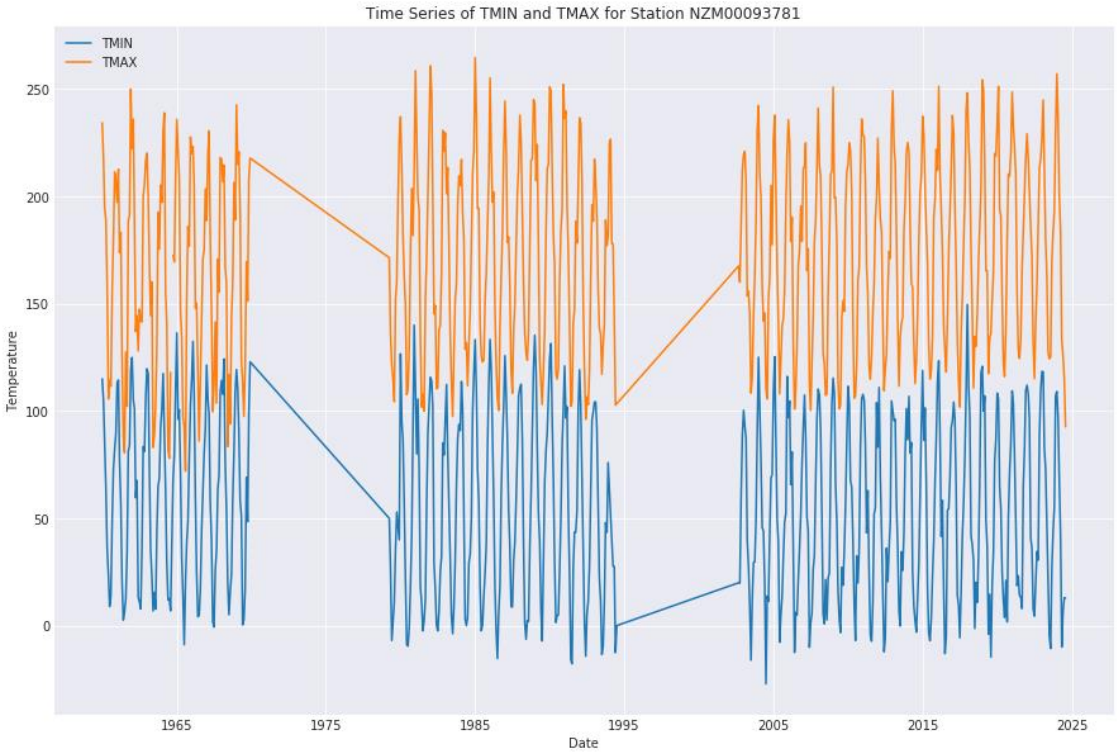

Fig 9.17.: Time series of TMIN and TMAX for Station NZM00093781

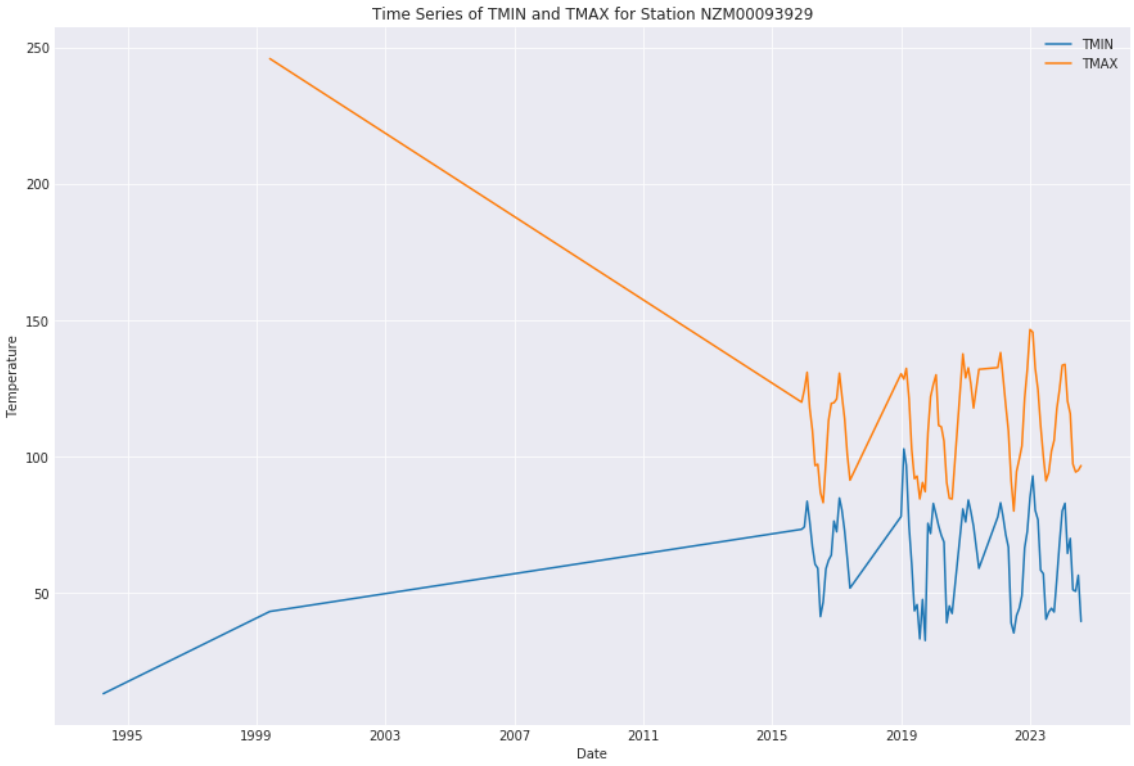

Fig 9.18.: Time series of TMIN and TMAX for Station NZM00093929