Comprehensive Analysis of Global Historical Climate Data Using Apache Spark

DATA420 Assignment 1

Name: Dan Wei

ID: 59174245

Table of content

1. Background

1.1 The Importance of Historical Climate Data Analysis

As global climate change increasingly affects ecosystems, economies, and societies, comprehensively understanding these changes and developing effective response strategies have become crucial. The rise in extreme weather events, sea levels, and glacier melting presents unprecedented challenges to global ecology and human activities (Intergovernmental Panel on Climate Change [IPCC], 2021). Against this backdrop, the analysis of historical climate data has become a key tool, not only aiding scientists in identifying and understanding long-term climate trends but also providing a scientific basis for policymakers to formulate response measures.

1.2 Introduction to the Dataset

The Global Historical Climatology Network (GHCN) dataset, integrating historical climate summaries from over 100,000 weather stations worldwide spanning 250 years, offers a valuable data resource for global climate change research. This dataset, by recording variables such as the highest and lowest temperatures, total precipitation, snowfall, and snow depth, supports the assessment of climate change impacts, including potential effects on ecosystems, human health, agricultural productivity, and economic activities (National Research Council, 2010).

1.3 Choosing Apache Spark as the Analysis Tool

When dealing with such large and complex datasets, selecting an efficient data processing tool is essential. Apache Spark, with its excellent data processing capabilities and efficient memory computing features, becomes the ideal choice for analyzing historical climate data. Spark not only has a significant advantage in data processing speed but also offers robust data processing and analysis capabilities through its Resilient Distributed Datasets (RDDs) and a rich set of APIs. Additionally, the successful application cases of Apache Spark across scientific research and the industrial sector demonstrate its effectiveness and reliability in handling large-scale data analysis tasks.

1.4 Purpose and Structure of the Report

The purpose of this report is to use Apache Spark to delve into the GHCN dataset and analyze the active periods of global weather stations, daily variables, and the differences in climate change across different regions. Through these analyses, we aim to provide profound insights into global climate change patterns, offering a scientific basis for formulating climate policies and response strategies.

The structure of the report is arranged as follows: It begins with an introduction to the background of global climate change and the importance of historical climate data analysis, followed by a detailed description of the GHCN dataset and the reasons for choosing Apache Spark as the analysis tool. Next, key findings and results are presented through data preprocessing and analysis. Finally, the report concludes by summarizing the entire text and discussing the potential impacts on understanding global climate change and the formulation of climate policies.

2. Processing

2.1 Dataset Structure

The structure of the data stored in hdfs:///data/ghcnd can be viewed using the HDFS command **hdfs dfs -ls**. The data is structured as follows:

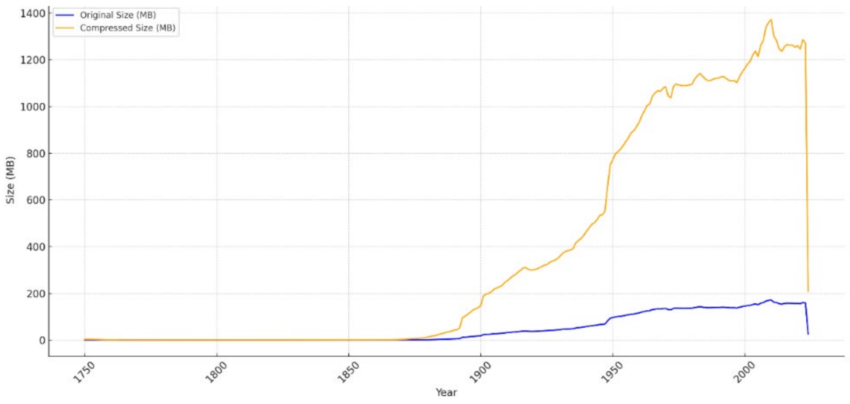
```
/data/ghcnd/
├── daily/
```

```
│    ├──1750.csv.gz
│    ├──1763.csv.gz
│    ├──……
│    ├──2023.csv.gz
│    ├──2024.csv.gz
├── ghcnd-countries.txt
├── ghcnd-inventory.txt
├── ghcnd-states.txt
└── ghcnd-stations.txt
```

The "daily" folder contains multiple ".csv.gz" compressed files named after years. Judging from the file names, these data cover a span from 1750 to 2024, involving a time span of 274 years. However, due to the absence of data for some years, there are actually files for 263 different years.

The size of the files can be viewed using the HDFS command **hdfs dfs -du**. The total size of all the data is 99.27 GB, which after compression is reduced to 12.41 GB. Specifically, the size of the "daily" folder is 98.94 GB, which after compression is reduced to 12.37 GB.

Figure 1: The yearly change in data size.



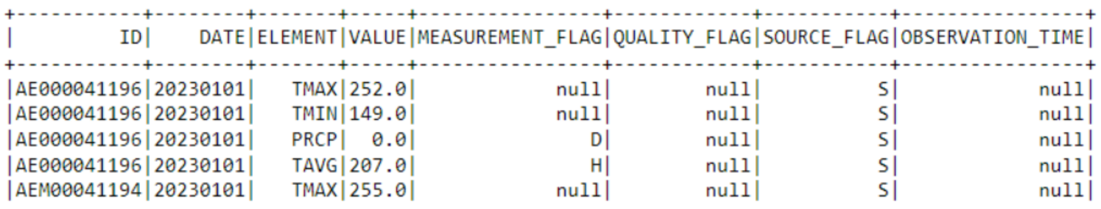
The size of the files varies significantly across different years. As shown in Figure 1, some files are as small as 3KB, while others are as large as 171MB. Over time, especially after entering the 20th century, the size of the files has shown a rapid growth trend. The annual volume of data collected has increased, which may be related to improvements in data collection methods, an increase in the number of monitoring points, or enhancements in the detail level of the data. The data volume for the year 2024 sharply decreases, which is due to the incompleteness of the data.

2.2 Schema Definition

Before reading the daily data with Spark, a schema is used to define the structure of the data in the DataFrame. A precisely defined schema ensures that the data has the correct format when read and improves performance when processing large volumes of data because Spark no longer needs to infer the data types. The schema is usually defined using a StructType object, which is a list containing multiple StructFields. Each StructField object defines a column, including its name, dataType, and whether the column can have null values. The dataType can be StringType, IntegerType, FloatType, DoubleType, DateType, etc. The choice of data types is based on the content of the fields.

In the daily dataset, except for the "VALUE" column which is of FloatType, all other columns are defined as StringType. The "VALUE" column represents measurement values, such as temperature, precipitation, or other meteorological data. These data are usually continuous values and can include decimals. Choosing FloatType allows us to perform calculations on the values, such as sum, average, maximum, and minimum. Other columns are considered text fields; they are used for identification and reference, not for mathematical calculations. Figure 2 shows the first ten rows of data in the daily folder for the year 2023.

Figure 2: Screenshot of the first five rows of daily data for 2023



```
+-----------+--------+-------+-----+----------------+------------+-----------+----------------+
|         ID|    DATE|ELEMENT|VALUE|MEASUREMENT_FLAG|QUALITY_FLAG|SOURCE_FLAG|OBSERVATION_TIME|
+-----------+--------+-------+-----+----------------+------------+-----------+----------------+
|AE000041196|20230101|   TMAX|252.0|            null|        null|          S|            null|
|AE000041196|20230101|   TMIN|149.0|            null|        null|          S|            null|
|AE000041196|20230101|   PRCP|  0.0|               D|        null|          S|            null|
|AE000041196|20230101|   TAVG|207.0|               H|        null|          S|            null|
|AEM00041194|20230101|   TMAX|255.0|            null|        null|          S|            null|
```

2.3 Data Preprocessing

2.3.1 Metadata Integration

The four datasets: stations, states, countries, and inventory are all in fixed-width text format. Since PySpark's read method does not directly support fixed-width formats, we need to manually specify the start position and length of each field based on character ranges. The specific method involves first using spark.read.text to load the data, followed by parsing with pyspark.sql.functions.substring. Afterwards, the DataFrame.count() method can be used to calculate the number of rows in each dataset, and the DataFrame.filter() method can be used to count the number of rows in stations that do not have a WMO ID. The results are as follows:

Number of rows in stations	125983
Number of rows in states	74
Number of rows in countries	219
Number of rows in inventory	747382
Number of stations without a WMO ID	118023

When calling DataFrame.count(), Spark triggers a job to traverse the entire DataFrame, typically meaning it reads the entire dataset to count the number of rows. This operation is an action, as opposed to a transformation, and it causes the computation to be actually executed. In Spark's lazy evaluation model, only when an action is called do all previous transformations get computed.

The DataFrame.filter() method is a transformation, meaning it does not execute immediately but is triggered when an action such as **.show(), .collect(), or .count()** is called.

During the data preprocessing stage, combine the daily climate summaries with the metadata tables, joining on station, state, and country to form an enriched table with information such as country code, country name, state name, FIRST_YEAR, LAST_YEAR, NUM_ELEMENT, and CORE_ELEMENT_COUNTS, primarily using methods like **.join(), .filter()**, and **.groupby()**.

First, extract the two-character country code from each station code in stations and store the output as a new column using the withColumn method. Then use the .join() method to left join stations and countries by country code, and similarly use .join() to connect stations and states by state code, noting that state codes are only provided for stations in the US; therefore, filter out stations with the country code "US" before joining.

Before connecting stations with inventory, analyze the types of climate data collected and the active periods at each climate station. The groupBy method groups data by StationID, and the agg method allows us to perform aggregate functions, using min and max to find the earliest and latest active years for each station. Similarly, use groupBy and agg to calculate the number of unique meteorological elements collected at each station. Use .filter() to select records that contain both core and non-core elements, and use .count() to calculate the number of types of core and other elements at each station, as well as the number of stations that only collect precipitation. The results are as follows:

Number of stations with all five core elements	20467
Number of stations that collect only precipitation	16301

Number of stations in the daily subset that are not in the enriched stations	7

Among these, the number of stations in the daily subset that are not in the enriched stations is found by left joining stations with inventory and then connecting with the first 1000 rows of the 2023 daily data through "ID", filtering out rows where the "NAME" column is NULL.

Before connecting with the 2023 daily dataset, the enriched stations table can first be saved to the output directory in parquet format. Parquet is a binary column-oriented storage format, making it especially efficient for executing analytical queries, particularly when not all columns of data need to be processed. Unlike CSV, which can only store flat table structures, Parquet supports nested data structures, enabling it to easily store complex data similar to JSON.

The cost of left joining (LEFT JOIN) the daily dataset and stations dataset depends on several factors, including the size of each dataset, partitioning, cluster configuration, and performance. If both datasets are very large, the left join can be very costly. During the JOIN operation, Spark needs to partition and transmit data over the network, which can consume significant computational and I/O resources, especially if the data does not have a good partitioning strategy.

One alternative to using a LEFT JOIN to determine whether all the sites in the daily dataset are included in the stations dataset is to use the subtract operation. Specifically, we can select the station ID column from the daily dataset and then subtract the station ID column from the stations dataset. Using this method, we avoid the need to perform a full LEFT JOIN operation, thereby potentially saving a significant amount of resources.

3. Analysis

In executing data analysis, resources up to 4 executors, 2 cores per executor, 4 GB of executor memory, and 4 GB of master memory are employed. In Spark, executors, cores per executor, executor memory, and master memory are key parameters in the configuration of cluster resources, which together determine the resource allocation and execution efficiency of Spark jobs.

Executors are processes that run on cluster nodes, responsible for executing tasks within Spark jobs and storing data for those jobs. Increasing the number of executors means more processes can execute tasks simultaneously, thus enhancing parallelism and the speed of job execution. Cores per executor refers to the number of CPU cores available to each executor. Executor memory is the amount of memory allocated to each executor, used for storing data of Spark jobs (such as RDDs) and for temporary data during task execution. Allocating more memory to each executor can handle larger datasets and reduce the risk of memory overflow. Master memory refers to the memory allocated to the Spark cluster's master node (also known as the driver), and increasing the master's memory helps manage larger jobs, especially when the job involves a large number of parallel tasks, requiring more memory to maintain task states and scheduling information.

3.1 Exploring the stations

First it will be helpful to know more about the stations themselves before we study the daily climate summaries in more detail.

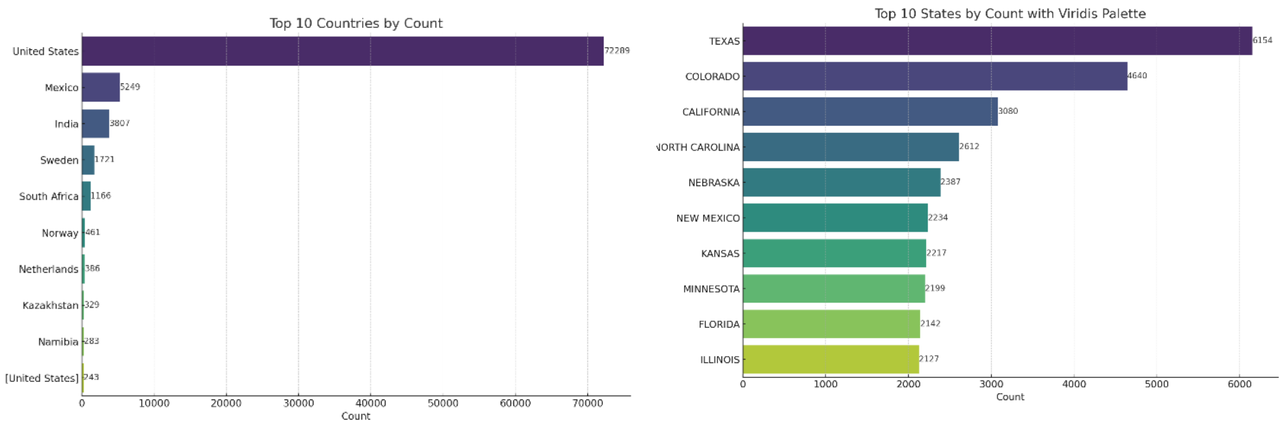
Before delving deeper into the daily climate summaries, it will be helpful to first understand more about the stations themselves. In the previously joined stations_enriched table, the number of unique station IDs can be identified and counted using **.select("ID")**.distinct().count(). The number of stations that were active so far in 2024 can be calculated with **.filter(stations_enriched['LAST_YEAR'] == 2024).select('ID').distinct().count().** To determine the number of stations located in the Southern Hemisphere, use **.filter(col('LATITUDE') < 0).count()**. The number of stations in former U.S. colonies, excluding those within the continental United States, can be identified by **.filter(col("COUNTRY_NAME").like("%United States%") &**

~(col("COUNTRY_CODE") == "US")).count(). The results are as follows:

Number of stations in total	125983
Number of stations that were active so far in 2024	31837
Number of stations in GSN	991
Number of stations in HCN	1218
Number of stations in CRN	234
Number of stations in multiple networks	15
Number of stations in the Southern Hemisphere	25316
Number of stations in the territories of the United States	386

In the same way, the number of stations in each country and each state in the United States can be calculated through **.groupBy('COUNTRY_CODE').count()** and **.groupBy('STATE_CODE').count()**. As shown on the left side of Figure 3 below, the United States has the most stations (72,289), followed by Mexico (5,249) and India (3,807). And TEXAS has the most stations in the United States, which is 6154.

Figure 3: Top 10 countries and states by number of sites



To calculate the two closest stations in New Zealand and the distance between them, we can first define a function haversine to calculate the great circle distance between two points. We need to convert the longitude and latitude into arcs first, and use the udf function to register it as A Spark UDF. We apply this UDF to a DataFrame containing all stations in New Zealand, compute the pairwise distances between all stations in New Zealand and find the two stations with the shortest distance via **.orderBy().first()**. The calculation results show that the two closest stations in New Zealand are: NZ000093417 and NZM00093439, the distance is: 50.529 kilometers.

3.2 Exploring all the daily climate summaries

In order to know how efficiently we can load and apply transformations to daily, we first need to understand the structure and format of daily.

We can use the command **hdfs getconf -confKey "dfs.blocksize"** to determine the default blocksize of HDFS. The output is 134217728 bytes (128 MB). In HDFS, HDFS splits the file into blocks, and Each block will be stored on a different node in the cluster. The size of each block is determined by the dfs.blocksize configuration.

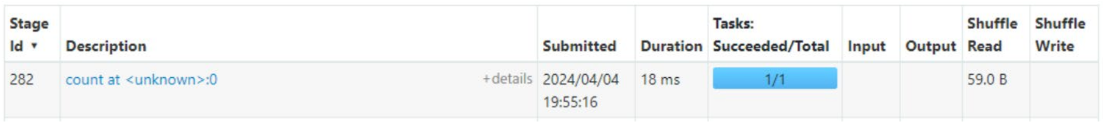
Since the size of the 2024 data file is 27,492,832 bytes (approximately 26.26MB), since the size of this file is smaller than the HDFS default one block size (128MB), the entire file can fit entirely into one HDFS block without needing to be split into Multiple blocks, even if the file size does not reach the size of a full block, a full block will be allocated to it. This means that the data files in 2024 only require one HDFS block.

The data file size for 2023 is 166,367,488 bytes (about 158.67MB), and since there can't be partial blocks, we have to round up to 2. This means that the data files for 2023 will be spread over 2 HDFS blocks. The size of the first block is 134,217,728 bytes (approximately 128 MB) and the size of the second block is 32,149,760 bytes (approximately 30.6 MB). Exploring all the daily data, We can get the following results:

Number of observations in 2023	37395851
Number of tasks were executed when count the number of observations in 2023	1
Number of observations in 2024	6061826
Number of tasks were executed when count the number of observations in 2023	1
Number of observations from 2014 to 2023	369419055
Number of tasks were executed when count the number of observations from 2014 to 2023	11
Number of observations in daily	3103953878

When we Load and count the number of observations in 2023 and then separately in 2024, the number of tasks were executed is one. By default, Spark will create a partition for each HDFS block, if each block of the file is executed separately If a partition is allocated, Spark will start a task for each partition when processing the data. 2023.csv.gz is a compressed file. Spark cannot determine how to split the file without decompressing it, and can only process the entire compressed file as a separate partition. Therefore, whether we check the number of tasks in the web user interface mathmadslinux2p:8080 (as shown in Figure 4) or use getNumPartitions to determine the number of partitions, we get the Number of tasks as 1.

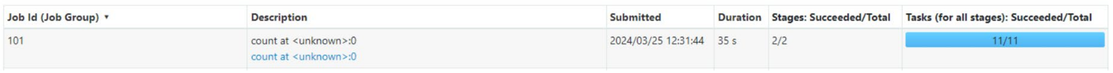
Figure 4: Screenshot 1 of Spark web user interface



Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
282	count at <unknown>:0 +details	2024/04/04 19:55:16	18 ms	1/1			59.0 B	

When we calculate the total number of observations from 2014 to 2023, the number of tasks were executed is 11. In Spark, since the gzip format does not support segmentation, when using the gzip compression format, Spark will separate the file creates a partition. This may be inefficient for large compressed files because it limits Spark's ability to parallelize data processing.

Figure 5: Screenshot 2 of Spark web user interface



Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
101	count at <unknown>:0 count at <unknown>:0	2024/03/25 12:31:44	35 s	2/2	11/11

As shown in Figure 5, There are 11 tasks would run in parallel when loading and applying transformations to all of daily. To increase parallelism, one can increase the number of executors or the number of cores per executor, or repartition the data, e.g. using The repartition() method increases the number of partitions. You can also change the way data is stored on HDFS, such as adjusting the HDFS block size or compression settings.

Figure 6: Number of the five core elements

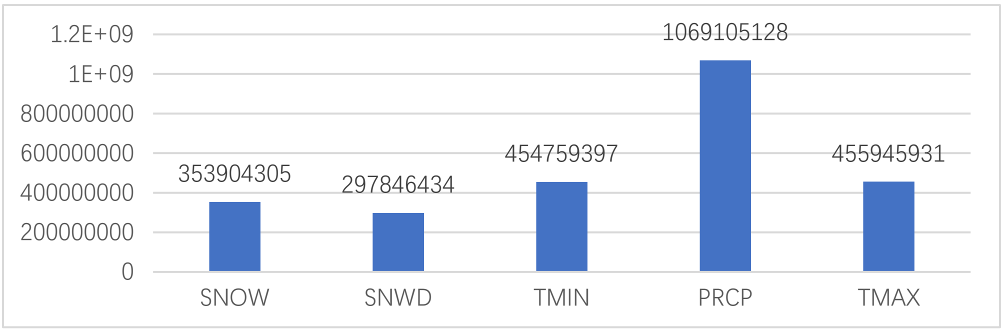


Figure 6 shows the results when exploring daily using filter command to obtain the subset of observations containing the five core elements. Obviously the PRCP has the most observations. The following are the results about temperature:

Number of observations of TMIN do not have a corresponding observation of TMAX	9322844
Number of stations of observations of TMIN do not have a corresponding observation of TMAX	27929
Number of observations of TMIN and TMAX in New Zealand	485520
Number of years of observations of TMIN and TMAX in New Zealand	85

To collect the changes in TMIN and TMAX of New Zealand's 15 stations over time, we can first cache the data set of New Zealand stations filtered by the cache to improve the computational efficiency. Figure 7 can clearly see the active time of each climate station in New Zealand. By comparing time series from different sites, differences in temperature changes across geographical locations can be observed. These differences may be related to elevation, geographic location, or other regional climate characteristics. The overall temperature is on an upward trend, which is consistent with global warming.

Figure 7: Plot time series for TMIN and TMAX for each station in New Zealand

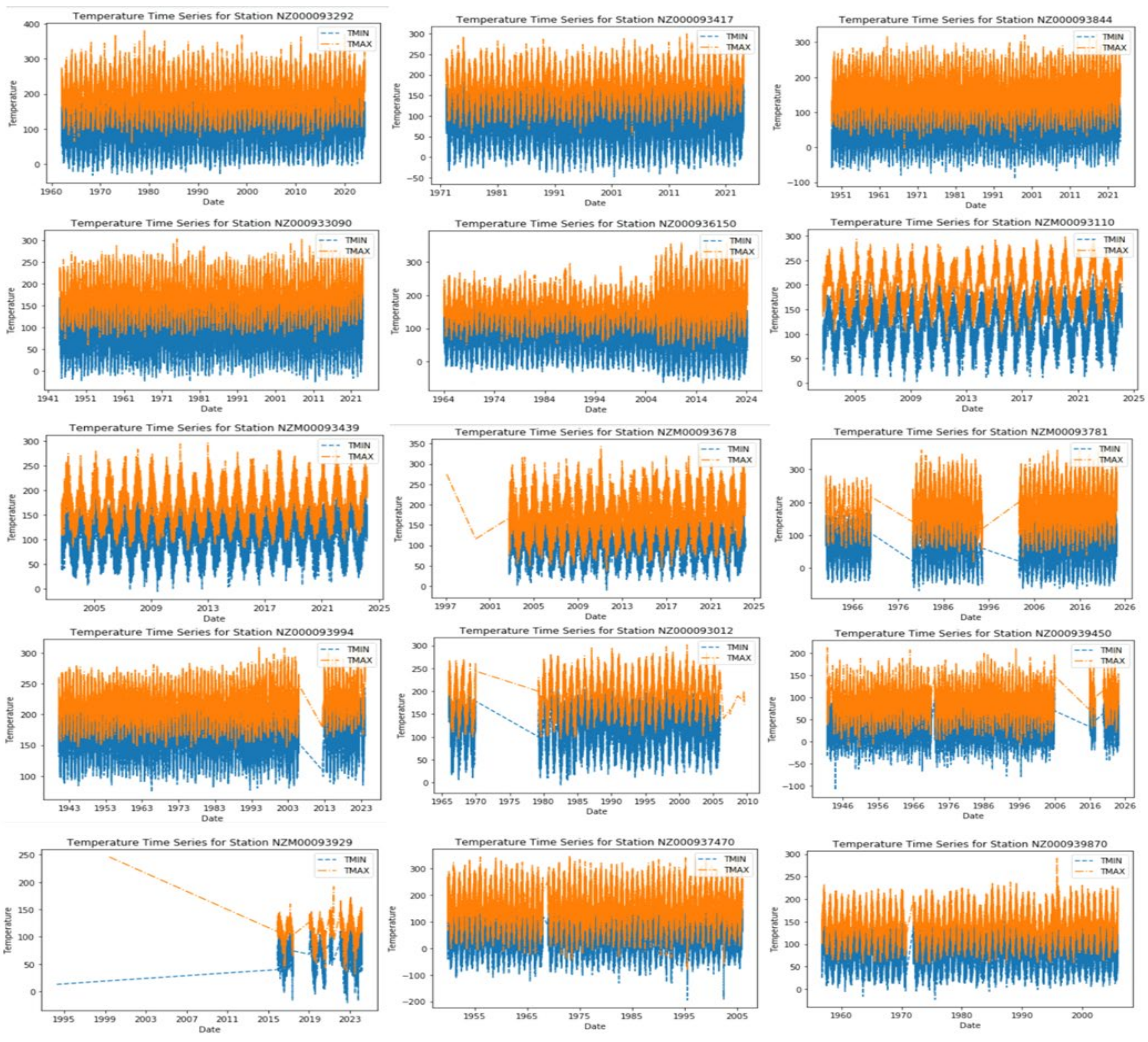


Figure 8: plot the average time series for TMIN and TMAX together for New Zealand

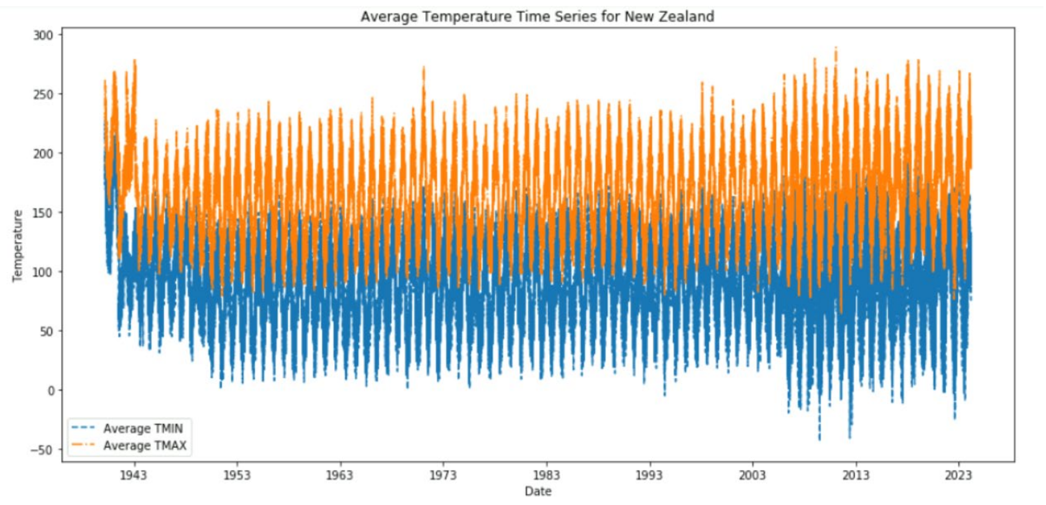


Figure 8 is a time series of average temperatures in New Zealand, showing temperature changes from approximately 1943 to recent times. It can be seen that TMAX gradually increases with time. In addition, the difference between the highest and lowest average temperatures, that is, the daily temperature difference, shows a gradually increasing trend.

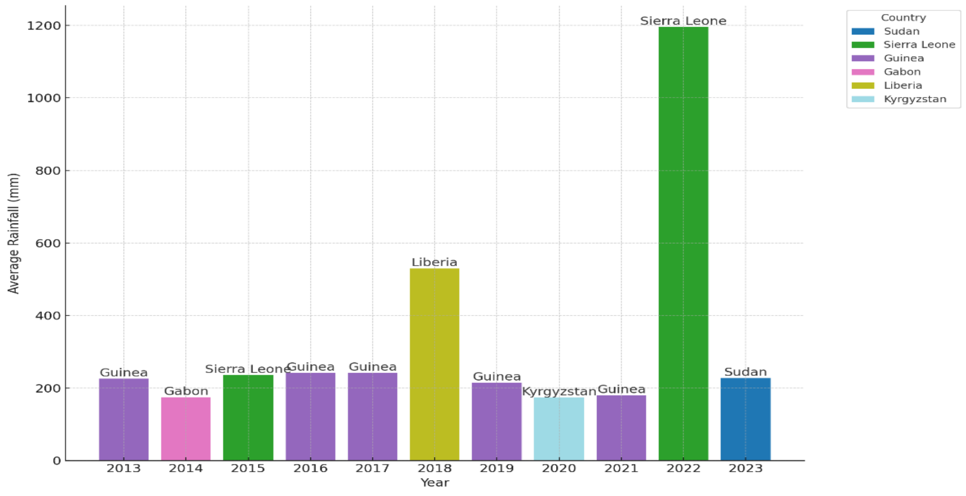
3.3 Exploring precipitation

Use PySpark's window function to find the country with the highest average rainfall in each year. For example:

windowSpec=Window.partitionBy("YEAR").orderBy(F.desc("AVERAGE_RAINFALL")).

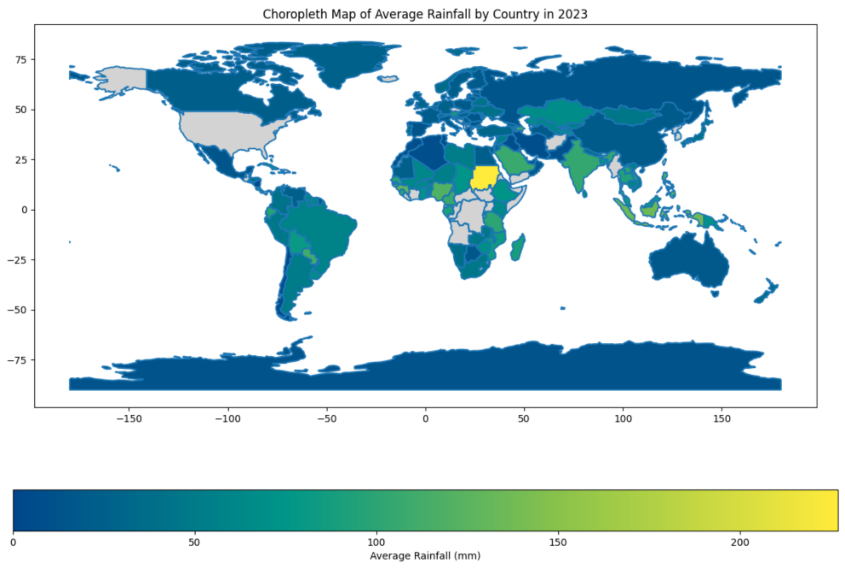
This line of code creates a window specification windowSpec. Window functions provide an efficient way to perform complex data transformation operations such as ranking and intra-partition sorting, and are suitable for handling such group-by-group aggregation and comparison problems. We can get Figure 9, which shows the countries with the highest average rainfall in the past 10 years. Among them, Guinea has been the country with the highest average rainfall in 5 of the past 10 years.

Figure 9: Countries with the highest average rainfall from 2013 to 2023



In order to plot the average rainfall of each country in 2023, we use the geopandas library. Due to permission issues, we cannot install geopandas directly in pyspark. Because the data is saved locally and the local Python is used to draw the graph. The specific method is to first use **mkdir -p /users/home/dwe55/rainfall** to create a local folder, and then use command: **hadoop fs -**

copyToLocal /user/dwe55/outputs/df_avg_rainfall /users/home/dwe55/rainfall to save the file to local. Finally we visualize the data as Figure 9, where the gray part represents missing values.
Figure9: The average rainfall in 2023 for each country.



4. Conclusions

This report provides a comprehensive analysis of the GHCN (Global Historical Climate Network) dataset using Apache Spark. Our focus has been on the active periods of weather stations, daily variables, as well as trends in average temperature changes and climate variation across different regions. Through these analyses, we have been able to observe long-term climate trends and the climatic characteristics of different geographical locations.

4.1 Insights and Impacts

The analysis has revealed several important insights, including the historical depth of weather station data, the spatial distribution of rainfall and temperature observations, and the calculation of distances between weather stations in New Zealand. These insights are of significant value in understanding global climate change and provide foundational data for the formulation of climate policies and the construction of climate models. In particular, by analyzing changes in average rainfall across countries, we can better understand potential changes in water resource distribution, which is crucial for formulating strategies to combat climate change.

4.2 Challenges and Limitations

During the analysis, we faced challenges with the large volume of data and diversity of data formats. Although Apache Spark offers robust data processing capabilities, the efficiency of reading and transforming large datasets in compressed formats remains a consideration. Additionally, the integrity and accuracy of data, given its broad sources, were also key concerns in the analysis.

4.3 Future Work

Based on the current findings, future research could delve deeper in multiple directions. Expanding the dataset range: Include more years of data and data collected from more weather stations to provide a more comprehensive perspective on climate change. Adopting new analytical methods: Explore more advanced data analysis techniques, such as machine learning and artificial intelligence, to extract deeper features of climate change.

Through these future work directions, we can further deepen our understanding of global climate change and support the formulation of more effective response measures.

Acknowledgements
In the completion of this report, I have been greatly aided and inspired by many individuals. First, I would like to especially thank my lecturer James, my tutors Hazel and Yiwen, and my classmates. They not only provided valuable knowledge and insights but also offered clear explanations and a solid theoretical foundation for my report through their in-depth and clear discussions of complex concepts.
Secondly, I would like to express my deep gratitude to OpenAI for providing ChatGPT. Throughout various stages of this project, ChatGPT has been a powerful tool in guiding and advising on report writing and data handling ideas.

Reference

Intergovernmental Panel on Climate Change. (2021). Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change.
National Research Council. (2010). Advancing the Science of Climate Change. Washington, DC: The National Academies Press.