DATA420-22S1 (D)

Assignment 1

GHCN Data Analysis using Spark

Bhanu Paturi (51210188)

May 2, 2022

Table of Contents

Introduction

The weather data collected by the *Global Historical Climatology Network (GHCN)* is explored, analysed and presented in this report.

The *GHCN is an integrated database of* historic weather covering 260 years of observations from over 100,000 distinct weather stations in 219 countries or territories around the world. This is a collective task where more than 20 independent sources are involved. This data is stored mainly in two different formats; they are compressed gzip files and text files. Along with these, a pdf documentation is also been released that provides an information related to the observations in the data.

In the current report, I have focused on:

- Processing
    - Exploring the data
    - Structuring
    - Performing transformations
- Analysing
    - Summaries
    - User Defined Function (UDF)
    - Understand the concept of parallelism and distribution in Spark
    - Finding insights

The pyspark is an interface for Apache Spark in Python, which is used for large-scale data processing. Some of the code snippets, outputs and plots were used along the way in answering to the questions in this report. Apart from this, the code was split into three files for processing, analysis and HDFS commands. Python is used for plotting all the graphs, which are stored in a zip file and submitted along with the report.

Processing

Q1 Exploring the data

There is one directory and five files in data/ghcnd/. The daily directory is a collection of daily climate summaries for 260 years, i.e. from 1763 to 2022 of data in 260 files. These files are of csv format and are compressed as gzip files. The metadata ghcnd-countries.txt, ghcnd-inventory.txt, ghcnd-stations.txt, ghcnd- states.txt and readme.txt are all text files.

The tree diagram shows how the data is structured in ghcnd.

```
/data/ghcnd/
├─ countries
├─ daily
│  ├─ 1763.csv.gz
│  ├─ 1764.csv.gz
│  ├─ 1765.csv.gz
│  ├─ ...
│  └─ 2022.csv.gz
├─ inventory
├─ states
└─ stations
```

The file sizes in the daily folder has an incremental growth; in the year 1763, the file size was in 3.358 KB or 3358 bytes. Until the year 1814 for about 60 years, there was a small change in the file size that was increased by just 8.7 KB. In 1877, it reached 1.114978 MB. By the year 1948, within a span of 70 years it was 113.945 MB in size, an increase from 1 MB to 100 MB. Until the year 2017, there was an increase in file size and reached up to 209.584 MB. As the years pass, more stations were included, which might be the reason for the increase in file size.

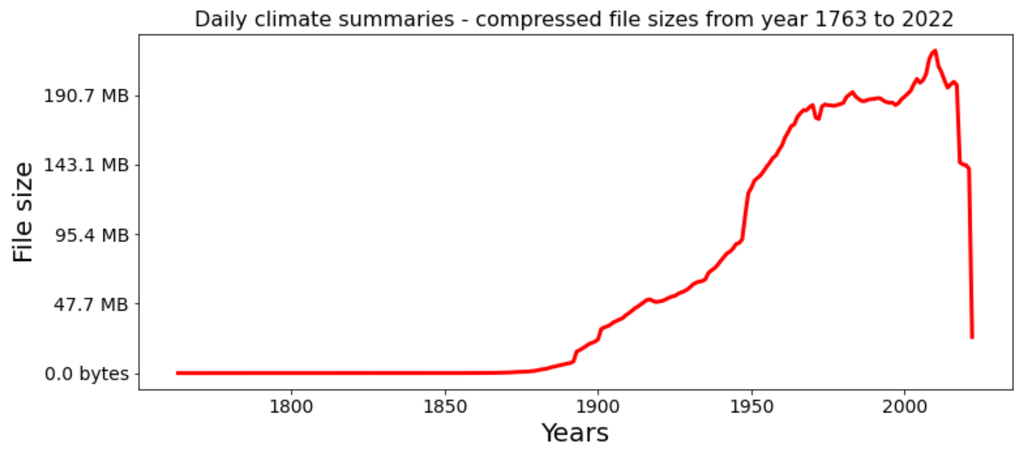


Figure. Compressed file sizes of daily climate summaries from the year 1763 to 2022.

From the above figure, we can see an increase in file size over the years. In year 2017, the file size was 207.342 MB, whereas in year 2018 it was 151.565 MB, which has been decreased by 55 MB. This sudden decrease in the file size might be related to data collection issues. Since then, the file size has been decreasing by 1 MB until year 2021. In year 2022, the file size is just 25.986 MB as it has only 3 months of data in it since we have just finished the first quartile of the year. The sizes mentioned above are for compressed files, and the actual files could be slightly higher than what we got. The overall size of compressed data for daily is 15.7 GB, but the actual size could be around 157 GB, because the gzip compression ratio is 10:1 for text based csv files.

```
$ hdfs dfs -du -h /data/ghcnd
```

File Name	**File Size**
daily	15.7 G
countries	3.6 K
inventory	30.9 M
stations	1.1 K
states	9.7 M

The total size of all the data is 15.7406307 GB. 15.7 GB of data in daily and the combined metadata has just 40.6307 MB. Approximately 99.97% of the total data is of daily.

Q2 Structuring

Throughout the processing steps, script provided here uses the Spark session with 2 executors, 1 core per executor, 1 GB of executor memory and 1 GB of master memory. The start_pysprak_shell command is used to start working in pyspark shell. Then import all the required libraries to work in.

The schema is defined for daily based on the documentation provided to parse data into columns with specified types. The daily table has DATE column, which represents DateType and OBSERVATION TIME column represents TimestampType but they were not formatted in a way that can be parsed automatically. Hence, StringType is used as a datatype for both the columns.

In order to explore the data in daily, I have loaded the 2022.csv.gz file using spark.read.load() limiting the rows to 1000. The top 20 rows, which were displayed compared to the documentation provided has accurate data in them. However, the OBSERVATION TIME had all null values and some of the id's in the ID column had repeated values for the same DATE. This might be because the observations were recorded several times in a day. If the OBSERVATION TIME for these particular rows had a value, then it would have confirmed our assumption. Data stored in metadata are all text files. To parse these files, a fixed text formatting is needed as we pass them into a separate load statement by using pyspark.sql.functions like trim(), substring() and alias() to change their column names according to documentation provided.

```
stations_text = (
   spark.read.format('text')
   .load("/data/ghcnd/ghcnd-stations.txt")
  )
stations = stations_text.select(
   F.trim(F.substring(F.col('value'),1,11)).alias('ID').cast(schema_Stations['ID'].dataType),
   F.trim(F.substring(F.col('value'),13,8)).alias('LATITUDE').cast(schema_Stations['LATITUDE'].dataType),
   …
   ...
```

Metadata	Rows
countries	219
inventory	704,963
stations	118,493
states	74

The countries has 219 observations, which represent 219 countries or territories. The inventory has 704,963 observations. Whereas, stations have 118,493 observations representing individual stations around the world. The states have 74 observations representing different states around the world. The data analysed in this report has stations with and without World Meteorological Organisation number (WMO ID). A total of 110,407 stations do not have WMO ID's, which is considerably a large number in the dataset.

Q3 Performing transformations

`hdfs dfs –mkdir /user/bpa78/outputs/ghcnd/` is used to create an output directory in our home directory to save the outputs based on the tasks that we perform during the processing and analysis.

To extract the two character code from the stations code in stations, the substr() is used based on the column 'ID' and named the new column as 'CODE' using withColumn(). As both the Stations and Countries metadata tables has the same column name 'NAME', the observations in these columns are not same as one has Station names and the other has Country names. So in order to avoid ambiguity, the 'NAME' in the Countries dataframe is changed to 'COUNTRYNAME' using alias(). Then a new table stations_joined is created by joining Stations and Countries based on 'CODE' using left join.

To get the stations in US states, the stations_joined and States metadata are joined by 'CODE' using left join after changing the column name 'NAME' in states to 'STATENAME' and then filtering stations in US based on 'CODE' == 'US'.

Using an inventory metadata, the observations for first year, last year and the count of elements for each station is obtained by selecting 'ID', 'ELEMENT', 'FIRSTYEAR' and 'LAST YEAR' columns from the inventory. Then grouping it by ID, aggregating the FIRSTYEAR with min, LASTYEAR to max and ELEMENT to count as a key value pairs (in the form of dictionaries) and alias() are used to change column names appropriately by creating a subset of inventory. Then to get the number of core elements, a separate dataframe of inventory is created where the 'ELEMENT' column is filtered using isin() specifying all the five core elements in it, grouped by 'ID' and aggregated ELEMENT by count. The new column is renamed as 'NUM_CORE_ELEMENT'. To get the number of other elements, the inventory table is filtered using negation '~' along with isin() and checked for the elements that are not in five core elements. It is then grouped by ID and aggregated by ELEMENT as done earlier and now the new column is 'NUM_OTHER_ELEMENT'. For the stations that collect only precipitation, the ELEMENT column is filtered for PRCP, as done earlier in two cases that are grouped, aggregated, and then stored that value in a new column called PRCP. Further, all the three newly created dataframes are joined using left join based on ID.

```
……
.filter(~F.col('ELEMENT').isin('PRCP', 'SNOW', 'SNWD', 'TMAX', 'TMIN'))
……
```

20,289 stations collected all five-core elements and 16,136 stations collected only precipitation; these values were obtained by filtering.

By joining the enriched stations table (stations_elements) with enriched inventory table (inventory_elements) using left join based on ID, a new table was created. This table has 18 columns comprising of both old and new columns. This dataframe is stored as .csv file and in .csv.gz formats. Two different directories each having 4 files were created. Since we had 4 partitions, we get 4 files. The file size of one csv file is 2.8 MB and the size of one csv.gz file is 728.4 KB. We can clearly see that 4 files in compressed format is equal to the size of 1 csv file. When tried to save as parquet file, errors were occurred because of GSN FLAG, HCN/CRN FLAG and WMO ID columns as they are separated by blank space. In order to save it in parquet, column names have to be changed.

Considering the pros and cons of data storage like accessibility, size in the disk, read and write cost, the csv is the best format to be used for storing the data. The size of the data is smaller, which can be easily stored in our disk, as we have enough disk space. It is the most convenient format to be used in R and Python for plotting. This retains the headers and data types of the columns. Read and write cost is much lower as the size of the file is small. It is easy to split or subset the columns for further use. Based on all these considerations, csv format is used to save the file. If opted to save the file as csv.gz and use it for further analysis, it could be a tedious job as the headers and the data types of the columns are not retained. If we had a large file size, then it could be the better option to save our file as csv.gz. The data stored is not splittable; if we need a subset of the data then the whole file should be read which is more expensive. The read and write cost is more expensive when we store the file on parquet format. Storing the data in binary format is a good option when we have a large file size. Therefore, for the current analysis, data is stored as a .csv file.

```
stations = spark.read.load('hdfs:///user/bpa78/outputs/ghcnd/Stations', format = "com.databricks.spark.csv", header = True)
stations.show()
# +-----------+----+--------+---------+---------+-----+--------------------+--------+------------+------+--------------------+---------+--------+--------+-----------+--------------
--+-----------------+----+
# |         ID|CODE|LATITUDE|LONGITUDE|ELEVATION|STATE|                NAME|GSN FLAG|HCN/CRN FLAG|WMO ID|
COUNTRYNAME|STATENAME|MIN_YEAR|MAX_YEAR|NUM_ELEMENT|NUM_CORE_ELEMENT|NUM_OTHER_ELEMENT|PRCP|
# +-----------+----+--------+---------+---------+-----+--------------------+--------+------------+------+--------------------+---------+--------+--------+-----------+--------------
--+-----------------+----+
# |AE000041196|  AE|  25.333|   55.517|     34.0| null| SHARJAH INTER. AIRP|     GSN|        null| 41196|United Arab Emirates|     null|    1944|
2021|          4|               3|                1|   1|
stations = spark.read.load('hdfs:///user/bpa78/outputs/ghcnd/Station.csv.gz', format = "com.databricks.spark.csv", header = True)
stations.show()
# +-----------+---+------+------+------+----+--------------------+----+----+-----+--------------------+----+----+----+----+----+----+----+
# |        _c0|_c1|   _c2|   _c3|   _c4| _c5|                 _c6| _c7| _c8|  _c9|                _c10|_c11|_c12|_c13|_c14|_c15|_c16|_c17|
# +-----------+---+------+------+------+----+--------------------+----+----+-----+--------------------+----+----+----+----+----+----+----+
# |AE000041196| AE|25.333|55.517|  34.0|null| SHARJAH INTER. AIRP| GSN|null|41196|United Arab Emirates|null|1944|2021|   4|   3|   1|
1|
[bpa78@canterbury.ac.nz@mathmadslinux2p ~]$ hdfs dfs -du -h /user/bpa78/outputs/ghcnd/Station.csv.gz
# 0       0     /user/bpa78/outputs/ghcnd/Station.csv.gz/_SUCCESS
# 728.4 K  2.8 M  /user/bpa78/outputs/ghcnd/Station.csv.gz/part-00000-443d5e2e-c5bc-4156-b64c-cdd8238e9a76-c000.csv.gz
# 736.4 K  2.9 M  /user/bpa78/outputs/ghcnd/Station.csv.gz/part-00001-443d5e2e-c5bc-4156-b64c-cdd8238e9a76-c000.csv.gz
# 733.8 K  2.9 M  /user/bpa78/outputs/ghcnd/Station.csv.gz/part-00002-443d5e2e-c5bc-4156-b64c-cdd8238e9a76-c000.csv.gz
# 744.1 K  2.9 M  /user/bpa78/outputs/ghcnd/Station.csv.gz/part-00003-443d5e2e-c5bc-4156-b64c-cdd8238e9a76-c000.csv.gz
[bpa78@canterbury.ac.nz@mathmadslinux2p ~]$ hdfs dfs -du -h /user/bpa78/outputs/ghcnd/Stations
# 0     0      /user/bpa78/outputs/ghcnd/Stations/_SUCCESS
# 2.8 M  11.2 M  /user/bpa78/outputs/ghcnd/Stations/part-00000-f2b9377c-56eb-48e4-9c6a-0253b32fae21-c000.csv
# 2.8 M  11.3 M  /user/bpa78/outputs/ghcnd/Stations/part-00001-f2b9377c-56eb-48e4-9c6a-0253b32fae21-c000.csv
# 2.8 M  11.2 M  /user/bpa78/outputs/ghcnd/Stations/part-00002-f2b9377c-56eb-48e4-9c6a-0253b32fae21-c000.csv
# 2.8 M  11.4 M  /user/bpa78/outputs/ghcnd/Stations/part-00003-f2b9377c-56eb-48e4-9c6a-0253b32fae21-c000.csv
```

Furthermore, stations are joined with 1000 rows of year 2022 daily data using left join and got zero stations, which were in daily but not in stations table.

Left join all of the daily and stations will be expensive when considering the size of daily data. By using subtract() in place of left join, we can get the information about which stations are in daily but not in stations.

Analysis

For the analysis part, the resource allocation was increased to 4 executors, 2 cores per executor, 4GB of executor memory and 4GB of master memory. This is done manually while starting the shell.

```
start_pyspark_shell -e4 -c2 -w4 -m4
```

Q1 Summaries

There are 118,493 stations in total and only 38,284 stations were active for 2021: 991 stations in GCOS Surface Network (GSN), 1218 stations in US Historical Climatology Network (HCN) and 0 stations in US Climate Reference Network (CRN). In order to get these values for different networks, the corresponding columns were filtered and count() is used.

There were 14 stations, which has more than one of these networks. The stations were obtained by checking not null constraint for the GSN FLAN and HCN/CRN FLAG columns.

To get the number of stations in each country, data were grouped by the 'COUNTRYNAME', aggregating 'ID' by count and then ColumnRenamed() is used for changing the column name as 'NUM_STATIONS'. This output is then stored as countries.csv file using write.csv() in outputs/ ghcnd/ directory. The same steps were performed accordingly to get the total number of stations in each state and the output is stored as states.csv in the output directory.

About 93,156 stations are in the Northern Hemisphere, which is obtained by filtering the LATITUDE column for all the values that are greater than 0.

There are about 339 stations in total in the territories of US after excluding stations in US. To get this, I have filtered all the values from 'COUNTRYNAME' column that contains United States and from 'CODE' column where code is not equal to US, and finally using count() to get the count of stations in US territories.

Q2 User Defined Function (UDF)

Haversine formula calculates the geographic distance between two different points on the earth. The central angle haversine can be computed, between two points with ***r*** as radius of earth, ***d*** as the distance between two points, ϕ_1, ϕ_2 is **latitude** of two points and λ_1, λ_2 is **longitude** of two points respectively, as:

$$\operatorname{haversin}\left(\frac{d}{r}\right) = \operatorname{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\operatorname{haversin}(\lambda_2 - \lambda_1)$$

This formula uses latitudes and longitudes of the two stations as arguments to calculate the geographical distance between them, which is used to write a Spark user defined function. The lat1, lon1, lat2 and lon2 are the latitudes and longitudes of the two stations, respectively. These are converted into radians and the difference between these latitudes and longitudes are computed and stored in two different variables; the area is calculated using these values. The central angle is computed. The distance is the central angle times the radius of the earth in kms i.e. R = 6373.0 kms, which is a constant value. This function is further called while computing the pairwise distance between New Zealand stations. First, the stations in New Zealand were filtered and two subsets of this dataframe was created. Then cross joining these two subsets and filter on ID1 < ID2 columns in order to remove duplicates and to get the results for unique pairs of different stations. Then to calculate the distance between these unique pairs, a new column DISTANCE is created and for its values, the User defined function that was created earlier is called by passing the parameters to it. The cast() is used to cast DISTANCE as double data type. The output is stored as a distance_NZstations.csv file in outputs directory. The above-obtained results are then ordered in ascending order from which we get WELLINGTON AERO AWS and PARAPARAUMU AWS stations that are geographically closest in New Zealand.

```
#code
distance_NZstations.orderBy(F.col('DISTANCE').asc()).show()
# Output
# +-----------+-------------------+---------+----------+-----------+-------------------+---------+----------+--------+
# |        ID1|              NAME1|LATITUDE1|LONGITUDE1|        ID2|              NAME2|LATITUDE2|LONGITUDE2|DISTANCE|
# +-----------+-------------------+---------+----------+-----------+-------------------+---------+----------+--------+
# |NZ000093417|    PARAPARAUMU AWS|    -40.9|   174.983|NZM00093439|WELLINGTON AERO AWS|  -41.333|     174.8|   50.54|
# |NZM00093439|WELLINGTON AERO AWS|  -41.333|     174.8|NZM00093678|           KAIKOURA|  -42.417|     173.7|  151.12|
# |NZ000936150| HOKITIKA AERODROME|  -42.717|   170.983|NZM00093781|  CHRISTCHURCH INTL|  -43.489|   172.532|  152.31|
# |NZM00093678|           KAIKOURA|  -42.417|     173.7|NZM00093781|  CHRISTCHURCH INTL|  -43.489|   172.532|  152.51|
```

Q3 Understanding the concept of parallelism and distribution in Spark

The default block size of HDFS is 134,217,728 bytes or 128 MB.

The file size of daily climate summaries for 2022 is about 25,985,757 bytes. As the file size is less than the default block size of 128 MB, we just need 1 block for daily climate summaries for year 2022.

For the year 2021, the file size is 146,936,025 bytes. Since the file size is more than the default block size of 134 MB, we need 2 data blocks for daily climate summaries for year 2021; 1 block of 128 MB and the remaining 12,718,297 bytes in the 2nd block. The block sizes were confirmed in hdfs using fsck command.

```
Input:
hdfs fsck /data/ghcnd/daily/2022.csv.gz -files –blocks
Output:
/data/ghcnd/daily/2022.csv.gz 25985757 bytes, replicated: replication=8, 1 block(s):  OK
0. BP-700027894-132.181.129.68-1626517177804:blk_1073769759_28939 len=25985757 Live_repl=8
```

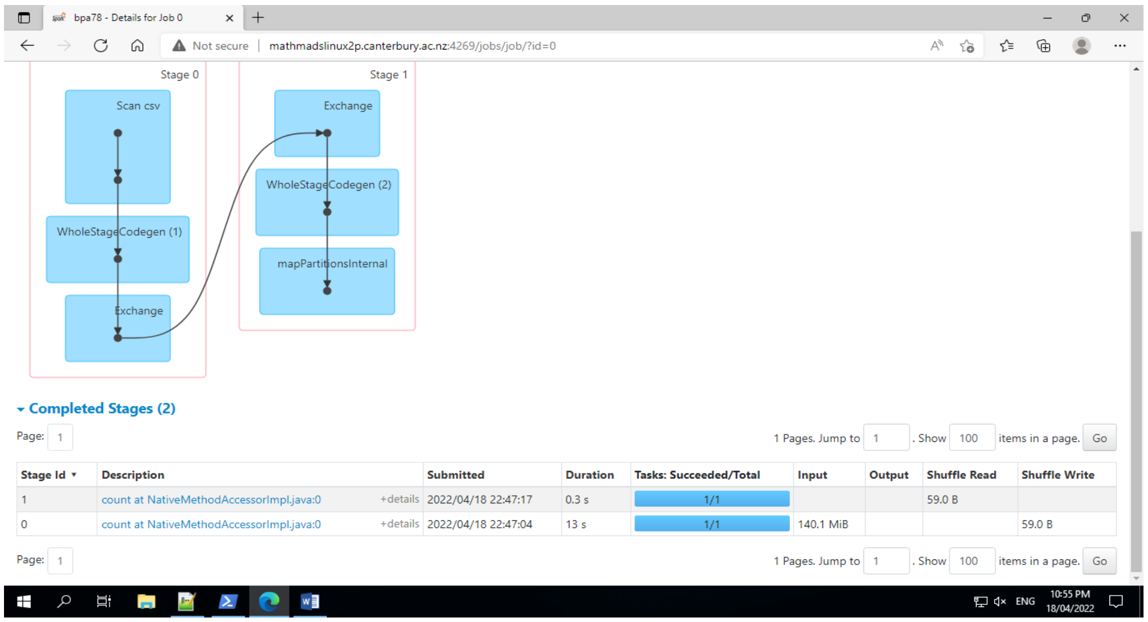
#What are the individual block sizes for the year 2021?

```
hdfs fsck /data/ghcnd/daily/2021.csv.gz -files -blocks
#/data/ghcnd/daily/2021.csv.gz 146936025 bytes, replicated: replication=8, 2 block(s):  OK
#0. BP-700027894-132.181.129.68-1626517177804:blk_1073769757_28937 len=134217728 Live_repl=8
#1. BP-700027894-132.181.129.68-1626517177804:blk_1073769758_28938 len=12718297 Live_repl=8
```

The data is loaded for the year 2021 and 2022 of daily climate summaries separately. There are 34,657,282 observations in 2021 and 5,971,307 observations for 2022.

It is not possible for spark to load and apply transformations in parallel for both year 2021 and 2022. To perform this job, we need 2 different stages. Each stage has one task in it, and that task has a single partition. As stage2's input is dependent on stage1's output, so it will be executed on the completion of stage1.

The task is the smallest execution unit where it executes a series of instructions like reading, filtering and applying map() on data blocks. Each task has one partition. Tasks are executed by executors. The number of partitions should be at least the number of cores per executors in the cluster or in multiples. A stage has several tasks and a job in turn has several stages. While reading a dataset, be it an initialisation of parameters, a new Resultstage is created and when the transformation functions like reduceByKey() or join() triggers a shuffle that will also create a new stage, which is called as Shufflestage. These stages are controlled by Directed Acyclic Graph (DAG). We can take a look at DAG to see these stages in the web interface (mathmadslinux2p:8080) while loading the daily climate summary for year 2021.



Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	count at NativeMethodAccessorImpl.java:0 +details	2022/04/18 22:47:17	0.3 s	1/1			59.0 B	
0	count at NativeMethodAccessorImpl.java:0 +details	2022/04/18 22:47:04	13 s	1/1	140.1 MiB			59.0 B

Figure. DAG shows the details for loading the daily climate summary for year 2021.

From the above figure, we can see that this job has 2 stages, for each stage there was one task. One task that shows we have one partition. In stage1, there is a task in Shuffle write after it was executed successfully and its output is sent as input to the stage2 with 1 task for Shuffle Read. Both the tasks are completed successfully. We have 2 data blocks, 1 input split, 1partition. It has 34,657,282 observations in it.

Loading the data for year 2022, RDD divides the data into one block. When number of blocks is greater than one or equal to one by default no partitions are created and hence no parallelism. According to DAG it has 2 stages. One task per each stage has only one partition. Stage1 has only 1 task, that means 1 execution per 1 partition since we are loading a single file. However, the input data size is just 24.8 MiB. It required only one data block, as it is less than the size of default block size. It has 5,971,307 observations in it. As we have seen earlier, it proves that the number of tasks executed does not correspond to the number of blocks in each input.

In order to load the daily climate summaries from the year 2014 to 2022, glob pattern is used in the path argument like 20{14,15,16,17,18,19,20,21,22}. There were 284,918,108 observations. From the DAG, there were two stages. In stage1, we had 9 tasks since we have 9 files to load and in stage2 there was 1 task. Both the stages were successfully completed. The input size of data was 1386.8 MiB. We have 9 tasks that were executed by each stage as we had 9 compressed files, one task per each stage. Therefore, we have 9 tasks that were executed by stage1.

As we have seen, the number of partitions should be at least equal to the number of cores per executors in the cluster or in multiples. The number of partitions should not be too high or too low. If an RDD has too many partitions, than task scheduling may take more time than the actual execution time. On contrary, if we have too less partitions then it is also not beneficial as some of the worker nodes could just be sitting idle resulting in less concurrency. Therefore, it is always better to use the same number of partitions as number of cores.

For the years 2022 and 2021, there were only one stage, one task. One task means one execution per partition for each year. Since it has one partition, they cannot be run in parallel. Compressed gzip files cannot be split. Since we are using 4 executors, 2 cores, we get 8 cores therefore we have 8 partitions for 8 blocks that can run in parallel. Hence tasks would run in parallel when loading and applying transformations to daily from year 2014 to 2022.

Q4 Finding insights

There are 3,000,243,596 rows in daily.

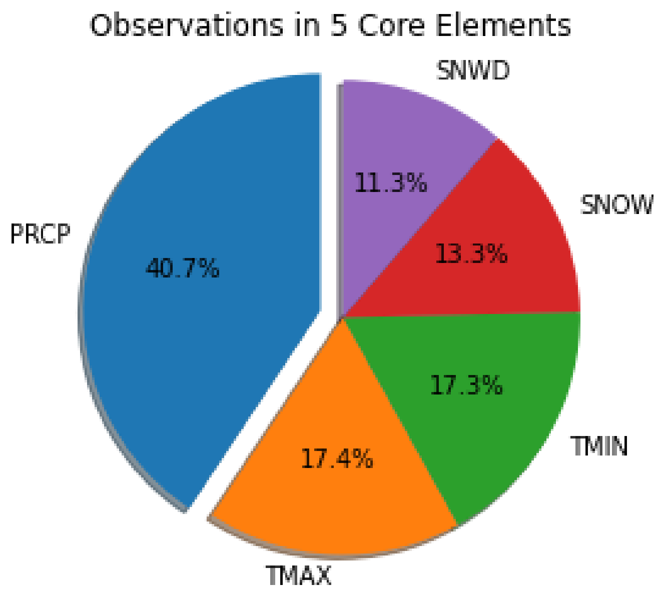


Figure. The observations containing 5 core elements.

Using the daily data, we now filter out the ELEMENT column for the five core elements TMAX, TMIN, PRCP, SNOW and SNWD to get an aggregated value for each of these. PRCP had most of the observations 1,043,785,667, TMAX has 445,623,712, TMIN has 444,271,327, SNOW has 341,985,067 and SNWD has 289,981,374 records.

About 8,808,805 observations has only TMIN but not TMAX, and 27,650 stations have contributed to these observations. These stations might had an issue of data collection or coverage. These result were obtained by flattening the data from long to wide format using pivot() and populating these new columns with the aggregated value of TMIN and TMAX. Then this dataframe is filtered by checking for TMIN is not null and TMAX is null to get the stations based on the distinct value of an ID.

```
TMIN_TMAX = (
   daily_all
   .select('ID', 'ELEMENT', 'DATE')
   .filter(F.col('ELEMENT').isin('TMAX','TMIN'))
   .groupBy('ID','DATE')
   .pivot('ELEMENT')
   .agg({'ELEMENT':'count'})
)
```

By filtering the TMIN and TMAX for all the stations in New Zealand, 472,271 observations for 83 years, i.e. from 1940 to 2022 were obtained. These 83 different files were copied to a local home directory using hdfs dfs –copyToLocal command. The wc –l command is used to count the number of rows in the directory. A total of 472,354 observations when compared to spark output, there was 83 observations more. These 83 rows were related to column header for 83 different csv files. These were further used for plotting. Hence, the outputs directory had 7 sub directories in it.

```
 #total number of files and folders in outputs/ghcnd folder
[…..@mathmadslinux2p ~]$ hdfs dfs -ls /user/bpa78/outputs/ghcnd
# Found 7 items
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-21 11:17 /user/bpa78/outputs/ghcnd/Stations
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-20 23:44 /user/bpa78/outputs/ghcnd/Stations.csv.gz
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-23 21:56 /user/bpa78/outputs/ghcnd/TMIN_TMAX_NZ
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-23 22:39 /user/bpa78/outputs/ghcnd/avgRainfallCountry
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-21 15:53 /user/bpa78/outputs/ghcnd/countries
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-23 19:14 /user/bpa78/outputs/ghcnd/distance_NZstations
# drwxr-xr-x   - bpa78 bpa78          0 2022-04-21 17:41 /user/bpa78/outputs/ghcnd/states
```

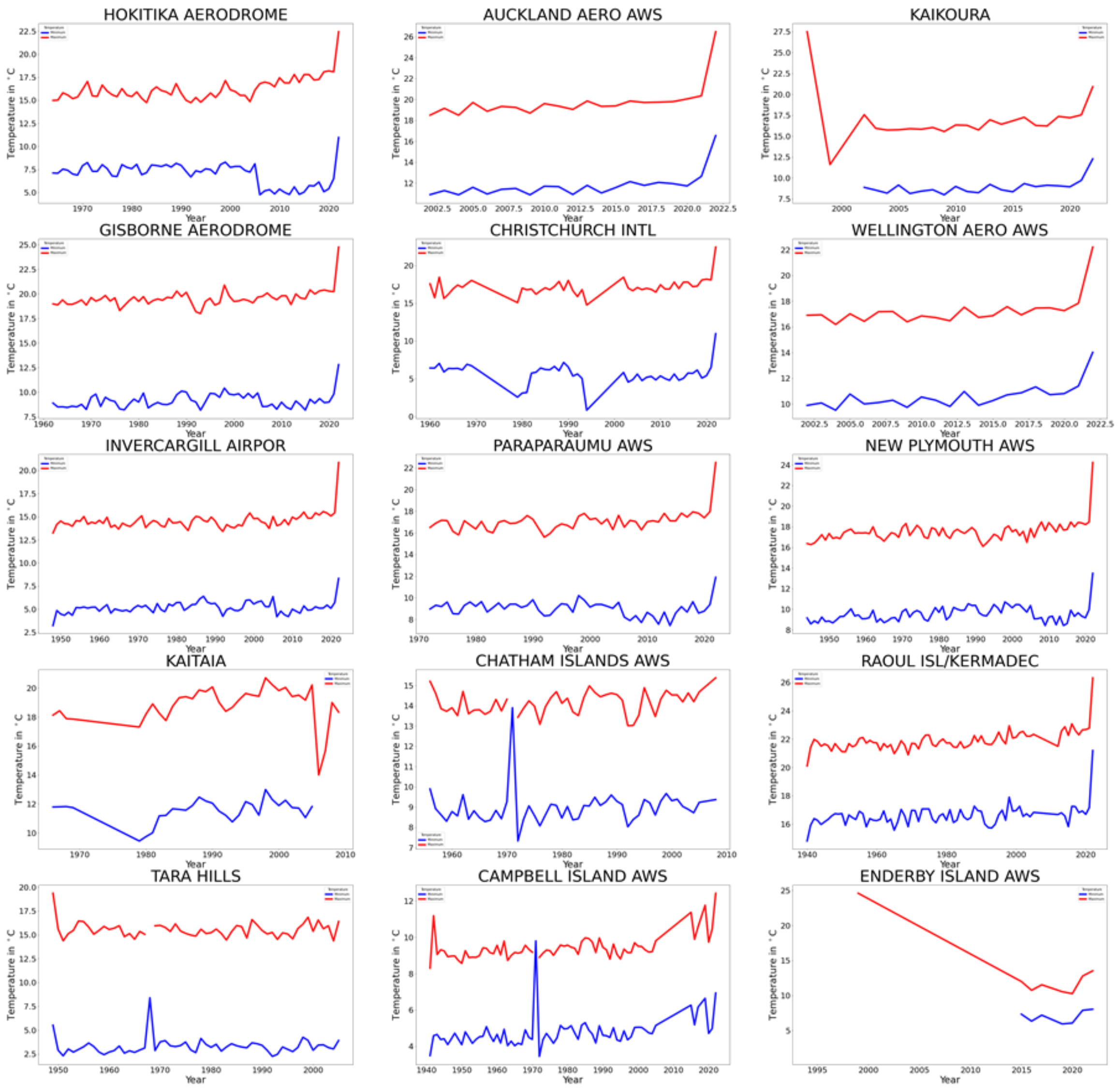


Figure. Minimum and maximum temperature for each station in New Zealand.

There are 15 different stations in New Zealand, which covered over a period of time when they are active. The minimum and maximum temperatures recorded are plotted for each station separately using a time series plot in Python. Here most of the stations show same pattern for both the maximum and minimum temperatures, but there values vary accordingly. If observed, most of the stations show a stable temperature, but from the year 2005, we can see a trend of increase in maximum temperature and decrease in minimum temperatures. The Tara hills and Chatham Island stations has some missing values in maximum temperatures. In the year 1970, there is a large peak of minimum average temperature for Chatham Island AWS and Campbell Island AWS stations. For a period around year 2005 and 2015, there were no values recorded for both maximum temperature and minimum temperature for Campbell Island AWS station; this might be due to a problem in the station for data collection. Some of the stations like Auckland Aero AWS, Kaikora, Wellington Aero AWS and Enderby Island AWS has only started recording the temperatures from year 2000. All the stations recorded a high value, as the data is incomplete for year 2022 since we are still in second quarter of 2022.

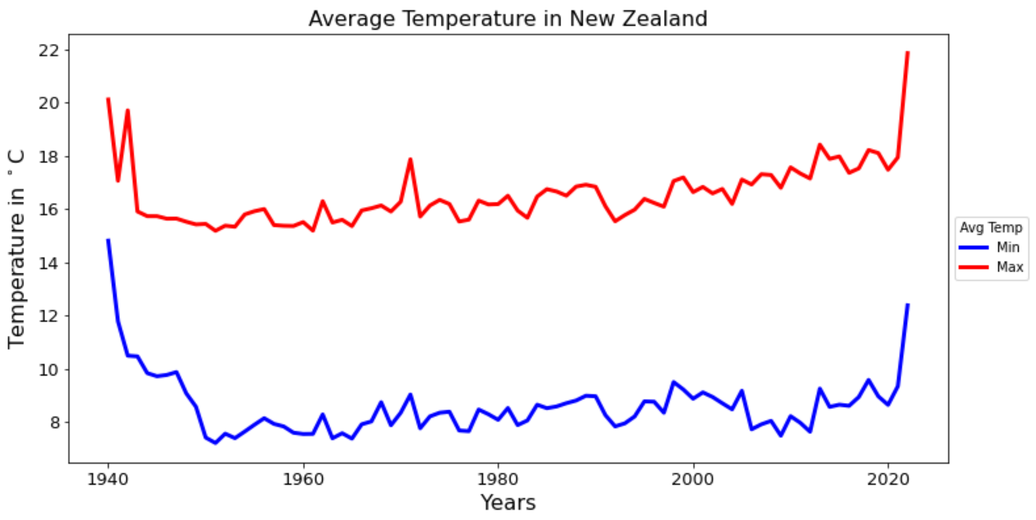


Figure. Average temperatures recorded in New Zealand.

From the above plot, we can see that the New Zealand's average maximum and minimum temperatures show similar pattern over the last eighty-two years. The temperature is converted to degree Celsius. The years ranged between 1940 and 2022. The overall minimum average temperature in New Zealand is between 14 ^{o}C to 7 ^{o}C and the maximum average temperature is between 22 ^{o}C to 1 5^{o}C. However, we can see from year 2000 the maximum average temperature has increased with a slight decrease in minimum average temperature; this temperature pattern could be due to global warming. The spike at the end of 2022 is due to the incompletion of the data.

To get the average rainfall in each year for each country, we need to get year from the DATE column and Country ID or country name. We can get Country name by joining it with stations data, in order to reduce the cost; splitting the country_code from ID by using substr(1, 2) since the daily dataframe has ID. The DATE column which is of StringType by using substr(1, 4), we get YEAR. Filtering for PRCP in ELEMENT and then group by country and year to find average value. These observations were saved as csv files. As there were 16 partitions, we have 16 files, one file for each partition.

Country code: EK - Equatorial Guinea is the country, which has the highest average rainfall in year 2000 as it is in the equatorial region. Yes, we can see it as an anomaly because its mean annual precipitation is only 2193.5 mm.

```
# +------------+----+-----------------+
# |COUNTRY_CODE|YEAR|      AVG_RAINFALL|
# +------------+----+-----------------+
# |          EK|2000|           4361.0|
# |          DR|1975|           3414.0|
# |          LA|1974|           2480.5|
# |          BH|1978| 2244.714285714286|
```

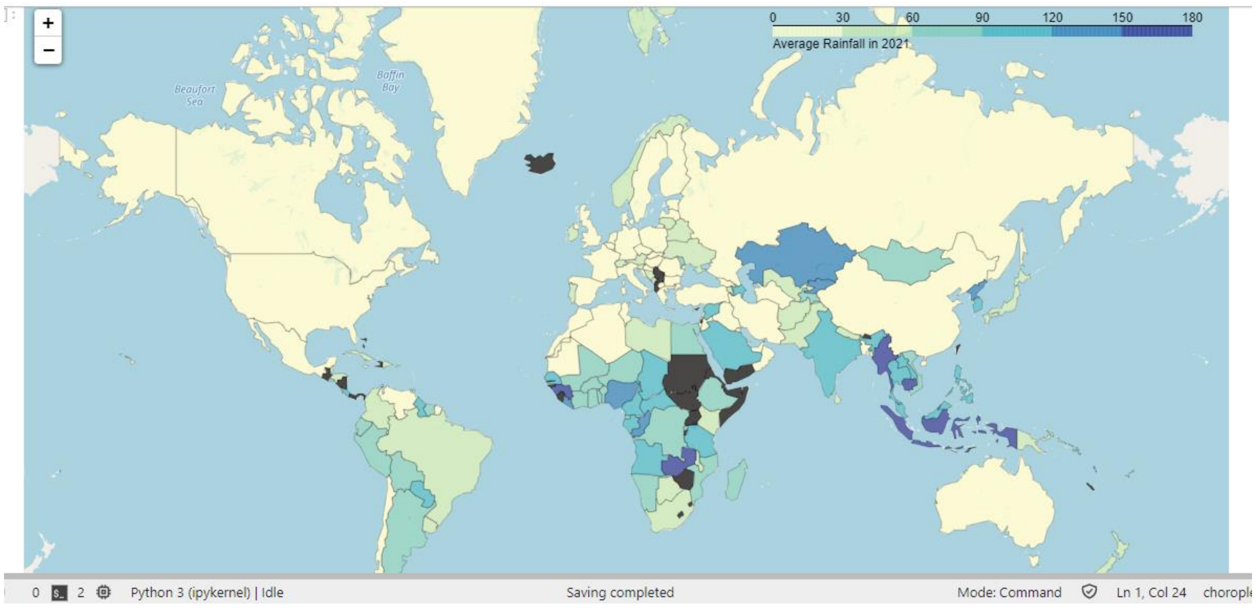


Figure: Choropleth map shows the average rainfall measured in 10th of mm for year 2021.

In the choropleth map, the regions are shaded according to the average rainfall. This visualisation helps us to represent the variability of average rainfall across a region using a sequential colour scheme. The light yellow regions represent low or no average rainfall, whereas the dark green regions represent moderate average rainfall and the dark blue region represents the highest average rainfall. This choropleth map is built in Python using Folium that requires Geo JSON file, which includes geospatial data of the region. The country codes used in GHCND is not the same as in Geo JSON file. Geo JSON has ISO-3 character codes for countries and these are matched to the GHCND data based on the Country Names and then used for plotting. The countries like Zambia, Myanmar, Indonesia, Philippines and Malaysia have highest average rainfall. The countries like USA, Greenland, China, Russia and Australia have lowest average rainfall, which is misleading. Since in our analysis, we found that only 38,284 stations were active in 2021 out of 118493 stations. The COVID-19 pandemic could be one of the other reason where the stations had difficulty in data collection. Some of the countries that are tiny, third or fourth world may not have weather stations to collect the data.

Conclusion

The GHCN data for daily and metadata stations, states, inventory and countries were explored. Found the file sizes of the data and used them appropriately in processing and analysing in order to use proper joins. The daily data set is of 99.97 % of the total GHCN data. Exploring the size of the file helped to decide the resource allocation in Spark cluster. Some of the functions like left join, groupBy, aggregate were applied to the dataset to transform all the metadata to a single dataset. The user defined function was created using Haversine formula to find the closest stations in New Zealand. The temperature filtered for New Zealand is converted to degree Celsius for TMIN and TMAX since it is referred as Celsius here in New Zealand. To present a visualisation of temperature for each station in New Zealand separately, and an average maximum and minimum temperature for the whole New Zealand a times series plots are used. An average rainfall for all the countries is shown in choropleth map. Further improvements can be made to the temperature time series plots by smoothing the average temperature of min of max and max of max in order to create a quantile that could be shaded and a solid line for the actual average temperature in the middle. These plots can be used for predicting the temperature in New Zealand.

References

RDD programing guide

https://spark.apache.org/docs/latest/rdd-programming-guide.html

Style guide showing the bad way and good way for writing code in Pyspark

https://github.com/palantir/pyspark-style-guide

Haversine formula to calculate geographic distance between two places

https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/