GHCN Data Analysis using PySpark

Pengcheng Wu

Contents

Foreword ... 2
Processing ... 3
Q1 ... 3
Q2 ... 4
Q3 ... 5
Analysis ... 9
Q1 ... 10
Q2 ... 13
Q3 ... 16
Q4 ... 19
Conclusion ... 23
Reference ... 24

Foreword

We will study and use Spark to explore and analyse the weather data included in the Global Historical Climate Network (GHCN). GHCN is a comprehensive database containing daily climate summaries from surface weather stations worldwide. It consists of daily climate records from multiple sources consolidated and subject to a standard set of quality assurance reviews (refer from NCEI-GHCN). This database includes papers from 118493 sites in 219 countries and territories covering the last 175 years and is from over 20 independent sources (Menne,2012). It provides several daily variables, including maximum and minimum temperatures, total daily precipitation, snowfall and snow depth. Around half of all-weather stations report only rain. The length and period of records vary from station to station, with coverage intervals ranging from less than one year to 175 years.

Spark as a tool is to run distributing computations over the large data sets. Pyspark is an interface for Apache Spark in Python. It can provide Spark application using Python APIs with users to write and contains the shell for interactively analysing data in a distributed environment. This report will apply some of Spark's features like Spark SQL, Data Frame, Machine Learning, and Spark Core for further analysing of these target data. Also, we will use the plotting method to present the time series for TMIN and TMAX for each station in New Zealand and the average rainfall distribution in 2021 for each country.

Processing

Q1

The data can be divided into two parts. The one part is daily file representing daily climate summaries which are stored in files by years as csv formats with gzip compressed. The other information is saved as TXT formats, namely: countries, inventory, states, stations respectively. After using the command line -ls-R code, we can find the data structure and draw the directory tree (graph Q1-1) below:

```
/data/ghcnd/
├─ daily/
│  ├─ 1763.csv.gz
│  ├─ 1764.csv.gz
│  ├─ 1765.csv.gz
│  ├─ . . . . . .
│  ├─ 2021.csv.gz
│  └─ 2022.csv.gz
├─ ghcnd-countries.txt
├─ ghcnd-inventory.txt
├─ ghcnd-states.txt
├─ ghcnd-stations.txt
└─ readme.txt
```

Q1-1 directory tree for data sets

Daily directory contains relative daily climate information which are separately stored into solely csv file, and countries, inventory, states and stations are separately saved as txt files. There are 260 items which means there are 260 years included in daily directory. The columns names of each csv file as the assignment shown includes ID, DATE, ELEMENT, VALUE, MEASUREMENTS FLAG, QUALITY FLAG, SOURCE FLAG and OBSERVATION TIME, and each field is separated by comma.

The total size of all of the data is about 15.7GB and disk-space-consumed with replicas being 125.6GB. The sizes of metadata, (namely: countries, inventory, states, stations text files) occupy 3.6K, 30.9M, 1.1K, 9.7M respectively. The size of data is 3.3K in 1763, and the size of data in 140.1 in 2021. It is found that the size of daily files account for the most of total size, each year of file size tend to gradually increase and the difference changing apparently, shown as the below graph (Q1-2).

```
[pwu17@canterbury.ac.nz@mathmadslinux2p ~]$ hdfs dfs -du -h /data/ghcnd/
15.7 G  125.3 G  /data/ghcnd/daily
3.6 K   28.6 K   /data/ghcnd/ghcnd-countries.txt
30.9 M  247.4 M  /data/ghcnd/ghcnd-inventory.txt
1.1 K   8.5 K    /data/ghcnd/ghcnd-states.txt
9.7 M   77.7 M   /data/ghcnd/ghcnd-stations.txt
26.0 K  207.9 K  /data/ghcnd/readme.txt
```

Q1-2 data file size code and result

Q2

Start pyspark shell with 2 executors, 1 core per executor, 1G of executor memory and 1G of master memory.
According to the description of the README, the schema can be set up with ID, DATE, ELEMENT, VALUE, MFLAG, QFLAG, SFLAG and OBSERVATION TIME. Especially, because DATE column and OBSERVATION TIME column represents date and time respectively, the Date Type and Timestamp Type methods can be used to convert them manually. Using string type formatting way is in these five columns, namely ID, ELEMENT, MFLAG, QFLAG, and SFLAG, but the VALUE column needs float type formatting (as the below graph shows, Q2-1). Based on loading the first 1000 rows, the description of data in the below table shown is accurate. Under the limitation, the Observation time and QFLAG shows null in the top 20 rows.

ID	DATE	ELEMENT	VALUE	MFLAG	QFLAG	SFLAG	OBV_TIME
AE000041196	2022-01-01	TAVG	204.0	H	null	S	null
AEM00041194	2022-01-01	TAVG	211.0	H	null	S	null
AEM00041218	2022-01-01	TAVG	207.0	H	null	S	null
AEM00041217	2022-01-01	TAVG	209.0	H	null	S	null
AG000060390	2022-01-01	TAVG	121.0	H	null	S	null
AG000060590	2022-01-01	TAVG	151.0	H	null	S	null
AG000060611	2022-01-01	TAVG	111.0	H	null	S	null
AGE00147708	2022-01-01	TMIN	73.0	null	null	S	null
AGE00147708	2022-01-01	PRCP	0.0	null	null	S	null
AGE00147708	2022-01-01	TAVG	133.0	H	null	S	null
AGE00147716	2022-01-01	TMIN	107.0	null	null	S	null
AGE00147716	2022-01-01	PRCP	0.0	null	null	S	null
AGE00147716	2022-01-01	TAVG	133.0	H	null	S	null
AGE00147718	2022-01-01	TMIN	90.0	null	null	S	null
AGE00147718	2022-01-01	TAVG	152.0	H	null	S	null
AGE00147719	2022-01-01	TMAX	201.0	null	null	S	null
AGE00147719	2022-01-01	PRCP	0.0	null	null	S	null
AGE00147719	2022-01-01	TAVG	119.0	H	null	S	null
AGM00060351	2022-01-01	PRCP	0.0	null	null	S	null
AGM00060351	2022-01-01	TAVG	126.0	H	null	S	null

Q2-1 2022 daily data table 1000 rows

The schema for station, country, state and inventory can be utilised after schema for these data groups, setting structure type of each item, and then new data frame will be created with loading relevant data and let relevant lambda maps the corresponding columns (as the graph Q2-2 shows). After using count () function, the countries data has 219 rows in total meaning 219 countries or territories selected from around the world. The state data has 74 rows in total representing 74 states selected in the data, and the station data include 118493 rows and inventory data has 704963 rows. However, there are 110407 stations which do not hold WMO ID.

ID	LATITUDE	LONGITUDE	ELEVATION	STATE	NAME	GFLAG	HCFLAG	WMOID
ACW00011604	17.1167	-61.7833	10.1		ST JOHNS COOLIDGE...			
ACW00011647	17.1333	-61.7833	19.2		ST JOHNS			
AE000041196	25.333	55.517	34.0		SHARJAH INTER. AIRP	GSN		41196
AEM00041194	25.255	55.364	10.4		DUBAI INTL			41194
AEM00041217	24.433	54.651	26.8		ABU DHABI INTL			41217

CODE	COUNTRY_NAME
AC	Antigua and Barbuda
AE	United Arab Emirates
AF	Afghanistan
AG	Algeria
AJ	Azerbaijan

CODE	STATE_NAME
AB	ALBERTA
AK	ALASKA
AL	ALABAMA
AR	ARKANSAS
AS	AMERICAN SAMOA

ID	LAT	LONG	ELEMENT	FIRSTYEAR	LASTYEAR
ACW00011604	17.1167	-61.7833	TMAX	1949	1949
ACW00011604	17.1167	-61.7833	TMIN	1949	1949
ACW00011604	17.1167	-61.7833	PRCP	1949	1949
ACW00011604	17.1167	-61.7833	SNOW	1949	1949
ACW00011604	17.1167	-61.7833	SNWD	1949	1949

Q2-2 station, country, state and inventory data table

Q3

Following the Q2, we load the daily, station, country, state and inventory into the Spark and will combine these items using withColumn () and join () functions in the section. In the first, two-character country code from each station code in stations are extracted and store the output as a new column using withColumn () function, as the below graph Q3-1 shown:

ID	LATITUDE	LONGITUDE	ELEVATION	STATE	NAME	GFLAG	HCFLAG	WMOID	COUNTRY_CODE
ACW00011604	17.1167	-61.7833	10.1		ST JOHNS COOLIDGE...				AC
ACW00011647	17.1333	-61.7833	19.2		ST JOHNS				AC
AE000041196	25.333	55.517	34.0		SHARJAH INTER. AIRP	GSN		41196	AE
AEM00041194	25.255	55.364	10.4		DUBAI INTL			41194	AE
AEM00041217	24.433	54.651	26.8		ABU DHABI INTL			41217	AE
AEM00041218	24.262	55.609	264.9		AL AIN INTL			41218	AE
AF000040930	35.317	69.017	3366.0		NORTH-SALANG	GSN		40930	AF
AFM00040938	34.21	62.228	977.2		HERAT			40938	AF
AFM00040948	34.566	69.212	1791.3		KABUL INTL			40948	AF
AFM00040990	31.5	65.85	1010.0		KANDAHAR AIRPORT			40990	AF
AG000060390	36.7167	3.25	24.0		ALGER-DAR EL BEIDA	GSN		60390	AG
AG000060590	30.5667	2.8667	397.0		EL-GOLEA	GSN		60590	AG
AG000060611	28.05	9.6331	561.0		IN-AMENAS	GSN		60611	AG
AG000060680	22.8	5.4331	1362.0		TAMANRASSET	GSN		60680	AG
AGE00135039	35.7297	0.65	50.0		ORAN-HOPITAL MILI...				AG
AGE00147704	36.97	7.79	161.0		ANNABA-CAP DE GARDE				AG
AGE00147705	36.78	3.07	59.0		ALGIERS-VILLE/UNI...				AG
AGE00147706	36.8	3.03	344.0		ALGIERS-BOUZAREAH				AG
AGE00147707	36.8	3.04	38.0		ALGIERS-CAP CAXINE				AG
AGE00147708	36.72	4.05	222.0		TIZI OUZOU			60395	AG

Q3-1 Extracting two character in one new column using withColumn command

Using LEFT JOIN to combine the above output with countries and states, we use the join command and join on the 'COUNTRY_CODE' or 'STATE' index in right with using left frame's index. As the previous seeing the metadata tables, we found some column names are the same in these tables and utilised the withColumnRenamed () function to change these names for distinguishing them, as the below code snippet and a new joining table.

```
#(b)
station_new = station_new.join(country.withColumnRenamed('CODE','COUNTRY_CODE')
                              .withColumnRenamed('NAME','COUNTRY_NAME'), on='COUNTRY_CODE', how="left")
# station_new.cache()
# station_new.show()

#(c)
station_new = station_new.join(state.withColumnRenamed('CODE','STATE')
                              .withColumnRenamed('NAME','STATE_NAME'), on = 'STATE', how= "left")
# station_new.cache()
# station_new.show()
```

STATE	COUNTRY_CODE	ID	LATITUDE	LONGITUDE	ELEVATION	NAME	GFLAG	HCFLAG	WMOID	COUNTRY_NAME	STATE_NAME
	AC	ACW00011604	17.1167	-61.7833	10.1	ST JOHNS COOLIDGE...				Antigua and Barbuda	null
	AC	ACW00011647	17.1333	-61.7833	19.2	ST JOHNS				Antigua and Barbuda	null
	AE	AE000041196	25.333	55.517	34.0	SHARJAH INTER. AIRP	GSN		41196	United Arab Emirates	null
	AE	AEM00041194	25.255	55.364	10.4	DUBAI INTL			41194	United Arab Emirates	null
	AE	AEM00041217	24.433	54.651	26.8	ABU DHABI INTL			41217	United Arab Emirates	null
	AE	AEM00041218	24.262	55.609	264.9	AL AIN INTL			41218	United Arab Emirates	null
	AF	AF000040930	35.317	69.017	3366.0	NORTH-SALANG	GSN		40930	Afghanistan	null
	AF	AFM00040938	34.21	62.228	977.2	HERAT			40938	Afghanistan	null
	AF	AFM00040948	34.566	69.212	1791.3	KABUL INTL			40948	Afghanistan	null
	AF	AFM00040990	31.5	65.85	1010.0	KANDAHAR AIRPORT			40990	Afghanistan	null
	AG	AG000060390	36.7167	3.25	24.0	ALGER-DAR EL BEIDA	GSN		60390	Algeria	null
	AG	AG000060590	30.5667	2.8667	397.0	EL-GOLEA	GSN		60590	Algeria	null
	AG	AG000060611	28.05	9.6331	561.0	IN-AMENAS	GSN		60611	Algeria	null
	AG	AG000060680	22.8	5.4331	1362.0	TAMANRASSET	GSN		60680	Algeria	null
	AG	AGE00135039	35.7297	0.65	50.0	ORAN-HOPITAL MILI...				Algeria	null
	AG	AGE00147704	36.97	7.79	161.0	ANNABA-CAP DE GARDE				Algeria	null
	AG	AGE00147705	36.78	3.07	59.0	ALGIERS-VILLE/UNI...				Algeria	null
	AG	AGE00147706	36.8	3.03	344.0	ALGIERS-BOUZAREAH				Algeria	null
	AG	AGE00147707	36.8	3.04	38.0	ALGIERS-CAP CAXINE				Algeria	null
	AG	AGE00147708	36.72	4.05	222.0	TIZI OUZOU			60395	Algeria	null

Q3-2 Join the station_new table into countries and states and set up new one

The inventory part in the metadata includes nine items, namely: ID, Latitude, Longitude, Element, First Year and Last Year. The first step is to find the first and last year with each active station based on the inventory. We can group the inventory according to ID items and aggregate it by the First Year and Last Year by using an alias () to rename the columns, and the final part is to sort it according to ID. We can utilise the count distinct () function to distinct count these number of elements by grouping ID with changing name and sorting it according to ID. We can get the number of different elements has each station collected representing the number of rows of each station. The code and tables are shown in graph Q3-3. All elements in each station should be set up with grouping ID and aggregating the items after collecting elements.

```
station_years = (invt.groupBy('ID')
                 .agg(F.min(invt.FIRSTYEAR).alias('Firstyear'),F.max(invt.LASTYEAR).alias('Lastyear'))
                 .sort(F.col('ID')))
station_element = invt.groupBy("ID").agg(F.countDistinct("ELEMENT").alias("ELEMENT_NUMBER")).sort(F.col("ID"))
# station_element.cache()
# station_element.show()
```

ID	Firstyear	Lastyear
ACW00011604	1949	1949
ACW00011647	1957	1970
AE000041196	1944	2021
AEM00041194	1983	2021
AEM00041217	1983	2021
AEM00041218	1994	2021
AF000040930	1973	1992
AFM00040938	1973	2021
AFM00040948	1966	2021
AFM00040990	1973	2020
AG000060390	1940	2021
AG000060590	1892	2021
AG000060611	1958	2021
AG000060680	1940	2004
AGE00135039	1852	1966
AGE00147704	1909	1937
AGE00147705	1877	1938
AGE00147706	1893	1920
AGE00147707	1878	1879
AGE00147708	1879	2021

ID	ELEMENT_NUMBER
ACW00011604	11
ACW00011647	7
AE000041196	4
AEM00041194	4
AEM00041217	4
AEM00041218	4
AF000040930	5
AFM00040938	5
AFM00040948	5
AFM00040990	5
AG000060390	5
AG000060590	4
AG000060611	5
AG000060680	4
AGE00135039	3
AGE00147704	3
AGE00147705	3
AGE00147706	3
AGE00147707	3
AGE00147708	5

ID	ELEMENT_LIST
ACW00011604	[TMAX, TMIN, WSFG...
ACW00011647	[TMAX, TMIN, PRCP...
AE000041196	[TMAX, TMIN, PRCP...
AEM00041194	[TMAX, TMIN, PRCP...
AEM00041217	[TMAX, TMIN, PRCP...
AEM00041218	[TMAX, TMIN, PRCP...
AF000040930	[TMAX, TMIN, PRCP...
AFM00040938	[TMAX, TMIN, PRCP...
AFM00040948	[TMAX, TMIN, PRCP...
AFM00040990	[TMAX, TMIN, PRCP...
AG000060390	[TMAX, TMIN, PRCP...
AG000060590	[TMAX, TMIN, PRCP...
AG000060611	[TMAX, TMIN, PRCP...
AG000060680	[TMAX, TMIN, PRCP...
AGE00135039	[TMAX, TMIN, PRCP]
AGE00147704	[TMAX, TMIN, PRCP]
AGE00147705	[TMAX, TMIN, PRCP]
AGE00147706	[TMAX, TMIN, PRCP]
AGE00147707	[TMAX, TMIN, PRCP]
AGE00147708	[TMAX, TMIN, PRCP...

Q3-3 the years, element number and element items included in each station in inventory

In the second step, core elements include "PRCP", "SNOW", "SNWD", "TMAX" and "TMIN". We can make the filter () function to separately filter out the core elements and other elements, and group the inventory ID and then aggregate the number of specific elements after counting and renaming, final table shown as Q3-4 graph.

```
In [35]: core_element = invt.filter(invt.ELEMENT.isin(coreElements)).groupBy("ID").agg(F.countDistinct("ELEMENT").alias('num_cElement'))

In [36]: core_element.show()
+-----------+-----------+
|         ID|num_cElement|
+-----------+-----------+
|ALE00100939|          2|
|AQC00914873|          5|
|AR000000002|          1|
|AR000087374|          4|
|ARM00087480|          4|
|ARM00087509|          4|
|ASN00001020|          3|
|ASN00002033|          1|
|ASN00006036|          1|
|ASN00006080|          3|
|ASN00007039|          1|
|ASN00007187|          1|
|ASN00008093|          3|
|ASN00009073|          1|
|ASN00009186|          3|
|ASN00009603|          3|
|ASN00009802|          1|
|ASN00009916|          1|
|ASN00010146|          1|
|ASN00010529|          1|
+-----------+-----------+
In [37]: other_element = invt.filter(~invt.ELEMENT.isin(coreElements)).groupBy("ID").agg(F.countDistinct("ELEMENT").alias('num_nElement'))

In [38]: other_element.show()
+-----------+-----------+
|         ID|num_nElement|
+-----------+-----------+
|US1COAR0249|          4|
|US1COBN0028|          2|
|US1COBO0165|          4|
|US1COBO0298|          4|
|US1COBO0477|          4|
|US1COCF0008|          4|
|US1COCF0015|          4|
|US1COCF0038|          2|
|US1COCF0043|          4|
|US1CODG0035|          4|
|US1CODG0059|          2|
|US1CODG0226|          4|
|US1COEG0017|          4|
|US1COEG0044|          3|
|US1COEP0416|          4|
|US1COFM0076|          4|
|US1COGF0049|          2|
|US1COHF0004|          4|
|US1COJF0060|          4|
|US1COJF0124|          4|
+-----------+-----------+
```

Q3-4 the tables of core-elements and other elements

There are 20289 stations collecting all five core elements. We can use filter function according to ELEMENT item to confirm whether or not the items would be included in core elements and group the inventory by ID and then aggregated the distinctly count the number of specific elements.

```
invt.filter(F.col('ELEMENT').isin(coreElements)).groupBy("ID").agg(F.countDistinct("ELEMENT").alias("count")).filter("count = 5").count()
20289
```

Based on the above element list in each station, it is necessary to make sure whether or not the ['PRCP'] could be included in the element list and count the rows. We can find that 16136 stations only collect precipitation.

```
station_element_list.filter(F.col('ELEMENT_LIST') == '[PRCP]').count()
16136
```

The third part is to join the items about the first and last year with each active station, the number of elements and element list included in each station of each station, and the number of core and other elements in the new station data sets based on each 'ID' and LEFT joining. The new station table (as shown below in picture Q3-5) is saved into the user's own output directory as "csv.gz" because the size of the file is big. When selecting the method to save the file, we need to think about consistency and efficiency, namely, the reading and writing process speed. In this case, compressed files should be considered compared with CSV format because (1) compressed files contain fewer bytes of data than uncompressed files; (2) the transfer speed. (3) the cost of storing the data is reduced by compressing files for storage, which will be helpful to us storing in limiting storage. However, this format would require a full data scan or read to load a subset of columns, and the next analysis may cause inefficiencies. Compared with the parquet method, a gzip-compressed file will cost less to decode and support common languages and frameworks. This is why we choose the compressing method to store the file of data sets in this case.

STATION_ID	STATE	COUNTRY_CODE	LATITUDE	LONGITUDE	ELEVATION	NAME	GFLAG	HCFLAG	WMOID	COUNTRY_NAME	STATE_NAME	Firstyear	Lastyear	ELEMENT_NUMBER	ELEMENT_LIST	num_cElement	num_nElement
AGE00147719		AG	33.7997	2.89	767.0	LAGHOUAT			60545	Algeria	null	1888	2021	4	[TMAX, TMIN, PRCP...	3	1
AGM00060445		AG	36.178	5.324	1050.0	SETIF AIN ARNAT			60445	Algeria	null	1957	2021	5	[TMAX, TMIN, PRCP...	4	1
AJ000037679		AJ	41.1	49.2	-26.0	SIASAN'			37679	Azerbaijan	null	1959	1987	1	[PRCP]	1	null
AJ000037831		AJ	40.4	47.0	160.0	MIR-BASHIR			37831	Azerbaijan	null	1955	1987	1	[PRCP]	1	null
AJ000037981		AJ	38.9	48.2	794.0	JARDIMLY			37981	Azerbaijan	null	1959	1987	1	[PRCP]	1	null
AJ000037989		AJ	38.5	48.9	-22.0	ASTARA	GSN		37989	Azerbaijan	null	1936	2017	5	[TMAX, TMIN, PRCP...	4	1
ALE00100939		AL	41.3331	19.7831	89.0	TIRANA				Albania	null	1940	2000	2	[TMAX, PRCP]	2	null
AM000037719		AM	40.6	45.35	1834.0	CHAMBARAK			37719	Armenia	null	1912	1992	5	[TMAX, TMIN, PRCP...	4	1
AM000037897		AM	39.533	46.017	1581.0	SISIAN			37897	Armenia	null	1936	2021	5	[TMAX, TMIN, PRCP...	4	1
AQC00914873	AS	AQ	-14.35	-170.7667	14.9	TAPUTIMU TUTUILA				American Samoa [U...	AMERICAN SAMOA	1955	1967	12	[WT03, TMAX, TMIN...	5	7
AR000000002		AR	-29.82	-57.42	75.0	BONPLAND				Argentina	null	1981	2000	1	[PRCP]	1	null
AR000087007		AR	-22.1	-65.6	3479.0	LA QUIACA OBSERVATO	GSN		87007	Argentina	null	1956	2021	5	[TMAX, TMIN, PRCP...	4	1
AR000087374		AR	-31.783	-60.483	74.0	PARANA AERO	GSN		87374	Argentina	null	1956	2021	5	[TMAX, TMIN, PRCP...	4	1
AR000875850		AR	-34.583	-58.483	25.0	BUENOS AIRES OBSERV			87585	Argentina	null	1908	2021	5	[TMAX, TMIN, PRCP...	4	1
ARM00087022		AR	-22.62	-63.794	449.0	GENERAL ENRIQUE M...			87022	Argentina	null	1973	2021	4	[TMAX, TMIN, PRCP...	3	1
ARM00087480		AR	-32.904	-60.785	25.9	ROSARIO			87480	Argentina	null	1965	2021	5	[TMAX, TMIN, PRCP...	4	1
ARM00087509		AR	-34.588	-68.403	752.9	SAN RAFAEL			87509	Argentina	null	1973	2021	5	[TMAX, TMIN, PRCP...	4	1
ARM00087532		AR	-35.696	-63.758	139.9	GENERAL PICO			87532	Argentina	null	1973	2021	5	[TMAX, TMIN, PRCP...	4	1
ARM00087904		AR	-50.267	-72.05	204.0	EL CALAFATE AERO			87904	Argentina	null	2003	2021	5	[TMAX, TMIN, PRCP...	4	1
ASN00001003		AS	-14.1331	126.7158	5.0	PAGO MISSION				Australia	null	1909	1940	1	[PRCP]	1	null

Q3-5 The final joining new data frame

Also, we can use the left join to combine the 1000 rows subset of daily with the above output of the new station table. Using left anti join is to select in the case, and using the count function presents the stations in a subset of days that are not in stations at all, and the counting result is zero (Pipis, G, 2021). Namely, the output from Left anti method is the rows in the left side that are not in the right side. It would confirm that there are null stations in the subset of daily that are not in stations. Just pick up the left part; we can find the count is zero, which can confirm there are null any stations in the subset of daily that are not in stations at all. However, the cost of knowing the above function using different kinds of joins is based on how many rows the station data frame has. It is ensured that the smaller shuffle will lead to less cost. Shuffle is to move data with the same key collected into one executor for executing some specific processing on it. Joining the datasets requires many data moving across executors to make sure rows with matching join-keys get on the same node. Left Join will pick more joined results than anti-left to pick only join items from the stations rather than inventory. Another option is making both the daily and station id as a hash table (such as set in Python) and doing a subset of them. **Set(daily_id) - Set(station_id) **.

```
In [81]: daily_station = daily.join(station_new, daily.ID == station_new.STATION_ID, "leftanti")

In [82]: daily_station.count()
Out[82]: 0

In [83]: daily_station.show()
+---+----+-------+-----+-----+-----+-----+--------+
| ID|DATE|ELEMENT|VALUE|MFLAG|QFLAG|SFLAG|OBV_TIME|
+---+----+-------+-----+-----+-----+-----+--------+
+---+----+-------+-----+-----+-----+-----+--------+
```

Analysis

In this section, the resources should be increased up to 4 executors, 2 cores per executor, 4 GB of executor memory, and 4 GB of master memory. The code is utilised in “start_pyspark_shell -e 4 -c 2 -w 4 -m 4”.

```
In [1]: sc.getConf().getAll()
Out[1]:
[('spark.dynamicAllocation.enabled', 'false'),
 ('spark.executor.instances', '4'),
 ('spark.driver.extraJavaOptions',
  '-Dderby.system.home=/tmp/pwu17@canterbury.ac.nz/pyspark/pwu17'),
 ('spark.driver.port', '44321'),
 ('spark.sql.warehouse.dir', 'file:/users/home/pwu17/spark-warehouse'),
 ('spark.app.id', 'app-20220425160329-0802'),
 ('spark.repl.local.jars', 'file:///opt/spark/jars/hadoop-auth-3.2.0.jar'),
 ('spark.driver.memory', '4g'),
 ('spark.executor.memory', '4g'),
 ('spark.master', 'spark://masternode2:7077'),
 ('spark.executor.id', 'driver'),
 ('spark.app.startTime', '1650859407414'),
 ('spark.executor.cores', '2'),
 ('spark.driver.host', 'mathmadslinux2p.canterbury.ac.nz'),
 ('spark.app.name', 'pwu17'),
 ('spark.sql.shuffle.partitions', '16'),
 ('spark.ui.port', '4970'),
 ('spark.sql.catalogImplementation', 'hive'),
 ('spark.rdd.compress', 'True'),
 ('spark.app.initial.jar.urls',
  'spark://mathmadslinux2p.canterbury.ac.nz:44321/jars/hadoop-auth-3.2.0.jar'),
 ('spark.serializer.objectStreamReset', '100'),
 ('spark.driver.maxResultSize', '0'),
 ('spark.cores.max', '8'),
 ('spark.submit.pyFiles', ''),
 ('spark.submit.deployMode', 'client'),
 ('spark.jars', 'file:///opt/spark/jars/hadoop-auth-3.2.0.jar'),
 ('spark.ui.showConsoleProgress', 'true')]
```

Q1

According to the above new station data table, we can find there are 118493 stations in total, 38284 stations were active in 2021, 991 stations are in each of GSN, 1218 stations are in each of the HCN and null stations in each of the CRN, therefore, there are 14 stations in more than one of these networks, with following by code snippets Q1-1.

```
In [7]: station_new.count()
Out[7]: 118493

In [8]: station_new.filter(station_new.Lastyear ==2021).count()
Out[8]: 38284

In [9]: station_new.filter(station_new.GFLAG == 'GSN').count()
Out[9]: 991

In [10]: station_new.filter(station_new.HCFLAG == 'HCN').count()
Out[10]: 1218

In [11]: station_new.filter(station_new.HCFLAG == 'CRN').count()
Out[11]: 0

In [12]: station_new.filter((station_new.GFLAG == 'GSN')&(station_new.HCFLAG == 'HCN') |(station_new.HCFLAG == 'CRN')).count()
Out[12]: 14
```

Q1-1 COUNTING THE SEVERAL ITMES

The following part is to explore the total number of stations in each country. The first step is to count the number of stations in each country by grouping the new table in 'COUNTRY CODE' and count rows in each item. Secondly, we use Left Join () mapping on 'CODE' and withColumnRenamed () functions to creating one new table. The new data item data frame will be stored into the user own path. The same way is to create data frame about counting number of states in each country and store it, as the Q1-2 shown.

```
num_station_country = station_new.groupBy(F.col('COUNTRY_CODE')).count()
station_num_each_country = country.join(num_station_country
                                        .withColumnRenamed('count', 'STATION_NUM_EACH_COUNTRY')
                                        .withColumnRenamed('COUNTRY_CODE','CODE'),
                                        on = 'CODE',
                                        how = 'left')

# station_num_each_country.cache()
# station_num_each_country.show()
+----+--------------------+------------------------+
|CODE|        COUNTRY_NAME|STATION_NUM_EACH_COUNTRY|
+----+--------------------+------------------------+
|  AU|             Austria|                      13|
|  BA|             Bahrain|                       1|
|  BB|            Barbados|                       1|
|  CQ|Northern Mariana ...|                      11|
|  DR|  Dominican Republic|                       5|
|  EU|Europa Island [Fr...|                       1|
|  EZ|      Czech Republic|                      12|
|  FR|              France|                     111|
|  MY|            Malaysia|                      16|
|  TI|          Tajikistan|                      62|
|  AJ|          Azerbaijan|                      66|
|  BG|          Bangladesh|                      10|
|  BL|             Bolivia|                      36|
|  CA|              Canada|                    8910|
|  IN|               India|                    3807|
|  IZ|                Iraq|                       1|
|  JM|             Jamaica|                       3|
|  MX|              Mexico|                    5249|
|  MZ|          Mozambique|                      19|
|  NI|             Nigeria|                      10|
+----+--------------------+------------------------+
```

```
#Same way for state and save a copy of each table to output directory
num_station_state = station_new.groupBy(F.col('STATE')).count()
station_num_state = state.join(num_station_state
                                        .withColumnRenamed('count', 'STATION_NUM_STATE')
                                        .withColumnRenamed('STATE','CODE'),
                                        on = 'CODE',
                                        how = 'left')
# station_num_state.cache()
# station_num_state.show()
+----+-------------------+-----------------+
|CODE|         STATE_NAME|STATION_NUM_STATE|
+----+-------------------+-----------------+
|  IL|           ILLINOIS|             1889|
|  NJ|         NEW JERSEY|              739|
|  NT|NORTHWEST TERRITO...|              137|
|  PI|    PACIFIC ISLANDS|             null|
|  CA|         CALIFORNIA|             2879|
|  IN|            INDIANA|             1741|
|  OK|           OKLAHOMA|             1018|
|  WY|            WYOMING|             1211|
|  CT|        CONNECTICUT|              350|
|  MN|          MINNESOTA|             1557|
|  WV|      WEST VIRGINIA|              516|
|  MT|            MONTANA|             1240|
|  ND|       NORTH DAKOTA|              545|
|  NH|      NEW HAMPSHIRE|              431|
|  OH|               OHIO|             1179|
|  WI|          WISCONSIN|             1102|
|  AZ|            ARIZONA|             1534|
|  ID|              IDAHO|              794|
|  MB|           MANITOBA|              722|
|  SD|       SOUTH DAKOTA|             1035|
+----+-------------------+-----------------+
```

```
In [88]: ! hdfs dfs -ls /user/pwu17/outputs/ghcnd/station_num_each_country.csv
Found 17 items
-rw-r--r--   4 pwu17 pwu17          0 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/_SUCCESS
-rw-r--r--   4 pwu17 pwu17        207 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00000-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        269 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00001-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        181 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00002-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        312 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00003-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        239 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00004-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        205 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00005-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        284 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00006-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        228 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00007-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        253 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00008-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        249 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00009-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        158 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00010-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        208 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00011-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        217 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00012-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        298 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00013-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        399 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00014-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
-rw-r--r--   4 pwu17 pwu17        369 2022-04-26 16:06 /user/pwu17/outputs/ghcnd/station_num_each_country.csv/part-00015-66dd2d5d-0a57-4ba9-8dbf-d82b3babec74-c000.csv
```

```
In [92]: ! hdfs dfs -ls /user/pwu17/outputs/ghcnd/station_num_state.csv
Found 17 items
-rw-r--r--   4 pwu17 pwu17          0 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/_SUCCESS
-rw-r--r--   4 pwu17 pwu17         86 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00000-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         68 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00001-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         58 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00002-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         88 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00003-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         66 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00004-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         16 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00005-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17        129 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00006-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         86 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00007-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         33 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00008-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         81 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00009-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17        111 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00010-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17        110 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00011-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         77 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00012-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         74 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00013-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17         90 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00014-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
-rw-r--r--   4 pwu17 pwu17        191 2022-04-26 16:19 /user/pwu17/outputs/ghcnd/station_num_state.csv/part-00015-3774a431-969c-4ee2-a5ee-a410527a1d1e-c000.csv
```

Q1-2 Count the total number of stations and states in each country code and result with storing.

There are 93156 stations in the Northern Hemisphere only.

```
In [93]: station_new.filter(station_new.LATITUDE >= 0).count()
Out[93]: 93156
```

399 stations are in total in the territories of the United States around the world excluding the United States itself.

```
territory_US = (
                station_num_each_country
                .filter((F.col('COUNTRY_NAME')
                .contains('United States')) & (~F.col('COUNTRY_NAME')
                .startswith('United States'))))
#territory_US.cache()
#territory_US.show()
+----+--------------------+------------------------+
|CODE|        COUNTRY_NAME|STATION_NUM_EACH_COUNTRY|
+----+--------------------+------------------------+
|  CQ|Northern Mariana ...|                      11|
|  WQ|Wake Island [Unit...|                       1|
|  AQ|American Samoa [U...|                      21|
|  LQ|Palmyra Atoll [Un...|                       3|
|  GQ|Guam [United States]|                      21|
|  JQ|Johnston Atoll [U...|                       4|
|  MQ|Midway Islands [U...|                       2|
|  VQ|Virgin Islands [U...|                      54|
|  RQ|Puerto Rico [Unit...|                     222|
+----+--------------------+------------------------+
territory_US.select(F.sum('STATION_NUM_EACH_COUNTRY')).show()
+-----------------------------+
|sum(STATION_NUM_EACH_COUNTRY)|
+-----------------------------+
|                          339|
+-----------------------------+
```

Q2

We explore a spark function to compute the geographical distance between two stations using relative latitude and longitude. In the python function, the earth's radius is defined as 6371 km. Both latitude and longitude of two coordinates are put into the function. This computation uses the Haversine formula to calculate the great circle distance between two coordinates with the shortest distance over the earth's surface. The function is divided into four parts, including transforming to radians, calculating area, calculating the central angle and calculating distance.

Haversine formula (Chris Veness, w. ,2022):

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

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

$$d = R \cdot c$$

where *φ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!*

```
import math

def stations_distance(lat1,lon1,lat2,lon2):
    """point1: local -- tuple of float (lat,lon)
    and point2: loca2 ---tuple of float (lat,lon)
    """
    radius = 6371  #km
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2)**2 + math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) * math.sin(dlon / 2)**2

    circle = 2 * math.asin(math.sqrt(a))
    distance = radius * circle
    return distance
```

After defining as a python function, we need to convert the function to UDF in order to creating one custom function and operating it on columns in the data frame. The data type should be set columns as float type.

udf_distance = F.udf (stations_distance, DoubleType ())

Now, based on the different range of longitude, we utilise filter () and select () functions to take two subsets, and take these subsets to cross join them with themselves separately, as the Q2-1 shown.

```
subset = (station_new.filter(station_new.LONGITUDE < 25).limit(10).select(F.col('STATION_ID'),
        F.col('LATITUDE'),F.col('LONGITUDE')))
#subset.cache()
#subset.show()
+-----------+--------+---------+
| STATION_ID|LATITUDE|LONGITUDE|
+-----------+--------+---------+
|AGE00147718|   34.85|     5.72|
|AGM00060417|  36.383|    3.883|
|AGM00060421|  35.867|    7.117|
|AGM00060531|  35.017|    -1.45|
|AGM00060555|  33.068|    6.089|
|AO000066447| -15.833|    20.35|
|AQC00914594|-14.3333|-170.7667|
|AQW00061705|-14.3306|-170.7136|
|AR000087828|   -43.2|  -65.266|
|AR000087925| -51.617|  -69.283|
+-----------+--------+---------+

subset_new = (station_new.filter(station_new.LONGITUDE > 30 ).limit(10).select(F.col('STATION_ID').alias('STATION_ID_NEW'),
            F.col('LATITUDE').alias('LATITUDE_NEW'),
            F.col('LONGITUDE').alias('LONGITUDE_NEW')))
#subset_new.cache()
#subset_new.show()
+--------------+------------+-------------+
|STATION_ID_NEW|LATITUDE_NEW|LONGITUDE_NEW|
+--------------+------------+-------------+
|   AJ000037679|        41.1|         49.2|
|   AJ000037831|        40.4|         47.0|
|   AJ000037981|        38.9|         48.2|
|   AJ000037989|        38.5|         48.9|
|   AM000037719|        40.6|        45.35|
|   AM000037897|      39.533|       46.017|
|   ASN00001003|    -14.1331|     126.7158|
|   ASN00001020|      -14.09|     126.3867|
|   ASN00002031|    -17.0103|     128.4669|
|   ASN00002033|       -17.8|        128.3|
+--------------+------------+-------------+
```

Q2-1 limiting range of longitude to split the data into two subsets for testing.

Then, some repeated rows could be removed according with these STATION_IDs.

Finally, the udf_distance function is applied to the subset, which is after cross join to add one new column, 'Distance/km'. The code and result of the example are shown in the below snippet (test1 & 2). We input the latitude and longitude data into the official calculator latitude & longitude distance to get the result; the result is the same as the number from our function (Q2-2 shows). The validating website: Latitude/Longitude Distance Calculator (noaa.gov).

STATION_ID_A	LATITUDE_A	LONGITUDE_A	STATION_ID_A1	LATITUDE_A1	LONGITUDE_A1	Distance/km
AGE00147718	34.85	5.72	AGM00060417	36.383	3.883	237.9625309815391
AGE00147718	34.85	5.72	AGM00060421	35.867	7.117	169.8130914156512
AGE00147718	34.85	5.72	AGM00060531	35.017	-1.45	653.7369539465011
AGE00147718	34.85	5.72	AGM00060555	33.068	6.089	201.04990595869236
AGE00147718	34.85	5.72	AO000066447	-15.833	20.35	5843.749552413994
AGE00147718	34.85	5.72	AQC00914594	-14.3333	-170.7667	17706.72315560044
AGE00147718	34.85	5.72	AQW00061705	-14.3306	-170.7136	17705.608418388274
AGE00147718	34.85	5.72	AR000087828	-43.2	-65.266	11266.138895690467
AGE00147718	34.85	5.72	AR000087925	-51.617	-69.283	12056.386865502114
AGM00060417	36.383	3.883	AGE00147718	34.85	5.72	237.9625309815391
AGM00060417	36.383	3.883	AGM00060421	35.867	7.117	296.0612631656579
AGM00060417	36.383	3.883	AGM00060531	35.017	-1.45	504.8698926894565
AGM00060417	36.383	3.883	AGM00060555	33.068	6.089	420.1070853594153
AGM00060417	36.383	3.883	AO000066447	-15.833	20.35	6058.442382575106
AGM00060417	36.383	3.883	AQC00914594	-14.3333	-170.7667	17506.229431953736
AGM00060417	36.383	3.883	AQW00061705	-14.3306	-170.7136	17504.810594290204
AGM00060417	36.383	3.883	AR000087828	-43.2	-65.266	11271.99553876459
AGM00060417	36.383	3.883	AR000087925	-51.617	-69.283	12084.214472603207
AGM00060421	35.867	7.117	AGE00147718	34.85	5.72	169.8130914156512
AGM00060421	35.867	7.117	AGM00060417	36.383	3.883	296.0612631656579

Q2-2 example of test result

Applying the above function is to compute the pairwise distances between all stations in New Zealand. We firstly pick up the data about NZ station from new station data sets by filter () function in 'COUNTRY_CODE'. There are 15 NZ stations. The following step is same as the above process with station_nz join to themselves and create a new table (Q2-3). In this computing part, we can also think about alternative method using Cross Join () function which is similar to the above test1&2 method. We can save the data into our own path and use hdfs command to show. We can use the sort () function to easier find the closest distance between stations in New Zealand. The station in PARAPARAUMU AWS is closest to WELLINGTON AERO AWS station and the closest distance is about 50.53 km.

COUNTRY_CODE	ID	NAME	LAT	LON	ID_1	NAME_1	LAT_1	LON_1	Distance/km
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	714.1268050318199
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZM00093678	KAIKOURA	-42.417	173.7	224.98088249003268
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZ000937470	TARA HILLS	-44.517	169.9	218.3091932907015
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZ000939870	CHATHAM ISLANDS AWS	-43.95	-176.567	1015.2499467565266
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZM00093781	CHRISTCHURCH INTL	-43.489	172.532	152.25804856764034
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZ000939450	CAMPBELL ISLAND AWS	-52.55	169.167	1101.719032102152
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZM00093929	ENDERBY ISLAND AWS	-50.483	166.3	934.2477752798079
NZ	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZM00093439	WELLINGTON AERO AWS	-41.333	174.8	350.79606843225486
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	1796.3613283746295
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	1095.8224733430584
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZM00093678	KAIKOURA	-42.417	173.7	1645.5647898351037
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZ000937470	TARA HILLS	-44.517	169.9	2008.88592155291
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZ000939870	CHATHAM ISLANDS AWS	-43.95	-176.567	1638.937300349759
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZM00093781	CHRISTCHURCH INTL	-43.489	172.532	1796.5560036960667
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZ000939450	CAMPBELL ISLAND AWS	-52.55	169.167	2799.175956993722
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZM00093929	ENDERBY ISLAND AWS	-50.483	166.3	2705.4207241687313
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZM00093439	WELLINGTON AERO AWS	-41.333	174.8	1495.9413873304925
NZ	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	NZ000933090	NEW PLYMOUTH AWS	-39.017	174.183	1305.703712037678
NZ	NZ000093012	KAITAIA	-35.1	173.267	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	869.6235232640119
NZ	NZ000093012	KAITAIA	-35.1	173.267	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917	1053.5275724206938

Q2-3 Table of pairwise distance between all stations in NZ

```
In [164]: ! hdfs dfs -ls /user/pwu17/outputs/ghcnd/NZ_stations_distance.csv
Found 3 items
-rw-r--r--   4 pwu17 pwu17          0 2022-04-26 17:43 /user/pwu17/outputs/ghcnd/NZ_stations_distance.csv/_SUCCESS
-rw-r--r--   4 pwu17 pwu17          0 2022-04-26 17:43 /user/pwu17/outputs/ghcnd/NZ_stations_distance.csv/part-00000-452b913e-d2ee-4384-89ca-85384ea44f72-c000.csv
-rw-r--r--   4 pwu17 pwu17      11409 2022-04-26 17:43 /user/pwu17/outputs/ghcnd/NZ_stations_distance.csv/part-00008-452b913e-d2ee-4384-89ca-85384ea44f72-c000.csv
```

Q3

We can explore all of the daily climate summaries showing more details and know the efficiency of loading and applying transformations to daily part. use the following command to determine the default block size of HDFS.

hdfs getconf -confKey "dfs. blocksize", the default size is 134217728 bytes representing 128 MB.

For 2022 daily-data, we can use the "hdfs dfs -ls" to find the file size of 2022 csv.gz is 25985757 bytes which is apparently smaller than the default block size. "Hdfs fsck" command can work on the file and show more details and further determine.

```
View the blocks for the specific file 2022.csv.gz
$ hdfs dfs -ls /data/ghcnd/daily/2022.csv.gz  #25985757 bytes
$ hdfs getconf -confKey "dfs.blocksize" #134217728 bytes default block size
$ hdfs fsck /data/ghcnd/daily/2022.csv.gz -files -blocks
Showing:  /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
          Total blocks (validated):      1 (avg. block size 25985757 B)
```

For 2021 daily data, we use the same way as the above commands and get the below result. Because the total size of 2021 daily data is 146936025 bytes which are over the default block size, there are two blocks for validating. Two blocks are possible to present the spark's parallel feature. Interestingly, the input size is over the default size of 128M, the reason could be compressed file cannot be split by Spark.

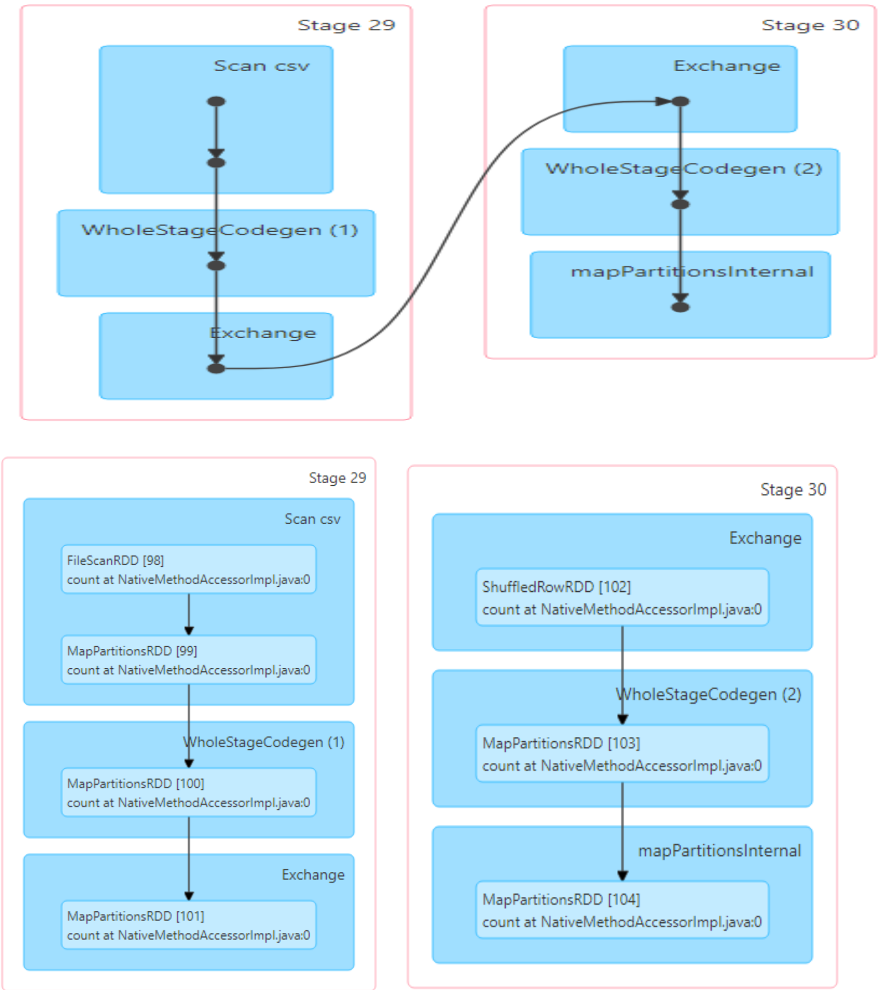
```
#What about the year 2021?

View the blocks for the specific file 2021.csv.gz
$ hdfs dfs -ls /data/ghcnd/daily/2021.csv.gz
$ hdfs fsck /data/ghcnd/daily/2021.csv.gz -files -blocks
Showing: /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
         Total blocks (validated):      2 (avg. block size 73468012 B)
```

We can check on an interface about the executor details by observing 'mathmadslinux2p:8080' and find more details about processing changes. By loading the 2021 CSV file, the CSV process includes 1 stage, 1 task, input size of 64 Kb, and the counting process has 2 stages, 2 tasks, input size of 140.1 Mb. During the counting procedure, 2 phases are created since the transformation is performed here. After the conversion operation is completed, the shuffle needs to start, because the data needs to be shuffled between different partitions. So, a stage is created and then another single stage of the transformation task will be created. Divided into tasks in internal stages, each stage is divided into 2 tasks because there are 2 partitions. Each partition runs a separate task. We run the 2022 file showing the same process and number of tasks. There are blocks in 2021

```
# Load and count the number of observations in 2021 and then separately in 2022.
daily_2021 = spark.read.csv("/data/ghcnd/daily/2021.csv.gz")
daily_2021.count() #34657282
daily_2022 = spark.read.csv("/data/ghcnd/daily/2022.csv.gz")
daily_2022.count() #5971307
```

Let's check the DAG visualization in WEB UI (as the below picture Q3-1). Vertices present the Data Frame or RDDs and edges represent an operation applied on RDD. There are two main stages, the first one contains scan csv, whole stage code generating with exchanging to next step including complete stage code generating and map partitions internal. Whole-stage code generation, as a query in Spark SQL, is to merge multiple physical operators into a single Java function, which is helpful to improve execution performance. From the display we can see Spark's optimization of pipelining operations, and each executor then operates on mapping on the same task to avoid association with the next stage after reading the partition. We open the DAG details and find that file scan and map partitions happen in the left stage and there are shuffle of rows and map partitions at the right stage.



Q3-1 DAG visual of process running daily_2021 and count.

If we want to know whether the number of tasks corresponds to the number of blocks, we should first know that each file contains many blocks. When Spark reads these files as input, it will parse according to the Input-Format corresponding to the specific data format. The blocks are merged into one input slice, called Input-Split. Note that Input-Split cannot span files. Concrete tasks will then be generated for these input shards. There is a one-to-one correspondence between Input-Split and Task. Then each of these specific tasks will be assigned to an Executor of a node on the cluster to execute. Each node can start one or more Executors. Each Executor consists of several cores, and each core of each Executor can only execute one task at a time. The result of each Task execution is to generate a partition of the target RDD. When CPU resources are sufficient, multiple threads within a process can be allocated to different CPU resources. Parallelism can ensure that two or more events are performed simultaneously without contention and waiting. Therefore, we cannot support the number of tasks executed in workers corresponding to the number of blocks in each input because it will be impacted by the input format and input split separately.

The number of tasks depends on the number of last RDD partitions. The number of tasks is determined by parallelism (number of partitions) and the cores which can be applied in Spark.

Loading the number of observations from 2014 to 2022 and running out its counting result, there are 284918108 observations, and we check the web user interface, which shows one job, two stages (1 task (partition) for the 2nd stage & 9 tasks (partitions) for 1st stage), ten tasks. Specifically, the stage is divided with shuffle operation as the boundary. As the below snippet shows, the first stage executes shuffle operations, namely: the process is to input 1386.8MB and then generate 531B of data to write the data to a hard disk. In the following stage, as an active operation, is to read the data reported in the last shuffle and goes back to the driver together so there is shuffle read size is the same as the shuffle write size.

Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1/1			531.0 B	
9/9	1386.8 MiB			531.0 B

Spark partitioning is one method of splitting data into multiple partitions to execute transformations on various partitions in parallel. The number of cores available for an executor decides the number of tasks in Spark that can run in parallel for the application. For example (cite from Hadoop in Real World), if you have five executors for your application with ten cores in each executor. It means 50 CPU cores are available in total and represents Spark can execute a maximum of 50 tasks in parallel. In this case, the gzip compressed file cannot be split into partitions (tasks) of more than 260 because there are 260 years total. With these different daily data sizes, the partitioning process could happen one partition per year for larger data size of years or one partition containing several smaller sizes of years. Note that, The concurrency of tasks executed equals the number of executors multiplying with the number of executor cores per task. We could get smaller data sizes of years across all of these partitions aggregated over multiple years. Spark can combine the smaller size of different years into a single larger partition. Based on this feature, Spark could also create 110 tasks.We can increase the number of tasks running in parallel by increasing the number of executors or the number of cores per executor.

Q4

We use the below code to show the all daily data in more details. There are 3000243596 rows in all daily (Q4-1 picture shows).

```
all_daily = (spark.read.format("com.databricks.spark.csv")
            .option("header","false")
            .option("inferSchema", "false")
            .schema(dailySchema)
            .load("hdfs:///data/ghcnd/daily/")
            )

all_daily.count() #3000243596
```

```
+-----------+----------+-------+------+-----+-----+-----+-------------------+------------+
|         ID|      DATE|ELEMENT| VALUE|MFLAG|QFLAG|SFLAG|           OBV_TIME|COUNTRY_CODE|
+-----------+----------+-------+------+-----+-----+-----+-------------------+------------+
|CA002303986|2010-01-01|   TMAX| 205.0| null|    G|    C|               null|          CA|
|CA002303986|2010-01-01|   TMIN|-300.0| null| null|    C|               null|          CA|
|CA002303986|2010-01-01|   PRCP|   4.0| null| null|    C|               null|          CA|
|CA002303986|2010-01-01|   SNOW|   4.0| null| null|    C|               null|          CA|
|CA002303986|2010-01-01|   SNWD|   0.0| null|    I|    C|               null|          CA|
|US1FLSL0019|2010-01-01|   PRCP|   0.0|    T| null|    N|               null|          US|
|ASN00037003|2010-01-01|   PRCP|   0.0| null| null|    a|               null|          AS|
|US1AZMR0019|2010-01-01|   PRCP|   0.0| null| null|    N|               null|          US|
|US1AZMR0019|2010-01-01|   SNOW|   0.0| null| null|    N|               null|          US|
|USC00178998|2010-01-01|   TMAX|   0.0| null| null|    0|1800-01-01 00:00:00|          US|
|USC00178998|2010-01-01|   TMIN| -56.0| null| null|    0|1800-01-01 00:00:00|          US|
|USC00178998|2010-01-01|   TOBS| -56.0| null| null|    0|1800-01-01 00:00:00|          US|
|USC00178998|2010-01-01|   PRCP|  43.0| null| null|    0|1970-01-01 07:00:00|          US|
|USC00178998|2010-01-01|   SNOW|  46.0| null| null|    0|               null|          US|
|USC00178998|2010-01-01|   SNWD| 102.0| null| null|    0|               null|          US|
|NOE00133566|2010-01-01|   TMAX|   2.0| null| null|    E|               null|          NO|
|NOE00133566|2010-01-01|   TMIN| -84.0| null| null|    E|               null|          NO|
|NOE00133566|2010-01-01|   PRCP|  85.0| null| null|    E|               null|          NO|
|NOE00133566|2010-01-01|   SNWD| 490.0| null| null|    E|               null|          NO|
|USC00242347|2010-01-01|   TMAX|  33.0| null| null|    0|1970-01-01 08:00:00|          US|
+-----------+----------+-------+------+-----+-----+-----+-------------------+------------+
```

Q4-1 all daily table

We filter out the ELEMENT and obtain the subset of observations including five core elements as the shown above inventory. There are 2565647147 observations for each of five core elements. After counting the total number of each element, we can find the PRCP contains more observations than other core elements.

```
all_daily_coreE = all_daily.filter(F.col('ELEMENT').isin(coreElements))
#How many observations are there for each of the five core elements? 2565647147
all_daily_coreE.count()
#Which element has the most observations?
all_daily_coreE.groupBy('ELEMENT').agg({'ELEMENT':'count'}).show()
+-------+--------------+
|ELEMENT|count(ELEMENT)|
+-------+--------------+
|   SNOW|     341985067|
|   SNWD|     289981374|
|   PRCP|    1043785667|
|   TMIN|     444271327|
|   TMAX|     445623712|
+-------+--------------+
```

Many stations collect TMIN and TMAX, but do not necessarily report both, due to data collection or coverage issues. 8808805 observations of TMIN don't contain a corresponding observation of TMAX, and there are 27650 different stations contributed to these observations.

We just pick all TMIN and TMAX observations for all New Zealand stations from daily data and get the below picture (Q4-2). We mainly use the filter function to take the NZ rows based on the 'COUNTRY_CODE' and 'ELEMENT' containing TMAX or TMIN. The final step is to save the result to the user's output directory. There are 472271 observations included there. According to the exit TMIN and TMAX data sets, ID, DATE and ELEMENT are grouped, and we use VALUE in the avg function to get the average which will be useful in plotting time series. Note that the year item can be taken as one subset and becomes a referring item in later plotting (Q4-3 pic shows). Eighty-three. The observations cover 83 years. We use the hdfs dfs -copyToLocal to copy the output from HDFS to the local home directory and find 472272 rows in the part files using the wc - l bash command.

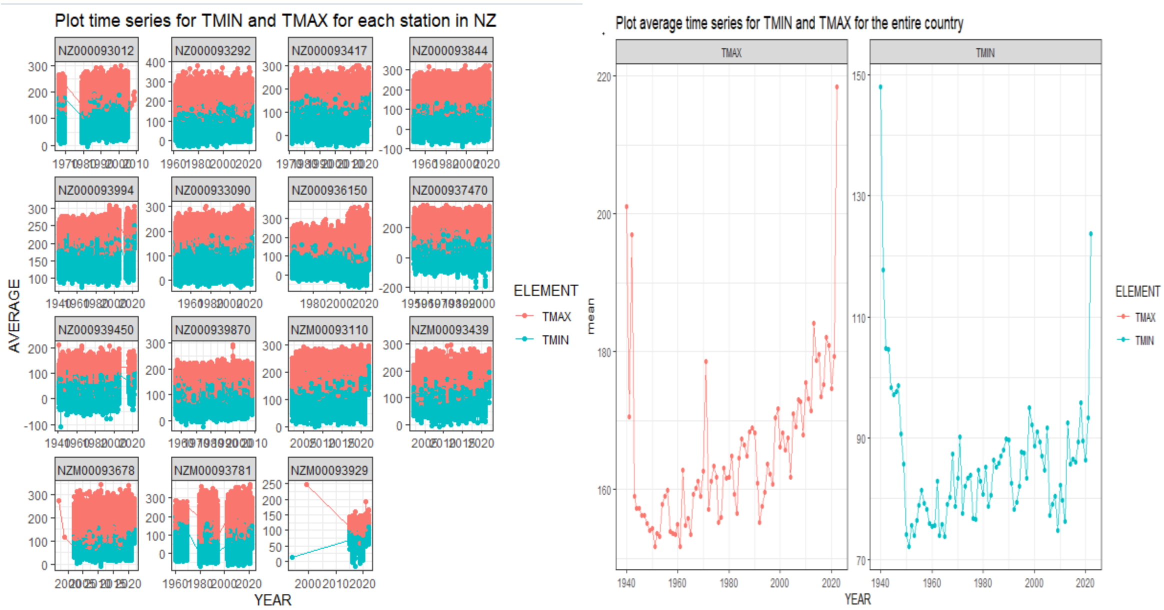
ID	DATE	ELEMENT	VALUE	MFLAG	QFLAG	SFLAG	OBV_TIME	COUNTRY_CODE
NZ000936150	2010-01-01	TMAX	324.0	null	null	S	null	NZ
NZM00093110	2010-01-01	TMAX	215.0	null	null	S	null	NZ
NZM00093110	2010-01-01	TMIN	153.0	null	null	S	null	NZ
NZM00093678	2010-01-01	TMAX	242.0	null	null	S	null	NZ
NZM00093678	2010-01-01	TMIN	94.0	null	null	S	null	NZ
NZ000093292	2010-01-01	TMAX	297.0	null	null	S	null	NZ
NZ000093292	2010-01-01	TMIN	74.0	null	null	S	null	NZ
NZM00093781	2010-01-01	TMAX	324.0	null	null	S	null	NZ
NZM00093439	2010-01-01	TMAX	204.0	null	null	S	null	NZ
NZM00093439	2010-01-01	TMIN	134.0	null	null	S	null	NZ
NZ000093844	2010-01-01	TMAX	232.0	null	null	S	null	NZ
NZ000093844	2010-01-01	TMIN	96.0	null	null	S	null	NZ
NZ000093417	2010-01-01	TMAX	180.0	null	null	S	null	NZ
NZ000093417	2010-01-01	TMIN	125.0	null	null	S	null	NZ
NZ000933090	2010-01-01	TMAX	197.0	null	null	S	null	NZ
NZ000933090	2010-01-01	TMIN	82.0	null	null	S	null	NZ
NZM00093110	2010-01-02	TMAX	241.0	null	null	S	null	NZ
NZM00093110	2010-01-02	TMIN	153.0	null	null	S	null	NZ
NZM00093678	2010-01-02	TMAX	289.0	null	null	S	null	NZ
NZ000093292	2010-01-02	TMAX	302.0	null	null	S	null	NZ

Q4-2 output of all observations of TMIN & TMAX for all stations in NZ

ID	DATE	ELEMENT	AVERAGE	YEAR
NZM00093678	2010-01-01	TMIN	94.0	2010
NZM00093781	2010-01-03	TMAX	294.0	2010
NZ000093844	2010-01-03	TMAX	180.0	2010
NZM00093781	2010-01-06	TMAX	277.0	2010
NZ000093844	2010-01-09	TMIN	64.0	2010
NZ000933090	2010-01-09	TMAX	190.0	2010
NZ000936150	2010-01-10	TMAX	234.0	2010
NZ000093292	2010-01-10	TMIN	71.0	2010
NZ000093844	2010-01-10	TMAX	140.0	2010
NZ000936150	2010-01-11	TMAX	191.0	2010
NZ000933090	2010-01-12	TMIN	74.0	2010
NZ000093292	2010-01-14	TMIN	105.0	2010
NZM00093439	2010-01-14	TMAX	229.0	2010
NZ000936150	2010-01-19	TMIN	137.0	2010
NZ000093292	2010-01-19	TMIN	133.0	2010
NZM00093781	2010-01-19	TMIN	137.0	2010
NZ000093417	2010-01-19	TMAX	230.0	2010
NZ000933090	2010-01-19	TMAX	220.0	2010
NZ000093844	2010-01-20	TMAX	242.0	2010
NZM00093781	2010-01-23	TMAX	152.0	2010

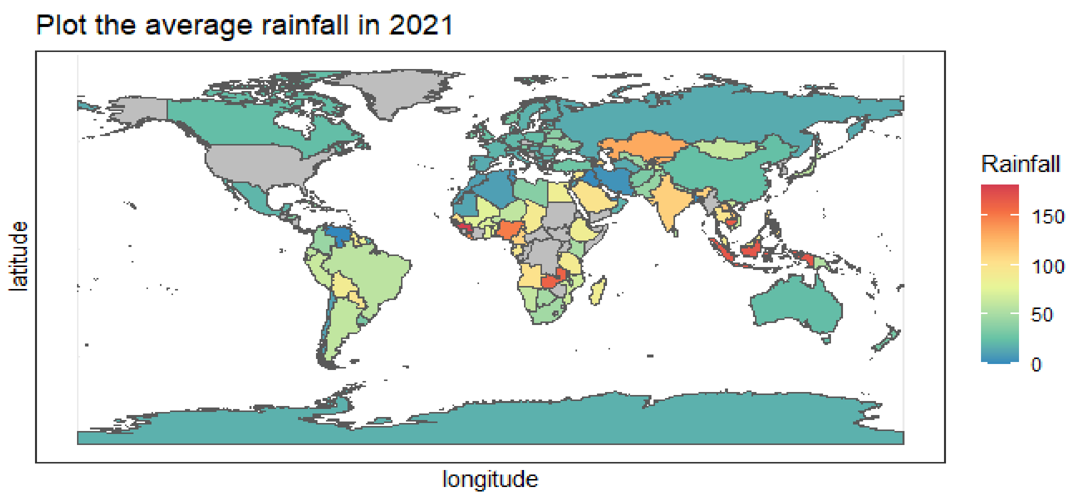
Q4-3 output of daily TMIN TMAX average value

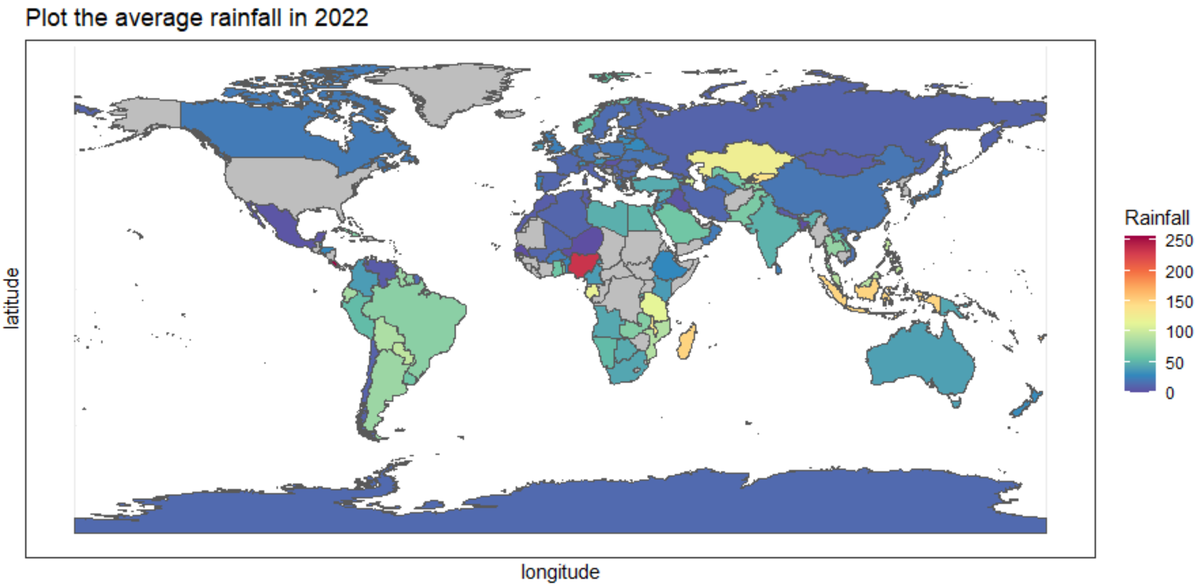
Loading file folders and using R programming language, we use the ggplot2 library and dplyr library to plot the time series and average time series for TMIN and TMAX on the same axes for the entire directory. We use the year as x and average as y to show the TMIN and TMAX time series distributing changes. There are two special outliers in the NZM00093929 for these two elements separately. The average values of TMIN and TMAX have the similar distributing changes and have apparent changes around 1940 and over 2022.



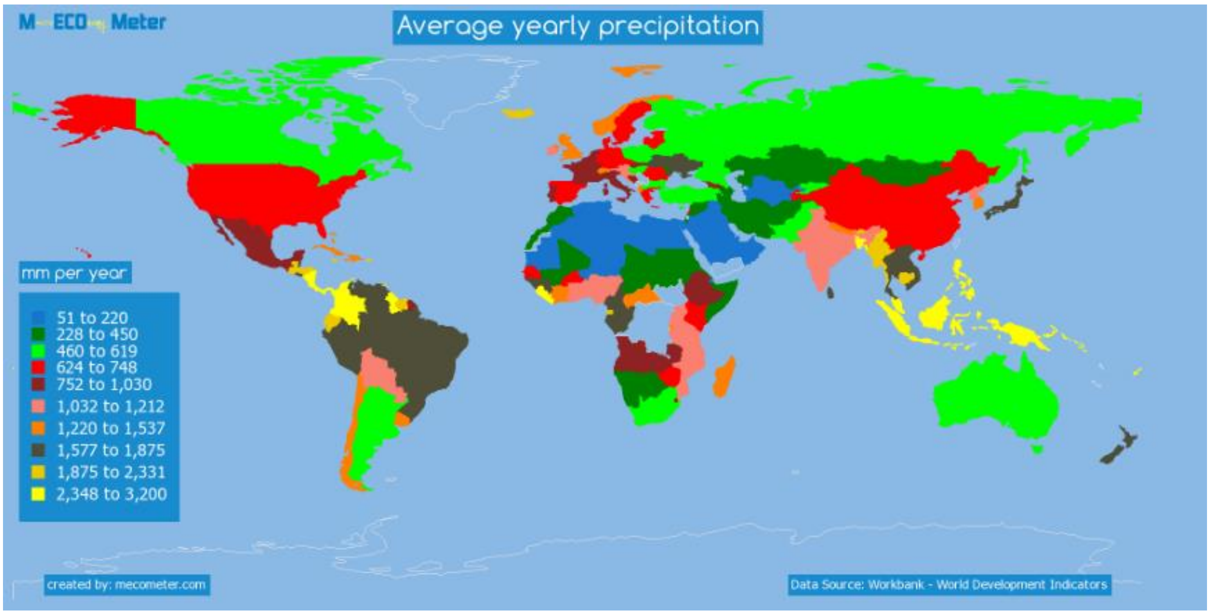
Q4-4 plot time series for TMIN & TMAX for the entire country

As the same method as the above shown, we call the specific packages about world maps using ggplot2 and use the choropleth to colour the map according to average rainfall (shown as Q4-5) (Andy, S, 2017) (Neuwirth, E,2022). Compared with the Q4-6 retrived from MECOMeter publishing average annual rainfall distribution map, rainfall distribution is uniform and, to some extent, the similar distribution verifies that our method of calling data and analysing is correct.





Q4-5 Rainfall distribution all over the world in 2021 & 2022



Q4-6 Average yearly precipitation - New Zealand (MECO METER)
http://mecometer.com/whats/new-zealand/average-yearly-precipitation/

As the final, the below picture shows my own home directory and all file folders in the path.

```
In [43]: ! hdfs dfs -ls /user/pwu17/outputs/ghcnd
Found 8 items
drwxr-xr-x   - pwu17 pwu17          0 2022-04-26 21:35 /user/pwu17/outputs/ghcnd/NZ_stations_distance.csv
drwxr-xr-x   - pwu17 pwu17          0 2022-04-27 17:52 /user/pwu17/outputs/ghcnd/daily_TMAX_TMIN_NZ
drwxr-xr-x   - pwu17 pwu17          0 2022-04-27 22:38 /user/pwu17/outputs/ghcnd/daily_TMAX_TMIN_NZ_VALUEAVE
drwxr-xr-x   - pwu17 pwu17          0 2022-04-27 19:00 /user/pwu17/outputs/ghcnd/daily_TMAX_TMIN_NZ_new
drwxr-xr-x   - pwu17 pwu17          0 2022-04-26 21:31 /user/pwu17/outputs/ghcnd/enriched-states.csv.gz
drwxr-xr-x   - pwu17 pwu17          0 2022-04-27 17:46 /user/pwu17/outputs/ghcnd/rainfall
drwxr-xr-x   - pwu17 pwu17          0 2022-04-27 14:43 /user/pwu17/outputs/ghcnd/station_num_each_country.csv
drwxr-xr-x   - pwu17 pwu17          0 2022-04-27 14:44 /user/pwu17/outputs/ghcnd/station_num_state.csv
```

Conclusion

GHCN is a comprehensive database containing daily climate summaries from surface weather stations worldwide. Spark is a way to distribute computations over large datasets. In this report, we mainly apply the concept of Spark to call and view the characteristics of each data and make related data visualizations. From this report, we can find the Spark calls the characteristics of data and uses the knowledge of pyspark to realize the extraction and calculation of specific data. It is a preliminary understanding of the application of data scalability. Hopefully, we are able to use more in-depth related discussions in the next study.

Reference

NCEI. (2021, December 15). Global Historical Climatology Network Daily (ghcnd). National Centres for Environmental Information (NCEI). Retrieved April 29, 2022, from https://www.ncei.noaa.gov/products/land-based-station/global-historical-climatologynetwork-daily

Menne, M.J., I. Durre, R.S. Vose, B.E. Gleason, and T.G. Houston, 2012: An overview of the Global Historical Climatology Network-Daily Database. *Journal of Atmospheric and Oceanic Technology*, 29, 897-910, doi.10.1175/JTECH-D-11-00103.1

PySpark documentation¶. PySpark Documentation - PySpark 3.2.1 documentation. (n.d.). Retrieved April 29, 2022, from https://spark.apache.org/docs/latest/api/python/

Pipis, G. (2021, September 5). Anti-joins with Pandas. Predictive Hacks. Retrieved April 29, 2022, from https://predictivehacks.com/?all-tips=anti-joins-with-pandas

Chris Veness, w. (2022). Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript. Retrieved 29 April 2022, from https://www.movable-type.co.uk/scripts/latlong.html

Hadoop in Real World. (2021,August 4),How does Spark decide the number of tasks and number of tasks to execute in parallel? (2022). Retrieved 29 April 2022, from https://www.hadoopinrealworld.com/how-does-spark-decide-the-number-of-tasks-and-number-of-tasks-to-execute-in-parallel/

Andy, S. (2017, March 21) The Comprehensive R Archive Network. (n.d.). Retrieved April 29, 2022, from https://cran.rproject.org/web/packages/rnaturalearth/README.html

Neuwirth, E. (2022, April 3). Colorbrewer palettes [R package RColorBrewer version 1.1-3]. The Comprehensive R Archive Network. Retrieved April 29, 2022, from https://cran.r-project.org/web/packages/RColorBrewer/index.html

Average yearly precipitation - New Zealand. MECOMeter. (n.d.). Retrieved April 29, 2022, from http://mecometer.com/whats/new-zealand/average-yearly-precipitation/