Assignment Report

Assignment No : 01
Scalable Data Science : DATA 420 19S1
University of Canterbury
College of Engineering

Author :Thiwanka Jayasiri
Student ID : 61623462
Email : gtj13@uclive.ac.nz

Global Historical Climatology Network (GHCN) Data Analysis using Spark.

The Global Historical Climatology Network (GHCN) is an integrated database of climate summaries from land surface stations across the globe that have been subjected to a common suite of quality assurance reviews. The data are obtained from more than 20 sources, each of which have been subjected to quality assurance reviews.

In this report we'll be mainly focusing on the following areas,

1) Processing
    a. Exploring data files
    b. Preparing schemas
    c. Perform JOIN operations to combine two or more data tables.
    d. Ideal format to save files
2) Analysis
    a. Summaries data
    b. Analysis of daily data along with few of the mergers
    c. Explore data in more details.

Processing

Firs we'll write simple Hadoop command code to check on how many files are in the given directory

Code:

```
hdfs dfs -ls ///data/ghcnd
```

Output

```
-rwxr-xr-x   4 hadoop hadoop       3670 2019-03-17 21:08 /data/ghcnd/countries
drwxr-xr-x   - hadoop hadoop          0 2019-03-17 21:27 /data/ghcnd/daily
-rwxr-xr-x   4 hadoop hadoop   27402154 2019-03-17 21:08 /data/ghcnd/inventory
-rwxr-xr-x   4 hadoop hadoop       1086 2019-03-17 21:07 /data/ghcnd/states
-rwxr-xr-x   4 hadoop hadoop    8914416 2019-03-17 21:08 /data/ghcnd/stations
```

Tree structure

This could be a silly work, but I wanted to try it on 'awk' as well since the $tree command was not working in my shell.

Code:

```
hdfs dfs -ls -R /data/ghcnd | awk '{print $8}' | sed -e 's/[^-][^\/]*\//--/g' -e 's/^/ /' -e 's/-/|/'
```

Output:

**I will not include the input over here, explanation at the bottom however the hand drawn diagram as follows,

```
 /data/shared/ghcnd/
 ├─ countries
 ├─ daily
 |    ├─ 1763.csv.gz
 |    ├─ 1764.csv.gz |
 |    ├─ 1765.csv.gz
 |    ├─ --------
 |    ├─ –2017.csv.gz
 ├─ inventory
 ├─ states
 ├─ stations
```

** As per the tree display the diagram 'Daily' got data from the year 1763 to 2017 and it's compressed as

'year.csv.gz' e.g. '1763.csv.gz'

To check on the file size changes in daily I've used the following command

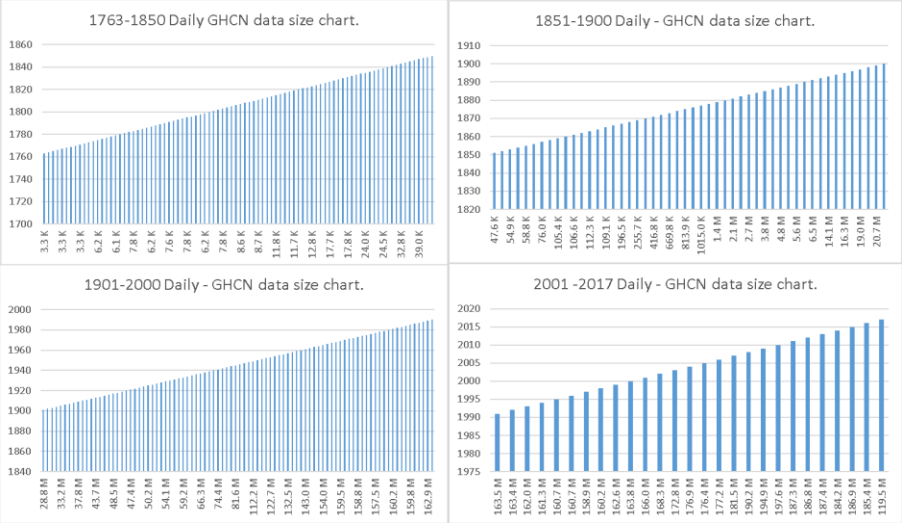
Code:

```
hdfs dfs -ls -R -h /data/ghcnd
```

I've provided the figure 1 to get an idea of the file sizes of the daily folder from year 1763 to 2017, as per the graph it shows that incremental growth of the file size, 3.3 kilobytes in year 1763, the size only reaches more than kilobytes in year 1814 only and thereafter to reach 100 kilobytes it takes till 1858, following to that the data size reaches to 1000 kilobyte mark in 1877 and gradually increase and passing 100 megabytes mark in 1948 and where as in 2017 it has recorded 119.5 megabytes.

```
Code :
hdfs dfs -count /data/ghcnd/daily
Output:
1           255         14354070619 /data/ghcnd/daily
```

Above code is to get number of files in the daily folder and it gives the result of 255



Checking the file size:

```
Code:
hdfs dfs -du -h ///data/ghcnd

Output
3.6 K   14.3 K   /data/ghcnd/countries
13.4 G  53.5 G   /data/ghcnd/daily
26.1 M  104.5 M  /data/ghcnd/inventory
1.1 K   4.2 K    /data/ghcnd/states
8.5 M   34.0 M   /data/ghcnd/stations
-------------------------------------------------------------------------------------

Code:
hdfs dfs -du -s -h ///data/ghcnd


Output:
13.4 G  53.6 G  /data/ghcnd
-------------------------------------------------------------------------------------

Code :
dfs dfs -du -s -h ///data/ghcnd/daily

Output :
13.4 G  53.5 G  /data/ghcnd/daily
```

As per the above code and output the file we have a total file size in ‘ghcnd’ directory, 53.6 gigabytes and out of which 53.5 gigabytes are from ‘daily’ folder only and that’s roughly about 99.813% of out of all the file sizes.

Setting up the schema.

For this we need to open up the spark shell, the command is ‘start_pyspark_shell’.

Defining schemas for each daily, stations, states, countries and inventory, there are several ways to do this sort of work, e.g. using a JSON, SQL. I'll be using the 'pyspark.sql' to perform in this session and will be using the data types define in the GHCN Daily README documentation.

Kindly refer to my code reference file data420-assignment.py and refer from line 47 to 90 for schemas I've listed schemas for Daily, Stations, Countries, States and Inventory.

Commented [GJ1]: Review the lines as I'll refactor the code at the final submission

Then to glance through the first 1000 rows of the 'Daily' schema I've written the code from line 98 to 123

Commented [GJ2]: Check on the line numbers again on the final code report.

```
from pyspark.sql.types import *

schema_dailys = StructType([
    StructField("ID", StringType(), True),
    StructField("DATE", StringType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("VALUE", StringType(), True),
    StructField("MEASUREMENT", StringType(), True),
    StructField("QUALITY FLAG", StringType(), True),
    StructField("SOURCE FLAG", StringType(), True),
    StructField("OBSERVATION TIME", StringType(), True),
])

dailys = (
    spark.read.format("com.databricks.spark.csv")
    .option("header", "false")
    .option("inferSchema", "false")
    .schema(schema_dailys)
    .load("hdfs:///data/ghcnd/daily/2017.csv.gz")
    .limit(1000)
)
dailys.show()
```

Observed that the field 'OBSERVATION TIME ' comes as a 'NULL' value in the top 20 rows but if load the first 1000 rows without adding the limit ()after load(), we could observe that some of the stations contain the time and most of them doesn't provide the observation time. However, the same stations ID must have taken an average value during the day and the type of collections focused on PRCP, SNOW, TMAX, TMIN, WESD, and WDFD. Looking first 1000 rows of the data, it's observable that the same IDs (unique station IDs) has picked several elements on a given day.

Let's look at the first 1000 rows of the daily 2017 files to check out the what the data contains on to perform this we can put a limit function soon after the read of the schema or else view the whole set we could make 1000 inside the show() function. Code snippet to view of these rows are as follows,

```
+-----------+--------+-------+-----+-----------+------------+-----------+----------------+
|         ID|    DATE|ELEMENT|VALUE|MEASUREMENT|QUALITY FLAG|SOURCE FLAG|OBSERVATION TIME|
+-----------+--------+-------+-----+-----------+------------+-----------+----------------+
|CA1MB000296|20170101|   PRCP|    0|       null|        null|          N|            null|
|US1NCBC0113|20170101|   PRCP|    5|       null|        null|          N|            null|
|ASN00015643|20170101|   TMAX|  274|       null|        null|          a|            null|
|ASN00015643|20170101|   TMIN|  218|       null|        null|          a|            null|
|ASN00015643|20170101|   PRCP|    2|       null|        null|          a|            null|
|US1MTMH0019|20170101|   PRCP|   43|       null|        null|          N|            null|
|US1MTMH0019|20170101|   SNOW|   28|       null|        null|          N|            null|
|US1MTMH0019|20170101|   SNWD|  178|       null|        null|          N|            null|
|ASN00085296|20170101|   TMAX|  217|       null|        null|          a|            null|
|ASN00085296|20170101|   TMIN|  127|       null|        null|          a|            null|
|ASN00085296|20170101|   PRCP|    0|       null|        null|          a|            null|
|US1MNWR0029|20170101|   PRCP|    0|       null|        null|          N|            null|
|ASN00085280|20170101|   TMAX|  215|       null|        null|          a|            null|
|ASN00085280|20170101|   TMIN|  156|       null|        null|          a|            null|
|ASN00085280|20170101|   PRCP|    0|       null|        null|          a|            null|
|ASN00040209|20170101|   TMAX|  293|       null|        null|          a|            null|
|ASN00040209|20170101|   TMIN|  250|       null|        null|          a|            null|
|ASN00040209|20170101|   PRCP|    0|       null|        null|          a|            null|
|US1MNHN0184|20170101|   PRCP|    0|       null|        null|          N|            null|
|US1MNHN0184|20170101|   SNOW|    0|       null|        null|          N|            null|
+-----------+--------+-------+-----+-----------+------------+-----------+----------------+
only showing top 20 rows
```

Schema for the defined schemes are as follows,

```
from pyspark.sql.types import *

schema_Daily = StructType([
    StructField("ID", StringType(), True),
    StructField("DATE", StringType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("VALUE", DoubleType(), True),
    StructField("MEASUREMENT", StringType(), True),
    StructField("QUALITY FLAG", StringType(), True),
    StructField("SOURCE FLAG", StringType(), True),
    StructField("OBSERVATION TIME", StringType(), True),
])

schema_Stations = StructType([
    StructField("ID", StringType(), True),
    StructField("LATITUDE", DoubleType(), True),
    StructField("LONGITUDE", DoubleType(), True),
    StructField("ELEVATION", DoubleType(), True),
    StructField("STATE", StringType(), True),
    StructField("NAME", StringType(), True),
    StructField("GSN FLAG", StringType(), True),
    StructField("HCN/CRN FLAG", StringType(), True),
    StructField("WMO ID", StringType(), True),
])

schema_Countries = StructType([
    StructField("CODE", StringType(), True),
    StructField("NAME", StringType(), True),
])

schema_States = StructType([
    StructField("CODE", StringType(), True),
    StructField("NAME", StringType(), True),
])

schema_Inventory = StructType([
    StructField("ID", StringType(), True),
    StructField("LATITUDE", DoubleType(), True),
    StructField("LONGITUDE", DoubleType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("FIRSTYEAR", IntegerType(), True),
    StructField("LASTYEAR", IntegerType(), True),
])
```

I've not included the .show () or count () function on above just defining the schema for later analysis work.

After setting up the schema for the all selected data tabs the summary of the tables are as follows, we can get the count by adding. count() function to the schema. e.g. stations.count ().

Let's assume that we needed to get the specifically the null value of the WMO_ID in the stations, using the following two lines of code

```
# count of the rows
stations.count()


#filter WMO ID and get the null values count

stations.filter(stations['WMO ID'] == '').count()
```

However prior to this we have to set the schema since if we are to use the same schema as mentioned above in code snippet (2) we gets the null values printed but can't count specifically as it doesn't remove the white space leading and trailing hence we require to use spark read format = 'text' and then use trim () function in to do whitespace data munging in spark. Refer the code line from 127 to 147 in my code file for this operation. I'll use the same read format as 'text' and 'trim ()' function for the rest of the tables for further analysis.

We could get the results as that stations schema contain **103,656** rows and out which the **'WMO ID'** holds null value for **95,595** rows. Following to that we can perform a similar operation for rest of the schemas, and we can get the following results on the row counts of each Meta data tables,

- ✓ Countries contain 218 rows.
- ✓ States contain 74 rows.
- ✓ Inventory has about 595,699 rows.

To save the output which being take upon performing tasks we need to create a different folder (output) and this could be easily achieve using the below command in the Hadoop environment

```
# creator new folder
hadoop fs -mkdir /user/gtj13/outputs


#check whether it's been built

hadoop fs -ls /user/gtj13
```

Let's first extract two character country code from each station code in stations and store the output in a new column using withColumn command.

```
country_code = F.substring(F.col('ID'), 1, 2)
stations_a = stations.withColumn('CODE', country_code)
stations_a.show()
```

Output as follows,

```
+-----------+--------+---------+---------+-----+--------------------+--------+------------+------+----+
|         ID|LATITUDE|LONGITUDE|ELEVATION|STATE|                NAME|GSN FLAG|HCN/CRN FLAG|WMO ID|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|
+-----------+--------+---------+---------+-----+--------------------+--------+------------+------+----+
```

Upon lets perform a LEFT join using the two tables countries and stations, as we seeing previous Meta data tables some column names are the same hence when joining tables we should not have same name on the columns hence performing the joint operation first we need to get the name change done in countries in this case I've used F.col to select the column and then alias () to change it to a different name prior the join , then I took stations_b as the new table after the joint perform , following code snippet would do the job efficiently.

```
countries = countries.select(
    F.col('CODE'),
    F.col('NAME').alias('COUNTRY_NAME')
    )

stations_b = stations_a.join(countries,"CODE","left")
stations_b.show()
```

Results are as follows,

```
+----+-----------+--------+---------+---------+-----+--------------------+--------+------------+------+--------------------+
|CODE|         ID|LATITUDE|LONGITUDE|ELEVATION|STATE|                NAME|GSN FLAG|HCN/CRN FLAG|WMO ID|        COUNTRY_NAME|
+----+-----------+--------+---------+---------+-----+--------------------+--------+------------+------+--------------------+
|  AC|ACW00011604| 17.1167| -61.7833|     10.1|     |ST JOHNS COOLIDGE...|        |            |      | Antigua and Barbuda|
|  AC|ACW00011647| 17.1333| -61.7833|     19.2|     |            ST JOHNS|        |            |      | Antigua and Barbuda|
|  AE|AE000041196|  25.333|   55.517|     34.0|     | SHARJAH INTER. AIRP|     GSN|            | 41196|United Arab Emirates|
|  AE|AEM00041194|  25.255|   55.364|     10.4|     |        DUBAI INTL|        |            | 41194|United Arab Emirates|
|  AE|AEM00041217|  24.433|   54.651|     26.8|     |     ABU DHABI INTL|        |            | 41217|United Arab Emirates|
|  AE|AEM00041218|  24.262|   55.609|    264.9|     |      AL AIN INTL|        |            | 41218|United Arab Emirates|
|  AF|AF000040930|  35.317|   69.017|   3366.0|     |       NORTH-SALANG|     GSN|            | 40930|         Afghanistan|
|  AF|AFM00040938|   34.21|   62.228|    977.2|     |            HERAT|        |            | 40938|         Afghanistan|
|  AF|AFM00040948|  34.566|   69.212|   1791.3|     |       KABUL INTL|        |            | 40948|         Afghanistan|
|  AF|AFM00040990|    31.5|    65.85|   1010.0|     |    KANDAHAR AIRPORT|        |            | 40990|         Afghanistan|
|  AG|AG000060390| 36.7167|     3.25|     24.0|     |  ALGER-DAR EL BEIDA|     GSN|            | 60390|             Algeria|
|  AG|AG000060590| 30.5667|   2.8667|    397.0|     |       EL-GOLEA|     GSN|            | 60590|             Algeria|
|  AG|AG000060611|   28.05|   9.6331|    561.0|     |      IN-AMENAS|     GSN|            | 60611|             Algeria|
|  AG|AG000060680|    22.8|   5.4331|   1362.0|     |     TAMANRASSET|     GSN|            | 60680|             Algeria|
|  AG|AGE00135039| 35.7297|     0.65|     50.0|     |ORAN-HOPITAL MILI...|        |            |      |             Algeria|
|  AG|AGE00147704|   36.97|     7.79|    161.0|     | ANNABA-CAP DE GARDE|        |            |      |             Algeria|
|  AG|AGE00147705|   36.78|     3.07|     59.0|     |ALGIERS-VILLE/UNI...|        |            |      |             Algeria|
|  AG|AGE00147706|    36.8|     3.03|    344.0|     |   ALGIERS-BOUZAREAH|        |            |      |             Algeria|
|  AG|AGE00147707|    36.8|     3.04|     38.0|     |  ALGIERS-CAP CAXINE|        |            |      |             Algeria|
|  AG|AGE00147708|   36.72|     4.05|    222.0|     |          TIZI OUZOU|        |            | 60395|             Algeria|
+----+-----------+--------+---------+---------+-----+--------------------+--------+------------+------+--------------------+
only showing top 20 rows
```

We follow the same steps we used in the stations and countries join to perform the states and stations but here I've demonstrated slightly different code structure but it perform in the same way.

```
stations_c = (
    stations_b
    .join(
        states
        .select(
             F.col('CODE'),
             F.col('NAME').alias('STATE_NAME')
        ),
        on = 'CODE',how = 'left')
    )

stations_c.show()
```

Results of the new table are as follows,

```
|CODE|         ID|LATITUDE|LONGITUDE|ELEVATION|STATE|                NAME|GSN FLAG|HCN/CRN FLAG|WMO ID|        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|
```

Looking at the earlier daily Meta table we understand that the 'daily' has elements such 'PRCP', 'SNOW', 'TMAX' and 'TMIN' also in the inventory table shown below we have Element column which has the same Elements set and to answer few of the exploratory questions it's better to do following operation as whole and try and find some interesting facts about the data by grouping and aggregating. The code snippet I used as follows,

First, we observe the inventory table and understand the set of columns we must perform some task

```
inventory.show()
+-----------+--------+---------+-------+---------+--------+
|         ID|LATITUDE|LONGITUDE|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|
|ACW00011604| 17.1167| -61.7833|   PGTM|     1949|    1949|
|ACW00011604| 17.1167| -61.7833|   WDFG|     1949|    1949|
|ACW00011604| 17.1167| -61.7833|   WSFG|     1949|    1949|
|ACW00011604| 17.1167| -61.7833|   WT03|     1949|    1949|
|ACW00011604| 17.1167| -61.7833|   WT08|     1949|    1949|
|ACW00011604| 17.1167| -61.7833|   WT16|     1949|    1949|
|ACW00011647| 17.1333| -61.7833|   TMAX|     1961|    1961|
|ACW00011647| 17.1333| -61.7833|   TMIN|     1961|    1961|
|ACW00011647| 17.1333| -61.7833|   PRCP|     1957|    1970|
|ACW00011647| 17.1333| -61.7833|   SNOW|     1957|    1970|
|ACW00011647| 17.1333| -61.7833|   SNWD|     1957|    1970|
|ACW00011647| 17.1333| -61.7833|   WT03|     1961|    1961|
|ACW00011647| 17.1333| -61.7833|   WT16|     1961|    1966|
|AE000041196|  25.333|   55.517|   TMAX|     1944|    2017|
|AE000041196|  25.333|   55.517|   TMIN|     1944|    2017|
+-----------+--------+---------+-------+---------+--------+
only showing top 20 rows
```

Secondly we can group the inventory by its ID and then aggregate it by its FIRST and LAST years, remember to change the column name of the new table, we can make countDistrinct() function to count number of elements and change the column name in the new table , then as another step we could filter out specific elements and in this case I've selected all four elements listed above , group by ID and then counted the number of elements captured by each ID and done a LEFT join , since I need to check on element PRCP only I took another column to the left and listed It's distinct count following code snippet will perform the job,

```
new_inv = (
    inventory
    .groupBy('ID')
    .agg(
        F.min('FIRSTYEAR').alias('MINIMUM_YEAR'),
        F.max('LASTYEAR').alias('MAXIMUM_YEAR'),
        F.countDistinct('ELEMENT').alias('NUM_ELEMENTS')
        )
    .join(
        inventory
        .filter((inventory.ELEMENT == 'PRCP') | (inventory.ELEMENT == 'SNOW')|
(inventory.ELEMENT == 'SNWD')| (inventory.ELEMENT == 'TMAX')| (inventory.ELEMENT == 'TMIN'))
        .groupBy('ID')
        .agg(
            F.countDistinct('ELEMENT').alias('NUM_CORE_ELEMENTS')
            ), 'ID', 'left'
        )
    .join(
        inventory
        .filter(inventory.ELEMENT == 'PRCP')
        .groupBy('ID')
        .agg(
            F.countDistinct('ELEMENT').alias('PRCP')
            ), 'ID', 'left'
        )

    )
new_inv = new_inv.withColumn('NUM_OTHER_ELEMENTS', F.col('NUM_ELEMENTS') -
F.col('NUM_CORE_ELEMENTS'))
new_inv.show()

stations_inventory_5 = new_inv.filter(F.col('NUM_CORE_ELEMENTS') == 5)
stations_inventory_5.count()

stations_inventory_pre = new_inv.filter((F.col('NUM_ELEMENTS') == 1) & (F.col('PRCP') == 1))
stations_inventory_pre.count()
```

Output

```
+-----------+------------+------------+------------+----------------+----+-----------------+
|         ID|MINIMUM_YEAR|MAXIMUM_YEAR|NUM_ELEMENTS|NUM_CORE_ELEMENTS|PRCP|NUM_OTHER_ELEMENTS|
+-----------+------------+------------+------------+----------------+----+-----------------+
|ACW00011647|        1957|        1970|           7|               5|   1|                2|
|AEM00041217|        1983|        2017|           4|               3|   1|                1|
|AGO00060590|        1892|        2017|           4|               3|   1|                1|
|AGE00147706|        1893|        1920|           3|               3|   1|                0|
|AGE00147708|        1879|        2017|           5|               4|   1|                1|
|AGE00147709|        1879|        1938|           3|               3|   1|                0|
|AGE00147710|        1909|        2009|           4|               3|   1|                1|
|AGE00147711|        1880|        1938|           3|               3|   1|                0|
|AGE00147714|        1896|        1938|           3|               3|   1|                0|
|AGE00147719|        1888|        2017|           4|               3|   1|                1|
|AGM00060351|        1981|        2017|           4|               3|   1|                1|
|AGM00060353|        1996|        2017|           4|               3|   1|                1|
|AGM00060360|        1945|        2017|           4|               3|   1|                1|
|AGM00060387|        1995|        2004|           4|               3|   1|                1|
|AGM00060445|        1957|        2017|           5|               4|   1|                1|
|AGM00060452|        1985|        2017|           4|               3|   1|                1|
|AGM00060467|        1981|        2017|           4|               3|   1|                1|
|AGM00060468|        1973|        2017|           5|               4|   1|                1|
|AGM00060507|        1943|        2017|           5|               4|   1|                1|
|AGM00060511|        1983|        2017|           5|               4|   1|                1|
+-----------+------------+------------+------------+----------------+----+-----------------+
only showing top 20 rows
```

Overall, we have 103,630 records

Here we have the first year as 1957 with the Number of elements recorded as 7 and since then the all 7 elements were recorded in 1970 that's by the station ID : ACW00011647 . Number of Core elements that got detected is 5 (the ones that I've defined in the schema).

The recent last year as 2017. As per the above table it's certain that the maximum number of elements were detected in 1970.

Below is the count that satisfies the maximum element number of 5 and the final count turns of to be 20,224 for the number of core elements.

```
In [13]: stations_inventory_5 = new_inv.filter(F.col('NUM_CORE_ELEMENTS') == 5)
    ...: stations_inventory_5.count()
    ...:
Out[13]: 20224
```

We get the number of other elements count can be taken as follows,

```
stations_inventory_other = new_inv.filter((F.col('NUM_OTHER_ELEMENTS') >= 1 ))
stations_inventory_other.show()
stations_inventory_other.count()

Out[20]: 67506
```

Final count is 67,506 with the total number of other elements.

We could also determine the set of elements that each station has collected and store this output as a new column using pyspark.sql.functions.collect set but it will be more efficient to first filter inventory by element type using the element column and then to join against that output as necessary. We can save in the output folder we created earlier by defining the path. I've used the file format as CSV in this case.

```
stations_inventory = (
    stations_c
    .join(
        new_inv,on='ID', how='left'))

stations_inventory.show()
stations_inventory.write.csv("hdfs:///user/gtj13/outputs/ghcnd/stations_inventory")

schema_newStations = StructType([
    StructField("ID", StringType(), True),
    StructField("CODE", StringType(), True),
    StructField("LATITUDE", DoubleType(), True),
```

```
    StructField("LONGITUDE", DoubleType(), True),
    StructField("ELEVATION", DoubleType(), True),
    StructField("STATE", StringType(), True),
    StructField("NAME", StringType(), True),
    StructField("GSN FLAG", StringType(), True),
    StructField("HCN/CRN FLAG", StringType(), True),
    StructField("WMO ID", StringType(), True),
    StructField("COUNTRY_NAME", StringType(), True),
    StructField("STATE_NAME", StringType(), True),
    StructField("MINIMUM_YEAR", IntegerType(), True),
    StructField("MAXIMUM_YEAR", IntegerType(), True),
    StructField("NUM_ELEMENTS", IntegerType(), True),
    StructField("NUM_CORE_ELEMENTS", IntegerType(), True),
    StructField("PRCP", IntegerType(), True),
    StructField("NUM_OTHER_ELEMENTS", IntegerType(), True),
])

stations_inventory = (
    spark.read.format("com.databricks.spark.csv")
    .option("header", "false")
    .option("inferSchema", "false")
    .schema(schema_newStations)
    .load("hdfs:///user/gtj13/outputs/ghcnd/stations_inventory")
)
```

Once the above schema being performed, we could perform small code snippet to find out where there are any subsets in daily which consist the value of 0 in the stations.

```
daily_stations = dailys.join(stations_inventory, dailys.ID == stations_inventory.ID,
'left_outer')
# count the number of stations which are in daily but not in stations
dailys.select('ID').subtract(stations_inventory.select('ID')).count()
```

There are 0 subsets that found which are not stations at all.

Left join would more expensive and I've performed a left_outer join in this case.

In the side note why selected csv without using other codec methods,

Benefits of selecting an appropriate file format,

1. Faster read times.
2. Faster write times.
3. Splittable files (so we don't need to read the whole file, just a part of it).
4. Schema evolution support (allowing us to change the fields in a dataset).
5. Advanced compression support (compress the files with a compression codec without sacrificing these features).

Well, if we consider csv.gz

It's recommended when achieving the highest level of compression (save the disk space) I could have used that but looking at the size of the data I decided go ahead with the csv format in the above case, however I would like to highlight following reasons as well when considering the compression. It's also good for cold data which doesn't access frequently, as I intend to use the saved file more frequently at this stage (hot data) I would like to stick to csv.

Reasons not to compress,

1. Compressed data is not splittable. Must be noted that many modern formats are built with block level compression to enable splitting and other partial processing of the files. b) Data is created in the cluster and compression takes significant time. Must be noted that compression usually much more CPU intensive then decompression.

2. Data has little redundancy and compression gives little gain.

Also, there are other methods in compressing files which would take certain amount of configuration time and also some comes with a cost of some speed, disk space and etc.

I think parquet would have been great choice , since at the challenge part I could have used it without spending more time, at the time of writing this bit I already saved the files on csv format and looking at the challenge part as the final step, I realized that the it's hard to access the particular column when reaching out and work on with csv and in parquet the data is stored in binary data in column oriented way , where the values of each column are organized so that they are adjacent , enabling better compression. It's especially good for queries which read particular columns from a "wide" (many columns) table since only needed columns read and I/O is minimized.

We can use the following code snippet to load the data in temporary view,

```
#write the file


df.write.parquet("hdfs:///user/gtj13/outputs/ghcnd/stations_inventory")


## loading the file
USING org.apache.spark.sql.parquet


hdfs:///user/gtj13/outputs/ghcnd/stations_inventory


OPTIONS (
  path " hdfs:///user/gtj13/outputs/ghcnd/stations_inventory.parquet"
)
```

Analysis

Exploratory analysis to get some basic understanding followed by the code snippets.

```
stations_inventory.select('ID').distinct().count()
#Out[4]: How many stations - 103656

stations_inventory.filter(stations_inventory.MAXIMUM_YEAR >=
2017).select('ID').distinct().count()
#Out[7]: How many stations have been active in 2017 - 37546

stations_inventory.filter(col('GSN FLAG') == 'GSN').count()
#Out[26]: GSN flag count 991

stations_inventory.filter(col('HCN/CRN FLAG') == 'HCN').count()
#Out[27]: HCN flag count 1218

stations_inventory.filter(col('HCN/CRN FLAG') == 'CRN').count()
#Out[28]: CRN flag count 230

stations_inventory.filter((col('HCN/CRN FLAG') != '')& (col('GSN FLAG') !=
'')).count()
#Out[7]: stations that are in more than one is 14
```

In this context of the this study proposed method of withColumnRenamed () for column name and there are other methods too, select () and also data.toDF(). I've used toDF() in some of the code snippets. I tried using the withColumnRenamed method in the following format just to test out the column changes.

First of all it's prudent to look at the first 20 columns of countries to do that we can perform countries.show() command we get the following countries table.(See appendix) Then when we perform the count_country

class , this is mainly to store the data frame that we are going to build to see how many stations are assigned in to one country. So we group it by country name and aggregate it by ID and then take a count.

```
count_country = stations_inventory.groupby('COUNTRY_NAME').agg({'ID':'count'})
```

Upon storing the results, we can use the withColumnRenamed ('Old Column name', 'New Column Name') to change the names accordingly.

```
count_country = (count_country
                        .withColumnRenamed('COUNTRY_NAME','NAME')
                        .withColumnRenamed('count(ID)' , 'number_of_stations'))
```

Then perform the following code snippet to do the left join.

```
countries = (
    countries
    .join(
        count_country
        .select(
            F.col('NAME'),
            F.col('number_of_stations'),
        ),

        on='NAME', how='left')
    )

countries.show()
```

On a side note we can perform the same inside the countries operation, but however when using withColumnRenamed() operation inside the operation tend to miss out the map on = NAME, hence I've performed it out as two operations.

Output as follows,

```
In [49]: countries.show()
+--------------------+----+------------------+
|                NAME|CODE|number_of_stations|
+--------------------+----+------------------+
| Antigua and Barbuda|  AC|                 2|
|United Arab Emirates|  AE|                 4|
|         Afghanistan|  AF|                 4|
|             Algeria|  AG|                87|
|          Azerbaijan|  AJ|                66|
|             Albania|  AL|                 3|
|             Armenia|  AM|                53|
|              Angola|  AO|                 6|
|American Samoa [U...|  AQ|                21|
|           Argentina|  AR|               101|
|           Australia|  AS|             17088|
|             Austria|  AU|                13|
|          Antarctica|  AY|               102|
|             Bahrain|  BA|                 1|
|            Barbados|  BB|                 1|
|            Botswana|  BC|                21|
|Bermuda [United K...|  BD|                 2|
|             Belgium|  BE|                 1|
|        Bahamas, The|  BF|                 6|
|          Bangladesh|  BG|                10|
+--------------------+----+------------------+
only showing top 20 rows
```

We can make the same changes accordingly on the stations and the output as follows and in this case, I've selected the select () and alias ()

```
count_states = stations_inventory.groupby('STATE').agg({'ID':'count'})
states = (
    states
    .join(
        count_states
        .select(
            F.col('STATE').alias('CODE'),
            F.col('count(ID)').alias('number_of_stations')
```

```
        ),
        on='CODE', how='left')
    )
states.show()
```

Output

```
+----+--------------------+-----------------+------------------+
|CODE|                NAME|number_of_stations|number_of_stations|
+----+--------------------+-----------------+------------------+
|  AB|             ALBERTA|             1404|              1404|
|  AK|              ALASKA|              982|               982|
|  AL|             ALABAMA|              926|               926|
|  AR|            ARKANSAS|              830|               830|
|  AS|      AMERICAN SAMOA|               21|                21|
|  AZ|             ARIZONA|             1327|              1327|
|  BC|    BRITISH COLUMBIA|             1668|              1668|
|  CA|          CALIFORNIA|             2663|              2663|
|  CO|            COLORADO|             3854|              3854|
|  CT|         CONNECTICUT|              233|               233|
|  DC|DISTRICT OF COLUMBIA|               14|                14|
|  DE|            DELAWARE|              102|               102|
|  FL|             FLORIDA|             1460|              1460|
|  FM|          MICRONESIA|               38|                38|
|  GA|             GEORGIA|             1120|              1120|
|  GU|                GUAM|               20|                20|
|  HI|              HAWAII|              720|               720|
|  IA|                IOWA|              748|               748|
|  ID|               IDAHO|              740|               740|
|  IL|            ILLINOIS|             1630|              1630|
+----+--------------------+-----------------+------------------+
only showing top 20 rows
```

```
count_states = stations_inventory.groupby('STATE').agg({'ID':'count'})
states = (
    states
    .join(
        count_states
        .select(
            F.col('STATE').alias('CODE'),
            F.col('count(ID)').alias('number_of_stations')
        ),
        on='CODE', how='left')
    )
states.show()

states.write.csv("hdfs:///user/gtj13/outputs/ghcnd/states", mode = 'overwrite',
header = True)
```

Output

```
+----+--------------------+-----------------+
|CODE|                NAME|number_of_stations|
+----+--------------------+-----------------+
|  AB|             ALBERTA|             1404|
|  AK|              ALASKA|              982|
|  AL|             ALABAMA|              926|
|  AR|            ARKANSAS|              830|
|  AS|      AMERICAN SAMOA|               21|
|  AZ|             ARIZONA|             1327|
|  BC|    BRITISH COLUMBIA|             1668|
|  CA|          CALIFORNIA|             2663|
|  CO|            COLORADO|             3854|
|  CT|         CONNECTICUT|              233|
|  DC|DISTRICT OF COLUMBIA|               14|
|  DE|            DELAWARE|              102|
|  FL|             FLORIDA|             1460|
|  FM|          MICRONESIA|               38|
|  GA|             GEORGIA|             1120|
|  GU|                GUAM|               20|
|  HI|              HAWAII|              720|
|  IA|                IOWA|              748|
|  ID|               IDAHO|              740|
|  IL|            ILLINOIS|             1630|
+----+--------------------+-----------------+
only showing top 20 rows
```

Upon setting up the new states count data frame we could explore bit about the data set and ask specific questions to find out interesting factors such as,

There are 25337 stations in the Southern Hemisphere and 56918 stations in are there in total in the United States territories.
Following example codes snippet is the one to use to find out about facts.

```
stations_inventory.filter(col('LATITUDE') < 0).count()
##Out[33]: 25337 stations in Southern Hemisphere

stations_inventory.filter(col('COUNTRY_NAME') == 'United States').count()
##Out[34]: 56918 stations are there in total in the territories of the United States
```

We'll look at how to write a user define function in spark (udf) and as an instance here we'll consider the distance between two stations using their latitude and longitude as arguments. Later we could test this function by using CROSS JOIN on small subset of stations to generate a table with two stations in each row. In this case we'll use the fact that the earth is spherical and take the geographical distances. The base equation for the same could list as follows,

Commented [GJ3]: Insert the equation over here, latex

The great-circle distance or orthodromic distance is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere and in general the equation as follows,

Let,

$$\boldsymbol{\phi_1, \lambda_1 \text{ and } \phi_2, \lambda_2}$$

be the geographical latitude and longitude, the central angle $\boldsymbol{\Delta\sigma}$ is given by the spherical law of cosines and then the equation becomes,

$$\boldsymbol{\Delta\sigma = \arccos(\sin\phi_1 \phi_2 + \cos\phi_1 \phi_2 \cos(\Delta\lambda))}$$

The actual arc length d on a sphere of the radius r can be trivially computed as

$$\boldsymbol{d = r\,\Delta\sigma}$$

We take the 'r' as the mean earth radius in this case. In pyspark we can import math it's among the built-in function in Pyspark.

When writing a function to calculate I've listed the same calculation using two distinct way in two operations which perform in the same manner.

Basic function define type in python is def FUNCTION_NAME (): in the following function we have done is performing what defined in equation ### and fold it as a user define function by assign it in to a class here in this it's calculate_udf

```
import math

def calculation(la1, lo1, la2, lo2):
    R = 6373.0
    dlo = radians(lo2 - lo1)
    dla = radians(la2 - la1)

    a = (sin(dla/2))**2 + cos(radians(la1)) * cos(radians(la2)) * (sin(dlo/2))**2

    c = 2 * atan2(sqrt(a), sqrt(1-a))
```

```
    distance = R * c
    return distance


calculation_udf = F.udf(lambda x, y, a, b: calculation(x, y, a, b), DoubleType())
-------------------------------------------------------------------------------
def calculation(la1, lo1, la2, lo2):
    R = 6373.0
    distance = math.acos(
        math.sin(math.radians(la1)) * math.sin(math.radians(la2)) +
        math.cos(math.radians(la1)) * math.cos(math.radians(la2)) *
        math.cos(math.radians(lo1 - lo2))
    ) * R
    return distance

calculation_udf = F.udf(lambda x, y, a, b: calculation(x, y, a, b), DoubleType())
```

We can perform few of the tests and confirm the functions behaviour to see whether it works fine I've listed few of the test from line 653 to 674 and commented out in the file.

Let's select New Zealand as the country in this instance and calculate the pairwise distances between all stations in New Zealand.

- ✓ In the first step I've filtered the country name == New Zealand out of the stations_inventory data table.
- ✓ By using the select () function I've filtered the Latitude, Longitude, and Name as I mentioned earlier have used the toDF() function to change the names of the selected columns and saved the data frame as NZ1
- ✓ Following the same I've saved NZ2
- ✓ As the final step save the data NZ_result and in this operation we can use withColumn () function creating a new column with the colExpr which is in this case calculation_udf() and as input we enter four parameters as we have defined in the function earlier.

```
NZ = stations_inventory.filter(col('COUNTRY_NAME') == 'New Zealand')

NZ1 = NZ.select('NAME', 'LATITUDE', 'LONGITUDE').toDF('NAME1', 'LATITUDE1',
'LONGITUDE1')
NZ2 = NZ.select('NAME', 'LATITUDE', 'LONGITUDE').toDF('NAME2', 'LATITUDE2',
'LONGITUDE2')

NZ = NZ1.crossJoin(NZ2).filter(F.col('NAME1') != F.col('NAME2'))
NZ.show(n = 10)

NZ_result = NZ.withColumn('distance', calculation_udf(F.col('LATITUDE1'),
F.col('LONGITUDE1'), F.col('LATITUDE2'), F.col('LONGITUDE2')))
NZ_result.show()

NZ_result.orderBy("distance").show()

NZ_result.write.csv("hdfs:///user/gtj13/outputs/ghcnd/NZ_result")
```

Can use the following command to determine the default block size of HDFS

```
hdfs getconf -confKey "dfs.blocksize"
```

We can use the following hdfs codes to identify the required block size to get all of the daily climate summaries, this would basically help to determine how efficiently the we can load and apply the transformation to daily.

To determine the size of files under a specific path in HDFS we can use,

```
hdfs dfs -ls hdfs:///data/ghcnd/daily/2017*
```

We can use the below command to use to determine the default block size of HDFS.

```
hdfs getconf -confKey "dfs.blocksize"##134217728
```

Results would be 134217728 for the year 2017.

To check further we could work on the following hdfs commands to get further details and determine,

```
hdfs dfs -ls hdfs:///data/ghcnd/daily/2010*

#Output
##207181730

hdfs fsck hdfs:///data/ghcnd/daily/2010.csv.gz -files -blocks ##134217728 72964002
```

In the above code snippet what we have determined is the block size of 2010 and the total of the block size is

Commented [GJ4]: Check on theoutput file

```
#Output

#0. BP-1663138130-132.181.39.102-1551363950352:blk_1073751760_10936 len=134217728 Live_repl=4
#1. BP-1663138130-132.181.39.102-1551363950352:blk_1073751761_10937 len=72964002 Live_repl=4
```

Let's run a test by counting the number of rows in the daily schema.

As the step 1 define the schema for daily,

```
schema_Daily = StructType([
    StructField("ID", StringType(), True),
    StructField("DATE", StringType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("VALUE", DoubleType(), True),
    StructField("MEASUREMENT", StringType(), True),
    StructField("QUALITY FLAG", StringType(), True),
    StructField("SOURCE FLAG", StringType(), True), StructField("OBSERVATION TIME",
StringType(), True),
])


daily_2010_2017 = (
    spark.read.format("com.databricks.spark.csv")
    .option("header", False)
    .option("inferSchema", False)
    .schema(schema_Daily)
    .load("hdfs:///data/ghcnd/daily/{2010,2017}.csv.gz")
)
daily_2010_2017.count()
```

Output value

```
###Out[5]: 58851079
```

By using the 'mathmadslinux1p:8080' user interface we could check on the executor details and see we further optimize the tasks. Following are the results I received by looking at it.

Figure 1 displays the number of spark jobs ran during this process.

The sequence of events here is straight forward. Shortly after all executors have registered, the application runs 1 job and it didn't fail. Then, when once the job is finished and the application exists, the executors are removed with it.

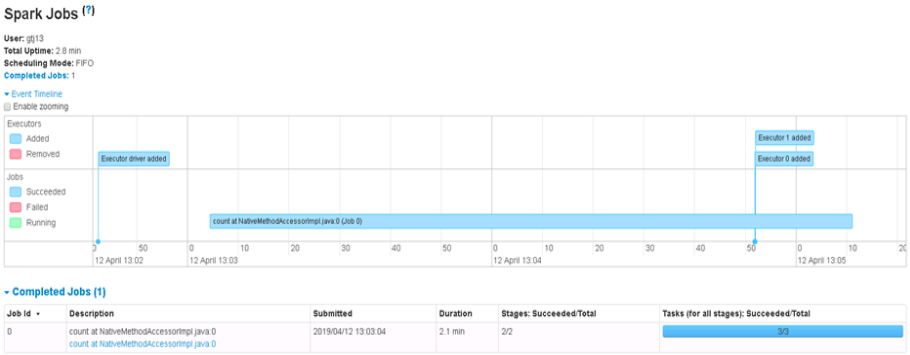
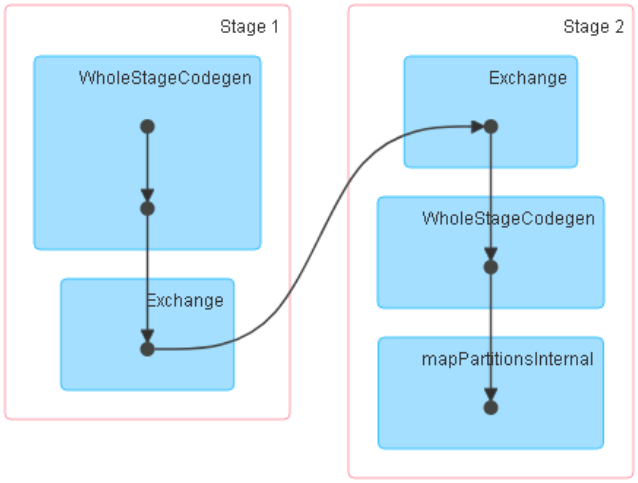
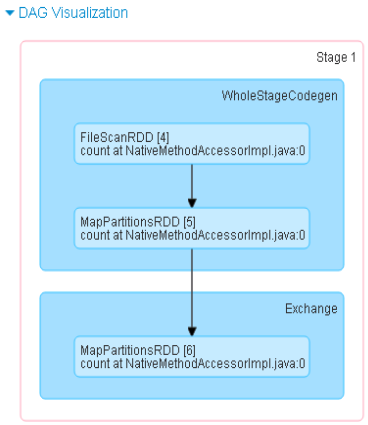
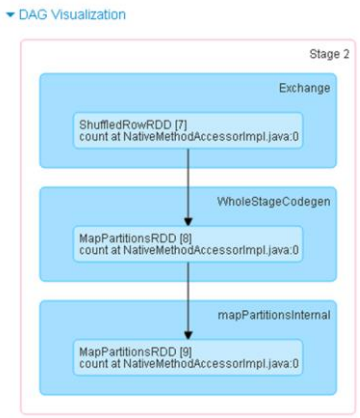


Figure 2 displays the DAG (Directed Acyclic Graph) which contains all the fellow operations from RDD to another RDD. In little more technical aspects; there are finitely many edges and vertices, where each edge directed from one vertex to another. It contains a sequence of vertices such that every edge is directed from earlier or later in the sequence. The DAG operation could perform a better global optimization than other systems, it provides more clearer picture once it comes to more complex jobs.



In above execution DAG we can notice that we had stage1 and stage 2.





Looking at the above two task DAG executors we can determine that the at the stage one we had a file scan and made MAP partitions and at the 2nd stage we had a shuffle of the rows and Map partitions operations. If take a little deeper look at the 'WholeStageCodegen', code generation is one of the primary components of the Spark SQL engine's Catalyst Optimizer. In summary it does the followings,

1. Analysing a logical plan to resolve references
2. Logical plan optimization
3. Physical planning and
4. Code generation

In brief it does the looking up relations by name from the catalog and mapping name attributes such as col to the input provided given operator's children. Determining which attributes refer to the same value to give them a unique ID. at the final stage the code generation step; query optimization involves generating Java bytecode to run on each machine.

Summary matrices are as follows, we can select more details in to the summary matrices by ticking required execution parameters but I tend to keep it the below way.

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	**0.1 s**	**0.1 s**	**0.1 s**	**0.1 s**	**0.1 s**
GC Time	**0 ms**	**0 ms**	**0 ms**	**0 ms**	**0 ms**
Shuffle Read Size / Records	**118.0 B / 2**	**118.0 B / 2**	**118.0 B / 2**	**118.0 B / 2**	**118.0 B / 2**

For daily six years from 2010 to 2015 the final row count comes as 207716098, following code snippet

```
daily_sixyear = (
    spark.read.format("com.databricks.spark.csv")
    .option("header", False)
    .option("inferSchema", False)
    .schema(schema_Daily)
    .load("hdfs:///data/ghcnd/daily/{2010,2011,2012,2013,2014,2015}.csv.gz")
)
daily_sixyear.count()
daily_sixyear.show()
##Out[6]: 207716098
```

Summary show as follows,

```
+-----------+--------+-------+-----+-----------+------------+-----------+----------------+
|         ID|    DATE|ELEMENT|VALUE|MEASUREMENT|QUALITY FLAG|SOURCE FLAG|OBSERVATION TIME|
+-----------+--------+-------+-----+-----------+------------+-----------+----------------+
|ASN00015643|20100101|   TMAX|414.0|       null|        null|          a|            null|
|ASN00015643|20100101|   TMIN|254.0|       null|        null|          a|            null|
|ASN00015643|20100101|   PRCP|  0.0|       null|        null|          a|            null|
|US1MSHD0003|20100101|   PRCP| 41.0|       null|        null|          N|            null|
|US1MODG0003|20100101|   PRCP|  0.0|       null|        null|          N|            null|
|US1MODG0003|20100101|   SNOW|  0.0|       null|        null|          N|            null|
|US1MODG0003|20100101|   SNWD|  0.0|       null|        null|          N|            null|
|US1MODG0003|20100101|   WESD|  0.0|       null|        null|          N|            null|
|US1MODG0003|20100101|   WESF|  0.0|       null|        null|          N|            null|
|US1MOCW0004|20100101|   PRCP|  0.0|       null|        null|          N|            null|
|US1MOCW0004|20100101|   SNOW|  0.0|       null|        null|          N|            null|
|ASN00085296|20100101|   TMAX|277.0|       null|        null|          a|            null|
|ASN00085296|20100101|   TMIN|170.0|       null|        null|          a|            null|
|ASN00085296|20100101|   PRCP| 58.0|       null|        null|          a|            null|
|ASN00085280|20100101|   TMAX|256.0|       null|        null|          a|            null|
|ASN00085280|20100101|   TMIN|186.0|       null|        null|          a|            null|
|ASN00085280|20100101|   PRCP| 16.0|       null|        null|          a|            null|
|ASN00040209|20100101|   TMAX|283.0|       null|        null|          a|            null|
|ASN00040209|20100101|   TMIN|224.0|       null|        null|          a|            null|
|ASN00040209|20100101|   PRCP| 14.0|       null|        null|          a|            null|
+-----------+--------+-------+-----+-----------+------------+-----------+----------------+
```

Apache Spark allows developers to run multiple tasks in parallel across machines in a cluster, or across multiple cores on a desktop. A partition, or split, is a logical chunk of a distributed data set. Apache Spark builds a Directed Acyclic Graph (DAG) with jobs, stages, and tasks for the submitted application. The number of tasks will be determined based on the number of partitions.

The general principles to be followed when tuning partition for Spark application are as follows:

- ✓ Too few partitions – Cannot utilize all cores available in the cluster.
- ✓ Too many partitions – Excessive overhead in managing many small tasks.
- ✓ Reasonable partitions – Helps us to utilize the cores available in the cluster and avoids excessive overhead in managing small tasks.

Let's look at daily in more detail, we use the following code snippet to execute the work

```
daily_all = (
    spark.read.format("com.databricks.spark.csv")
    .option("header", False)
    .option("inferSchema", False)
    .schema(schema_Daily )
    .load("hdfs:///data/ghcnd/daily/")
)
daily_all.count()
```

We can see the output as follows, and it indicates we have all the daily count as `2624027105`.

```
# Out[85]: 2624027105
```

We can now filter out the ELEMENT column and see what's the total out of each element, looking at the data table we can see that there are 5 elements, we can write one-line code as follows,

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

and get the output as,

```
#Out[11]: 2225133835
```

Let's cascade it and look at in more understandable way,

```
daily_all.
filter(F.col('ELEMENT').isin('PRCP','SNOW','SNWD','TMAX','TMIN')).groupby('ELEMENT').agg({'ELEME
NT':'count'}).show()
```

Output to get the total count on each element.

```
+-------+--------------+
|ELEMENT|count(ELEMENT)|
+-------+--------------+
|   SNOW|     322003304|
|   SNWD|     261455306|
|   PRCP|     918490401|
|   TMIN|     360656728|
|   TMAX|     362528096|
+-------+--------------+
```

As a fact it was observable that the Many stations collect TMAX and TMIN, but do not necessarily report them simultaneously due to issues with data collection or coverage, taking this fact to an account we can determine how many observations of TMIN do not have a corresponding observation of TMAX,

```
MAX_dairy = daily_all.filter(F.col('ELEMENT') == 'TMAX')
TMIN_dairy = daily_all.filter(F.col('ELEMENT') == 'TMIN')
difference = TMIN_dairy. select('ID','DATE'). subtract (TMIN_dairy.select('ID','DATE'))
difference.count()
difference.select('ID').distinct().count()
```

Output came as 0 for this one.

Let's look at how many different stations contributed to these observations,

```
station_three_observations = stations_inventory.filter((F.col('HCN/CRN FLAG') != '')|(F.col('GSN
FLAG') != ''))
difference.select('ID').intersect(station_three_observations.select('STATION_ID')).count()
```

```
dailys_TT = daily_all.filter(
        (F.substring(
            F.col('ID'), 1, 2) == 'NZ')&
             ((F.col('ELEMENT') == 'TMAX')|
                    (F.col('ELEMENT') == 'TMIN')))

dailys_TT.count()
```

Output is 447017 and we can find the average number as follows,

```
dailys_TT_AV = daily_all.groupBy(
                    'DATE', 'ELEMENT').agg(F.avg('VALUE').alias('AVERAGE'))

dailys_TT_AV.show()
```

Let's use the element precipitation to calculate the average rain fall in each year for each country, will save this results for later use. By using the following code snippet,

```
rainfall = daily_all.filter(F.col('element') == 'PRCP')

rainfall = (rainfall
    .select('ID', 'DATE', 'ELEMENT', 'VALUE')
    .withColumn('country_code', F.substring(F.col('DATE'), 1, 4))
    .withColumn('year',  F.substring(F.col('ID'), 1, 2))
    )
Rainfall.show()


## grouping
rainfall = (rainfall.groupby('country_code', 'year').agg({'value':'mean'}))
rainfall.write.format('com.databricks.spark.csv').options(delimiter =
',').save('hdfs:///user/gtj13/outputs/ghcnd/rainfall/', mode = 'overwrite', header = True)
```

In the final bit when saving the file if we save like that we get all the files from each partition we had and it basically time consuming process to combine all the csv file and it's prone for more errors, the code could change in to a bit if the file sizes are in less than few GBs.

```
rainfall.coalesce(1).write.format('com.databricks.spark.csv').options(delimiter =
',').save('hdfs:///user/gtj13/outputs/ghcnd/rainfall/', mode = 'overwrite', header = True)

use dbutils afters
```

Note sure whether this is ideal way. But could use the function as follows(haven't tried)

```
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs._

def merge(srcPath: String, dstPath: String): Unit =  {
   val hadoopConfig = new Configuration()
   val hdfs = FileSystem.get(hadoopConfig)
   FileUtil.copyMerge(hdfs, new Path(srcPath), hdfs, new Path(dstPath), true, hadoopConfig,
null)
   // the "true" setting deletes the source files once they are merged into the new output
}
```

Else ideal way should have been

```
Hadoop hdfs -getmerge
```