

DATA420-18S2(c)

Assignment 1

GHCN Data Analysis using Spark

Shun Li

65322005

In this assignment we will investigate the weather data collected by the Global Historical Climate Network (GHCN), an integrated database of climate summaries from weather stations around the world. This data covers the last 175 years, is collected from over 20 independent sources, and contains records from over 100,000 stations in 180 countries around the world.

The code used to solve each of the questions is provided separately and is commented well. This write up describes the approach used, answers any questions, and describes anything unexpected along the way.

Processing

Q1

Define the separate data sources as daily, stations, states, countries, and inventory respectively. All of the data is stored in HDFS under `hdfs:///data/ghcnd` and is read only. Do not copy any of the data to your home directory.

Use the `hdfs` command to explore `hdfs:///data/ghcnd` without actually loading any data into memory.

(a) How is the data structured? Draw a directory tree to represent this in a sensible way.

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

```
Found 5 items
-rw-r--r--  8 hadoop supergroup      3.6 K 2018-08-23 16:54 hdfs:///data/ghcnd/countries
drwxr-xr-x  - hadoop supergroup        0 2018-08-23 16:52 hdfs:///data/ghcnd/daily
-rw-r--r--  8 hadoop supergroup    26.1 M 2018-08-23 16:54 hdfs:///data/ghcnd/inventory
-rw-r--r--  8 hadoop supergroup      1.1 K 2018-08-23 16:54 hdfs:///data/ghcnd/states
-rw-r--r--  8 hadoop supergroup      8.5 M 2018-08-23 16:54 hdfs:///data/ghcnd/stations
```

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

```
Found 255 items
-rw-r--r--  8 hadoop supergroup    3367 2018-08-23 16:45 hdfs:///data/ghcnd/daily/1763.csv.gz
-rw-r--r--  8 hadoop supergroup    3336 2018-08-23 16:45 hdfs:///data/ghcnd/daily/1764.csv.gz
-rw-r--r--  8 hadoop supergroup    3344 2018-08-23 16:45 hdfs:///data/ghcnd/daily/1765.csv.gz
-rw-r--r--  8 hadoop supergroup    3353 2018-08-23 16:45 hdfs:///data/ghcnd/daily/1766.csv.gz
...
-rw-r--r--  8 hadoop supergroup 195876227 2018-08-23 16:52 hdfs:///data/ghcnd/daily/2012.csv.gz
-rw-r--r--  8 hadoop supergroup 196551118 2018-08-23 16:52 hdfs:///data/ghcnd/daily/2013.csv.gz
-rw-r--r--  8 hadoop supergroup 193121272 2018-08-23 16:52 hdfs:///data/ghcnd/daily/2014.csv.gz
-rw-r--r--  8 hadoop supergroup 196007583 2018-08-23 16:52 hdfs:///data/ghcnd/daily/2015.csv.gz
-rw-r--r--  8 hadoop supergroup 194390036 2018-08-23 16:52 hdfs:///data/ghcnd/daily/2016.csv.gz
-rw-r--r--  8 hadoop supergroup 125257391 2018-08-23 16:52 hdfs:///data/ghcnd/daily/2017.csv.gz
```

So, we can draw a directory tree as followings:

```
/data/shared/ghcnd/
├── countries
├── daily
│   ├── 1763.csv.gz
│   ├── 1764.csv.gz
│   ├── 1765.csv.gz
│   ├── ...
│   └── 2017.csv.gz
├── inventory
├── states
└── stations
```

(b) How many years are contained in daily, and how does the size of the data change?

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

```
3.3 K    26.3 K    hdfs:///data/ghcnd/daily/1763.csv.gz
3.3 K    26.1 K    hdfs:///data/ghcnd/daily/1764.csv.gz
3.3 K    26.1 K    hdfs:///data/ghcnd/daily/1765.csv.gz
3.3 K    26.2 K    hdfs:///data/ghcnd/daily/1766.csv.gz
...
184.2 M   1.4 G    hdfs:///data/ghcnd/daily/2014.csv.gz
186.9 M   1.5 G    hdfs:///data/ghcnd/daily/2015.csv.gz
185.4 M   1.4 G    hdfs:///data/ghcnd/daily/2016.csv.gz
119.5 M   955.6 M  hdfs:///data/ghcnd/daily/2017.csv.gz
```

From the result, we can find total year is **255 years**(from 1763 to 2017).

And the **red part** of the result table is the size of data for each year. we can find that the size of each year has changed **bigger and bigger in the whole trend**.

(c) What is the total size of all of the data? How much of that is daily?

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

```
13.4 G    107.2 G    hdfs:///data/ghcnd
1.9 K     15 K        hdfs:///data/helloworld
3.7 M     29.3 M       hdfs:///data/openflights
```

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

```
3.6 K     28.7 K    hdfs:///data/ghcnd/countries
13.4 G    106.9 G    hdfs:///data/ghcnd/daily
26.1 M    209.1 M    hdfs:///data/ghcnd/inventory
1.1 K     8.5 K      hdfs:///data/ghcnd/states
8.5 M     68.0 M     hdfs:///data/ghcnd/stations
```

The total size of the data is 13.4 GB, most of which is daily. All of the metadata combined only contributes 36.3 MB to the total. As the daily data is compressed, and the actual size of the uncompressed data will be significantly higher. The gzip compression ratio for csv files is usually around 10:1 so this could be between 100 GB and 1 TB.

Q2

- (a) Define schemas for each of daily, stations, states, countries, and inventory based on the descriptions in this assignment and the GHCN Daily README. These should use the data types defined in pyspark.sql.

The schema of daily is described in the assignment already, and the head of each file is consistent with the schema as expected. To look at the head of daily, we had to pipe the file through gunzip and pipe the decompressed csv data to head.

Define schemas of daily:

```
./start-pyspark.sh
# Imports

from pyspark.sql.types import *
import pyspark.sql.functions as F

# Load

schema_daily = StructType([
    StructField("ID1", StringType(), True),
    StructField("DATE", StringType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("VALUE", FloatType(), True),
    StructField("MEASUREMENT FLAG", StringType(), True),
    StructField("QUALITY FLAG", StringType(), True),
    StructField("SOURCE FLAG", StringType(), True),
    StructField("OBSERVATION TIME", StringType(), True)
])
```

The schema for countries, inventory, stations, and states could not be applied while they were read from HDFS, as the fixed width text format requires columns to be extracted by selecting the relevant substring of the text record. This was achieved by loading each file into a dataframe with one text column and using a select statement to extract substrings, trim any whitespace, and apply the datatype from the schema based on column name.

Define schemas of stations:

```
schema_station = StructType([
    StructField("ID", StringType(), True),
    StructField("LATITUDE", IntegerType(), True),
    StructField("LONGITUDE", IntegerType(), True),
    StructField("ELEVATION", IntegerType(), 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)
])
```

Define schemas of countries:

```
schema_countries = StructType([
    StructField("CODE", StringType(), True),
    StructField("NAME1", StringType(), True)
])
```

Define schemas of states:

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

Define schemas of inventory:

```
schema_inventory = StructType([
    StructField("ID2", StringType(), True),
    StructField("LATITUDE", IntegerType(), True),
    StructField("LONGITUDE", IntegerType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("FIRSTYEAR", IntegerType(), True),
    StructField("LASTYEAR", IntegerType(), True)
])
```

- (b) Load 1000 rows of `hdfs:///data/ghcnd/daily/2017.csv.gz` into Spark using the `limit` command immediately after the read command. Was the description of the data accurate? Was there anything unexpected?

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

Result:

```
In [4]: daily.show(5,False)
+-----+-----+-----+-----+-----+-----+-----+-----+
|ID1|DATE|ELEMENT|VALUE|MEASUREMENT FLAG|QUALITY FLAG|SOURCE FLAG|OBSERVATION TIME|
+-----+-----+-----+-----+-----+-----+-----+-----+
|CA1MB000296|20170101|PRCP|0.0|null|null|N|null|
|US1NCBC0113|20170101|PRCP|5.0|null|null|N|null|
|ASN00015643|20170101|TMAX|274.0|null|null|a|null|
|ASN00015643|20170101|TMIN|218.0|null|null|a|null|
|ASN00015643|20170101|PRCP|2.0|null|null|a|null|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Based on loading the first 1000 rows of `daily` in pyspark the description of the data above and in the assignment is accurate. However, while the `DATE` column represents the date and the `OBSERVATION TIME` column represents a time, they are not formatted in a way that `DateTime` and `TimestampType` can parse automatically.

These columns had to be loaded as `StringType` instead, and converted to `DateTime` and `TimestampType` by hand.

- (c) Load each of `stations`, `states`, `countries`, and `inventory` into Spark as well. You will need to find a way to parse the fixed width text formatting, as this format is not included in the standard `spark.read` library. You could try using `spark.read.format('text')` and `pyspark.sql.functions.substring` or finding an existing open source library.
- How many stations do not have a WMO ID?

Load stations:

```
station = (  
    spark.read.text("hdfs:///data/ghcnd/stations")  
)  
from pyspark.sql.functions import trim  
  
stations = station.select(  
    station.value.substr(1,11).alias('ID'),  
    station.value.substr(13,8).alias('LATITUDE'),  
    station.value.substr(22,9).alias('LONGITUDE'),  
    station.value.substr(32,6).alias('ELEMENT'),  
    station.value.substr(39,2).alias('STATE'),  
    station.value.substr(42,30).alias('NAME'),  
    station.value.substr(73,3).alias('GSN_FLAG'),  
    station.value.substr(77,3).alias('HCN_CRN_FLAG'),  
    trim(station.value.substr(81,5)).alias('WMO_ID')  
)
```

Load countries:

```
countries = (  
    spark.read.text("hdfs:///data/ghcnd/countries")  
)  
  
countries = countries.select(  
    countries.value.substr(1,2).alias('CODE'),  
    countries.value.substr(4,47).alias('NAME1')  
)
```

Load states:

```
state = (  
    spark.read.text("hdfs:///data/ghcnd/states")  
)  
  
states = state.select(  
    state.value.substr(1,2).alias('CODE'),  
    state.value.substr(4,47).alias('NAME3')  
)
```

Load inventory:

```
inventory = (spark.read.text("hdfs:///data/ghcnd/inventory"))

inventory = inventory.select(
    inventory.value.substr(1,11).alias('ID2'),
    inventory.value.substr(13,8).alias('LATITUDE'),
    inventory.value.substr(22,9).alias('LONGITUDE'),
    inventory.value.substr(32,4).alias('ELEMENT'),
    inventory.value.substr(37,4).alias('FIRSTYEAR'),
    inventory.value.substr(42,4).alias('LASTYEAR')
)
```

How many rows are in each metadata table?

```
In [10]: stations.count()
Out[10]: 103656

In [11]: states.count()
Out[11]: 74

In [12]: countries.count()
Out[12]: 218

In [13]: inventory.count()
Out[13]: 595699
```

The stations has 103656 rows; states has 74 rows; countries has 218 rows, in total, representing 218 distinct countries or territories from around the world; inventory has 595699 rows in total.

How many stations do not have a WMO ID?

```
stations1 = stations.withColumn(
    "WMO_ID1",
    trim(stations.WMO_ID)
)

stations1_null = stations1.filter(stations1.WMO_ID1 == "")
stations1_null.count()
```

```
Out[14]: 95595
```

Here, using the trim function to remove the space in the column of WMO ID to make it easy to count the number. So, 95595 stations do not have a WMO ID.

Q3

Next you will combine the daily climate summaries with the metadata tables, joining on station, state, and country. Note that joining the daily climate summaries and metadata into a single table is not efficient for a database of this size, but joining the metadata into a single table is very convenient for filtering and sorting based on attributes at a station level. You will need to start saving some intermediate outputs along the way. Create an output directory in your home directory (e.g. `hdfs:///user/abc123/outputs/ghcnd/`). Note that we only have 400GB of storage available in total, so think carefully before you write output to HDFS.

```
hadoop fs -mkdir <hdfs:///user/sli171/output/ghcnd>
```

Using `mkdir` to create a new output directory in the home directory.

(a) Extract the two character country code from each station code in `stations` and store the output as a new column using the `withColumn` command.

```
stations = stations.withColumn(
    "COUNTRY_CODE",
    stations.ID.substr(1,2)
)
```

Result:

```
In [18]: stations.show(5,False)
```

ID	LATITUDE	LONGITUDE	ELEMENT	STATE	NAME	GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE
ACW00011604	17.1167	-61.7833	10.1		ST JOHNS COOLIDGE FLD				AC
ACW00011647	17.1333	-61.7833	19.2		ST JOHNS				AC
AEO00041196	25.3330	55.5170	34.0		SHARJAH INTER. AIRP	GSN		41196	AE
AEM00041194	25.2550	55.3640	10.4		DUBAI INTL			41194	AE
AEM00041217	24.4330	54.6510	26.8		ABU DHABI INTL			41217	AE

only showing top 5 rows

Using the `substr` function to extract the first two character country code from the station ID code and using `withColumn` to store it as a new column. Like the result shows.

(b) LEFT JOIN `stations` with `countries` using your output from part (a).

```
station_country = stations.join(
    countries, stations.COUNTRY_CODE == countries.CODE, "left").distinct()
```

Using the `join` function to join the `stations` with `countries` by the `COUNTRY_CODE` in `stations` is equal to `CODE` in `countries`, and left join.

Result:

```
In [21]: station_country.show(5,False)
....:
```

ID	LATITUDE	LONGITUDE	ELEMENT	STATE	NAME	GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE	CODE	NAME1
AFM00040938	34.2100	62.2280	977.2		HERAT			40938	AF	AF	Afghanistan
AFM00040990	31.5000	65.8500	1010.0		KANDAHAR AIRPORT			40990	AF	AF	Afghanistan
AG000060680	22.8000	5.4331	1362.0		TAMANRASSET	GSN		60680	AG	AG	Algeria
AGE00147705	36.7800	3.0700	59.0		ALGIERS-VILLE/UNIVERSITE				AG	AG	Algeria
AGE00147709	36.6300	4.2000	942.0		FORT NATIONAL				AG	AG	Algeria

only showing top 5 rows

(c) LEFT JOIN stations and states, allowing for the fact that state codes are only provided for stations in the US.

```
station_state = stations.join(states,stations.STATE == states.CODE,"left")
```

Result:

```
In [23]: station_state.show(5,False)
....:
```

ID	LATITUDE	LONGITUDE	ELEMENT	STATE	NAME	GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE	CODE	NAME3
ACW00011604	17.1167	-61.7833	10.1		ST JOHNS COOLIDGE FLD				AC	null	null
ACW00011647	17.1333	-61.7833	19.2		ST JOHNS				AC	null	null
AE000041196	25.3330	55.5170	34.0		SHARJAH INTER. AIRP	GSN		41196	AE	null	null
AEW00041194	25.2550	55.3640	10.4		DUBAI INTL			41194	AE	null	null
AEW00041217	24.4330	54.6510	26.8		ABU DHABI INTL			41217	AE	null	null

only showing top 5 rows

(d)

Based on inventory, what was the first and last year that each station was active and collected any element at all?

```
#### Method1:

inventory_year = inventory.groupBy('ID2').agg({'FIRSTYEAR': 'min','LASTYEAR': 'max'})

#### Method 2:

inventory_year = inventory.groupBy('ID2').agg(F.min("FIRSTYEAR").alias("FirstYear"),
                                              F.max("LASTYEAR").alias('LastYear')
                                              )
```

Using the groupby to group the data according to the ID and using the min function to find the minimum year for the whole FIRSTYEAR column ,and the max function to find the maximum year for the whole LAST YEAR, and using alias function to rename them as FirstYear and last year respectively.

Result:

```
In [25]: inventory_year.show(5,False)
+-----+-----+-----+
|ID2      |FirstYear|LastYear|
+-----+-----+-----+
|ACW00011647|1957      |1970     |
|AEM00041217|1983      |2017     |
|AG000060590|1892      |2017     |
|AGE00147706|1893      |1920     |
|AGE00147708|1879      |2017     |
+-----+-----+-----+
only showing top 5 rows
```

How many different elements has each station collected overall?

Further, count separately the number of core elements and the number of "other" elements that each station has collected overall.

I have put two methods here :

```
#### Method1:

inventory_element = inventory.groupBy('ID2').agg((F.count("ELEMENT").alias("DISTINCT_ELEMENT")))

inventory_core_1 = inventory.filter(inventory.ELEMENT == "PRCP").distinct()
inventory_core_2 = inventory.filter(inventory.ELEMENT == "SNOW").distinct()
inventory_core_3 = inventory.filter(inventory.ELEMENT == "SNWD").distinct()
inventory_core_4 = inventory.filter(inventory.ELEMENT == "TMAX").distinct()
inventory_core_5 = inventory.filter(inventory.ELEMENT == "TMIN").distinct()
inventory_core = inventory_core_1.unionAll(inventory_core_2)
inventory_core = inventory_core.unionAll(inventory_core_3)
inventory_core = inventory_core.unionAll(inventory_core_4)
inventory_core = inventory_core.unionAll(inventory_core_5)

inventory_core = inventory_core.groupBy('ID2').agg((F.count("ELEMENT").alias("DISTINCT_CORE_ELEMENT")))

inventory_other_1 = inventory.filter(inventory.ELEMENT != "PRCP").distinct()
inventory_other_2 = inventory_other_1.filter(inventory_other_1.ELEMENT != "SNOW").distinct()
inventory_other_3 = inventory_other_2.filter(inventory_other_2.ELEMENT != "SNWD").distinct()
inventory_other_4 = inventory_other_3.filter(inventory_other_3.ELEMENT != "TMAX").distinct()
inventory_other = inventory_other_4.filter(inventory_other_4.ELEMENT != "TMIN").distinct()

inventory_other = inventory_other.groupBy('ID2').agg((F.count("ELEMENT").alias("DISTINCT_OTHER_ELEMENT")))
```

In this method,

for core elements:

firstly using filter function to filter each core element and using the unionAll function to put the together and count each element by the grouped ID.

For other elements:

Remove the one core element each time and using the result that created each step ,then remove again to remove the total five core elements. And then count the elements by ID.

The method2 is a very smart way ,here create a list consisting of five core elements. And using When and otherwise to give the value for each different conditions.Like: if the element is in the five core elements , give them the value of 1, otherwise give them 0.

Fisrt , count the total elememt ;

Then using the sum function , here because when we add the 0 to sum ,it means we have not added anything ,which is that we remove the other elements automatically. Just count the core elements.

Third, using the count result for all elements to minus the count for core element ,we can get the count for other element.

```
##### Method2:

core_elements = ["PRCP","SNOW","SNWD","TMAX","TMIN"]
inventory = inventory.withColumn(
    "core_flag",
    F.when(inventory.ELEMENT.isin(core_elements),1)
    .otherwise(0)
)
inventory_enriched = inventory.groupBy('ID2').agg(F.count("ELEMENT").alias("elements_count"),
    F.sum("core_flag").alias('core_elements'),
    (F.count("ELEMENT") - F.sum('core_flag')).alias("other_elements")
)
```

Result:

```
In [27]: inventory_enriched.show(5,False)
+-----+-----+-----+-----+
|ID2      |elements_count|core_elements|other_elements|
+-----+-----+-----+-----+
|ACW00011647|7             |5             |2             |
|AEM00041217|4             |3             |1             |
|AG000060590|4             |3             |1             |
|AGE00147706|3             |3             |0             |
|AGE00147708|5             |4             |1             |
+-----+-----+-----+-----+
only showing top 5 rows
```

How many stations collect all five core elements?

```
inventory_core_all = inventory_enriched.filter(inventory_enriched.core_elements == 5)
inventory_core_all.count()
```

Result:

```
Out[28]: 20224
```

Just filter the core element if the value of it is equal to 5. And count the result.

Here , **20224** stations collect all five core elements.

Note that 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.

(e) LEFT JOIN stations and your output from part (d).

This enriched stations table will be useful. Save it to your output directory. Think carefully about the file format that you use (e.g. csv, csv.gz, parquet) with respect to consistency and efficiency. From now on assume that stations refers to this enriched table with all the new columns included.

```
station_inventory = stations.join(inventory_enriched,stations.ID == inventory_enriched.ID2,"left")
```

Result:

```
In [30]: station_inventory.show(5,False)
```

ID	LATITUDE	LONGITUDE	ELEMENT	STATE	NAME	GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE	ID2	elements_count	core_elements	other_elements
ACW00011647	17.1333	-61.7833	19.2		ST JOHNS				AC	ACW00011647	7	5	2
AEM00041217	24.4330	54.6910	26.8		ABU DHABI INTL			41217	AE	AEM00041217	4	3	1
AGE00060590	30.5667	2.8667	397.0		EL-GOLEA	GSN		60590	AG	AGE00060590	4	3	1
AGE00147706	36.8000	3.0300	344.0		ALGIERS-BOUZAREAH				AG	AGE00147706	3	3	0
AGE00147708	36.7200	4.0500	222.0		TIZI OUZOU			60395	AG	AGE00147708	5	4	1

only showing top 5 rows

(f) LEFT JOIN your 1000 rows subset of daily and your output from part (e). Are there any stations in your subset of daily that are not in stations at all?

```
station_inventory_daily = station_inventory.join(daily,station_inventory.ID == daily.ID1,"left")
station_inventory_daily = station_inventory_daily.drop(station_inventory_daily.ID1)
station_inventory_daily = station_inventory_daily.drop(station_inventory_daily.ID2)
```

Here ,when we join the file together ,usually it will create two same columns .which is used for the condition of join.so here using the drop function to remove these columns.

How expensive do you think it would be to LEFT JOIN all of daily and stations?

It is very slow to LEFT JOIN all of daily and stations, and takes too much memory to get the result.

Result:

```
[32]: station_inventory_daily.show(False)
```

ID	LATITUDE	LONGITUDE	ELEMENT	STATE	NAME		GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE	elements_count	core_elements	other_elements	DATE	ELEMENT	VALUE	MEAS
UREMENT_FLAG	QUALITY_FLAG	SOURCE_FLAG	OBSERVATION_TIME														
IACW00011647	17.1333	-61.7833	19.2		ST JOHNS					IAC	7	5	2	null	null	null	null
IAEM00041217	24.4330	54.6510	26.8		ABU DHABI INTL				41217	AE	14	3	1	null	null	null	null
IAGS00060590	30.5667	2.8667	397.0		EL-GOLEA	IGSN			60590	AG	14	3	1	null	null	null	null
IAGE00147706	36.8000	3.0300	344.0		ALGIERS-BOUZAREAH					AG	3	3	0	null	null	null	null
IAGE00147708	36.7200	4.0500	222.0		TIZI OUZOU				60395	AG	15	14	1	null	null	null	null

only showing top 5 rows

Could you determine if there are any stations in daily that are not in stations without using LEFT JOIN?

Yes, because here got some rows that from the daily part is null, it means some stations in subset of daily are not in stations .

Yes, because here got some rows that from the daily part is null, it means some stations in subset of daily are not in stations .

Analysis

In the section you will answer specific questions about the data using the code that you have developed.

For the some of the tasks in this section you will need to process all of the daily climate summaries, and you may want to increase the resources available to your Spark shell. You may increase your resources up to 4 executors, 2 cores per executor, 4 GB of executor memory, and 4 GB of master memory.

Here when check the original spark shell to get the information about the it .We find that the eis the number of executors; c is the number of cores per executor; w is the worker memory; and m is the master memory.

So, here when we using the spark ,just applyn the new vaule for each character.

The struture of spark :

```
OPTIONS:
-h usage
-e number of executors (default: 2)
-c number of cores per executor (default: 1)
-w worker memory (default: 1)
-m master memory (default: 1)
-n application name (default: USERNAME)

./start-pyspark.sh -e 4 -c 2 -w 4 -m 4
```

Check Spark:

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20180912120645-0988	sl171 (kill)	8	4.0 GB	2018/09/12 12:06:45	sl171@canterbury.ac.nz	RUNNING	5.8 min

Don't forget that you are sharing cluster resources. Keep an eye on Spark using the web user interface (madslinux01:8080), and don't leave a shell running for longer than necessary. If you are asked to share your resources, please respond quickly to let them know how much more time you need.

Q1

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

(a) How many stations are there in total?

```
stations.count()
```

Result:

```
In [33]: stations.count()
Out[33]: 103656
```

103656 stations in total.

How many stations have been active in 2017?

```
inventory_year = inventory.filter((inventory.FIRSTYEAR == 2017) | (inventory.LASTYEAR == 2017))
inventory_year_1 = inventory_year.groupBy('ID2').agg((F.count("ELEMENT").alias("DISTINCT_OTHER_ELEMENT")))
inventory_year_1.count()
```

Result:

```
Out[34]: 37546
```

Using the filter function to select the firstyear or last year is equal to 2017 and then group them by ID and count the number of elements.

So, **37546** stations have been active in 2017.

How many stations are in each of the GCOS Surface Network (GSN), the US Historical Climatology Network (HCN), and the US Climate Reference Network (CRN)?

```
stations.filter(stations.GSN_FLAG == "GSN").count()# Number of GSN
stations.filter(stations.HCN_CRN_FLAG == "HCN").count()# Number of HCN ----@1
stations.filter(stations.HCN_CRN_FLAG == "CRN").count()# Number of CRN ----@2
```

Result:

```
In [36]: stations.filter(stations.GSN_FLAG == "GSN").count()
Out[36]: 991

In [37]: stations.filter(stations.HCN_CRN_FLAG == "HCN").count()
Out[37]: 1218

In [38]: stations.filter(stations.HCN_CRN_FLAG == "CRN").count()
Out[38]: 230
```

991 stations in GCOS Surface Network; **1218** stations in US Historical Climatology Network; **230** stations in Climate Reference Network.

Are there any stations that are in more than one of these networks?

It has three different conditions: stations in both GSN and HCN ; stations in both GSN and CRN; stations in both HCN and CRN.

Stations that are in GSN and HCN:

```
stations.filter(  
    (stations.GSN_FLAG == "GSN")  
    & (stations.HCN_CRN_FLAG == "HCN")).count()
```

Result:

```
In [39]: stations.filter(  
...:         (stations.GSN_FLAG == "GSN")  
...:         & (stations.HCN_CRN_FLAG == "HCN")).count()  
...:  
Out[39]: 14
```

14 stations in both GSN and HCN.

Stations that are in GSN and CRN:

```
stations.filter(  
    (stations.GSN_FLAG == "GSN")  
    & (stations.HCN_CRN_FLAG == "CRN")).count()
```

Result:

```
In [40]: stations.filter(  
...:         (stations.GSN_FLAG == "GSN")  
...:         & (stations.HCN_CRN_FLAG == "CRN")).count()  
...:  
Out[40]: 0
```

No stations in both GSN and CRN.

CHECK whether the Stations are in HCN and CRN:

```
stations1 = stations.withColumn(  
    "HCN_CRN_FLAG1",  
    trim(stations.HCN_CRN_FLAG))  
stations1.filter(stations1.HCN_CRN_FLAG1 == "").count() # Number of (null)----@3  
  
a = stations.filter(stations.HCN_CRN_FLAG != "HCN")  
a.filter(a.HCN_CRN_FLAG != "CRN").count() # Number of (not HCN & not CRN)----@4
```

@3 = @ 4= 102208, so it means that:

The HCN_CRN_FLAG just covers three different kinds of values:

"HCN";"CRN" and "NULL"

no station is in HCN and CRN.

(b) Count the total number of stations in each country

```
country_station_number = station_country.groupBy(
  'NAME1').agg((F.count("ID").alias("Count_of_stations")))
country_station_number.show()
```

Result:

```
+-----+-----+
|      NAME1|Count_of_stations|
+-----+-----+
|Antigua and Barbuda|      2|
|      Albania|      3|
|      Antarctica|     102|
|      Barbados|      1|
|      Botswana|     21|
|      Bolivia|     36|
|      China|    228|
|Cayman Islands [U...|      1|
|      Cuba|      6|
|      Cape Verde|      3|
|Dominican Republic|      5|
|      Fiji|      5|
|      France|     106|
|      Ghana|     18|
|      Hungary|      6|
|      Iceland|     11|
|      India|    3807|
|      Iran|     35|
|Juan De Nova Isla...|      1|
|      Korea, South|     56|
+-----+-----+
only showing top 20 rows
```

and store the output in countries using the withColumnRenamed command.

```
m = country_station_number.withColumnRenamed("NAME1", "NAME")
countries = countries.join(m,countries.NAME1 == m.NAME,"left").drop("NAME1")
```

Result:

```
In [51]: countries.show(5,False)
+-----+-----+
|CODE|NAME|Count of stations|
+-----+-----+
|AL|Albania|3|
|AY|Antarctica|102|
|AC|Antigua and Barbuda|2|
|BF|Bahamas, The|6|
|BB|Barbados|1|
+-----+-----+
only showing top 5 rows
```

Do the same for states and save a copy of each table to your output directory.

```
state_station_number = station_state.groupby('NAME3').agg((F.count("ID").alias("Count of stations")))
state_station_number.show()

n = state_station_number.withColumnRenamed("NAME3", "NAME")
states = states.join(n, states.NAME3 == n.NAME, "left").drop("NAME3")
```

Result:

```
In [54]: states.show(5, False)
+-----+-----+-----+
|CODE|NAME          |Count of stations|
+-----+-----+-----+
|AS  |AMERICAN SAMOA|21               |
|AR  |ARKANSAS      |830              |
|FL  |FLORIDA       |1460             |
|IL  |ILLINOIS      |1630             |
|KY  |KENTUCKY      |764              |
+-----+-----+-----+
only showing top 5 rows
```

(c) How many stations are there in the Southern Hemisphere only?

There are two ways:

Method 1:

Using the output from the (b) (total number of stations in each country), then filter the countries which in the Southern Hemisphere and sum the number of stations. But it is very difficult to filter all of the Southern-Hemisphere countries accurately, maybe form our own mistakes.

Method 2:

Checking the longitude for each station, and filter the stations in Southern Hemisphere. And count the number of them.

```
stations.filter(stations.LONGITUDE < 0).count()
```

Result:

```
In [61]: stations.filter(stations.LONGITUDE < 0).count()
Out[61]: 26063
```

26063 stations are in Southern Hemisphere.

Some of the countries in the database are territories of the United States as indicated by the name of the country. How many stations are there in total in the territories of the United States around the world?

When I check the territories of the united states . each one of them has the same ending with "[United States]" or "[United States] ". So using the endswith to select them, but I am not sure whether it has the white space in the end. So just put the space in the conditions.

```
Territories_of_USA = countries.filter(  
    (countries.NAME.endswith("[United States] "))  
    | (countries.NAME.endswith("[United States]"))  
    | (countries.NAME.endswith("[United States] "))  
    | (countries.NAME.endswith("[United States] "))  
)
```

Result:

CODE	NAME	Count_of_stations
AQ	American Samoa [United States]	21
MQ	Midway Islands [United States]	2
WQ	Wake Island [United States]	1
LQ	Palmyra Atoll [United States]	3
VQ	Virgin Islands [United States]	43
CQ	Northern Mariana Islands [United States]	11
GQ	Guam [United States]	21
RQ	Puerto Rico [United States]	203
JQ	Johnston Atoll [United States]	14

So, total number of stations in Territories of USA is **309**(the sum of each count of station).

Q2

You can create user defined functions in Spark by taking native Python functions and wrapping them using `pyspark.sql.functions.udf` (which can take multiple columns as inputs). You will find this functionality extremely useful.

- (a) Write a Spark function that computes the geographical distance between two stations using their latitude and longitude as arguments. You can test this function by using CROSS JOIN on a small subset of stations to generate a table with two stations in each row.

Note that there is more than one way to compute geographical distance, choose a method that at least takes into account that the earth is spherical.

Function:

```
from pyspark.sql.functions import udf
from math import radians, cos, sin, asin, sqrt

#udf
def haversine(lon1, lat1, lon2, lat2):
    """Calculate the great circle distance between two points on the earth """
    lon1, lat1, lon2, lat2 = map(radians,[lon1,lat1,lon2,lat2])
    # haversine
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) *sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371
    return (c*r *1000)
```

Test:

```
#Test:
station1 = stations.filter(stations.ID == "ACW00011604")
station2 = stations.filter(stations.ID == "AE000041196")
lon1 = -61.7833 #station1.LONGITUDE
lat1 = 17.1167 #station1.LATITUDE
lon2 = 55.5170 #station2.LONGITUDE
lat2 = 25.3330 #station2.LATITUDE
distance = haversine(lon1, lat1, lon2, lat2)
distance
```

Result:

```
In [133]: distance
Out[133]: 12659314.064683918
```

- (b) Apply this function to compute the pairwise distances between all stations in New Zealand, and save the result to your output directory.

Get all the stations in New Zealand:

```
NZ_Stations = stations.filter(stations.COUNTRY_CODE == 'NZ')
NZ_Stations.show(50,False)
```

Result:

```
In [139]: NZ_Stations.show(50,False)
```

ID	LATITUDE	LONGITUDE	ELEMENT	STATE	NAME	GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE
NZ000093012	-35.1000	173.2670	54.0		KAITIA			93119	NZ
NZ000093292	-38.6500	177.9830	5.0		GISBORNE AERODROME	GSN		93292	NZ
NZ000093417	-40.9000	174.9830	7.0		PARAPARAUMU AWS	GSN		93420	NZ
NZ000093844	-46.4170	168.3330	2.0		INVERCARGILL AIRPOR	GSN		93845	NZ
NZ000093994	-29.2500	-177.9170	49.0		RAOUL ISL/KERMADEC			93997	NZ
NZ000933090	-39.0170	174.1830	32.0		NEW PLYMOUTH AWS	GSN		93309	NZ
NZ000936150	-42.7170	170.9830	40.0		HOKITIKA AERODROME			93781	NZ
NZ000937470	-44.5170	169.9000	488.0		TARA HILLS	GSN		93747	NZ
NZ000939450	-52.5500	169.1670	19.0		CAMPBELL ISLAND AWS	GSN		93947	NZ
NZ000939870	-43.9500	-176.5670	49.0		CHATHAM ISLANDS AWS			93987	NZ
NZM00093110	-37.0000	174.8000	7.0		AUCKLAND AERO AWS			93110	NZ
NZM00093439	-41.3330	174.8000	12.0		WELLINGTON AERO AWS			93439	NZ
NZM00093678	-42.4170	173.7000	101.0		KAIKOURA			93678	NZ
NZM00093781	-43.4890	172.5320	37.5		CHRISTCHURCH INTL			93781	NZ
NZM00093929	-50.4830	166.3000	40.0		ENDERBY ISLAND AWS			93929	NZ

What two stations are the geographically closest in New Zealand?

```
LATITUDE_LONGITUDE = [
    (-35.1000,173.2670),
    (-38.6500,177.9830),
    (-40.9000,174.9830),
    (-46.4170,168.3330),
    (-29.2500,-177.9170),
    (-39.0170,174.1830),
    (-42.7170,170.9830),
    (-44.5170,169.9000),
    (-52.5500,169.1670),
    (-43.9500,-176.5670),
    (-37.0000,174.8000),
    (-41.3330,174.8000),
    (-42.4170,173.7000),
    (-43.4890,172.5320),
    (-50.4830,166.3000)
]

result = []
for (lat1,lon1) in LATITUDE_LONGITUDE:
    for (lat2,lon2) in LATITUDE_LONGITUDE:
        distance = haversine(lon1, lat1, lon2, lat2)
        result.append(distance)

result2 = []
for i in result:
    if i not in result2:
        result2.append(i)
result2.remove(0)

a = result.index(min(result2))

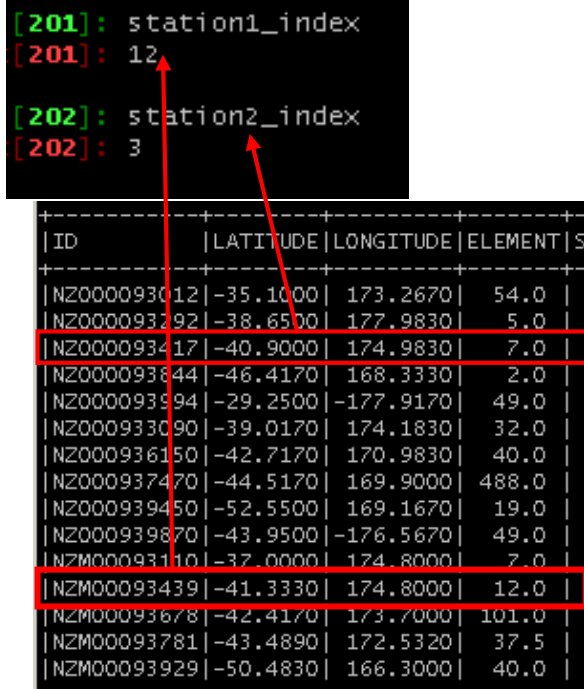
b = len(LATITUDE_LONGITUDE)

station1_index = int((a+1)/b + 1)####the index of row (station1) in NZ_station
station2_index = (a+1)%b ####the index of row (station2) in NZ_station
```

Using the python, to get all the distance for each two stations ,and get the minimum one of them .Check the position of it in final result.according to the position of it to check the index of these two stations in the original dataframe.

Result:

```
[201]: station1_index  
[201]: 12  
  
[202]: station2_index  
[202]: 3
```



ID	LATITUDE	LONGITUDE	ELEMENT	S
NZ000093412	-35.1000	173.2670	54.0	
NZ000093492	-38.6500	177.9830	5.0	
NZ000093417	-40.9000	174.9830	7.0	
NZ000093444	-46.4170	168.3330	2.0	
NZ000093494	-29.2500	-177.9170	49.0	
NZ000093490	-39.0170	174.1830	32.0	
NZ000093450	-42.7170	170.9830	40.0	
NZ000093470	-44.5170	169.9000	488.0	
NZ000093450	-52.5500	169.1670	19.0	
NZ000093470	-43.9500	-176.5670	49.0	
NZM00093410	-37.0000	174.8000	7.0	
NZM00093439	-41.3330	174.8000	12.0	
NZM00093478	-42.4170	173.7000	101.0	
NZM00093481	-43.4890	172.5320	37.5	
NZM00093429	-50.4830	166.3000	40.0	

The geographically _closest two stations in New Zealand :

NZM00093439 ; NZ000093417

Q3

Next you will start exploring all of the daily climate summaries in more detail. In order to know how efficiently you can load and apply transformations to daily, you need to first understand the level of parallelism that you can achieve for the specific structure and format of daily.

- (a) Recall the hdfs commands that you used to explore the data in Processing Q1. You would have used

```
hdfs dfs -ls [path]
hdfs dfs -du [path]
```

to determine the size of files under a specific path in HDFS.

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

Result:

```
[s11171@canterbury.ac.nz@madslinux01 ~]$ hdfs dfs -du -h hdfs:///data/ghcnd
3.6 K    28.7 K    hdfs:///data/ghcnd/countries
13.4 G   106.9 G   hdfs:///data/ghcnd/daily
26.1 M   209.1 M   hdfs:///data/ghcnd/inventory
1.1 K    8.5 K      hdfs:///data/ghcnd/states
8.5 M    68.0 M      hdfs:///data/ghcnd/stations
```

Use the following command

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

```
[s11171@canterbury.ac.nz@madslinux01 ~]$ hdfs getconf -confkey "dfs.blocksize"
134217728
```

to determine the default blocksize of HDFS.

How many blocks are required for the daily climate summaries for the year 2017?

What about the year 2010?

```
hdfs dfs -du -h hdfs:///data/ghcnd/daily/2017.csv.gz
hdfs dfs -du -h hdfs:///data/ghcnd/daily/2010.csv.gz
```

Result:

```
[s11171@canterbury.ac.nz@madslinux01 ~]$ hdfs dfs -du -h hdfs:///data/ghcnd/daily/2017.csv.gz
119.5 M   955.6 M   hdfs:///data/ghcnd/daily/2017.csv.gz
[s11171@canterbury.ac.nz@madslinux01 ~]$ hdfs dfs -du -h hdfs:///data/ghcnd/daily/2010.csv.gz
197.6 M   1.5 G     hdfs:///data/ghcnd/daily/2010.csv.gz
```

The size of daily IN 2017 (119.5M) is smaller than the default blocksize(134217728), so just **1 block** is enough.

The size of daily IN 2010 (197.6G) is bigger than the default blocksize(134217728), so needs **2 blocks**.

What are the individual block sizes for the year 2010?

You will need to use another hdfs command.

```
hdfs fsck hdfs:///data/ghcnd/daily/2010.csv.gz
```

Result:

```
Status: HEALTHY
Number of data-nodes: 16
Number of racks:      1
Total dirs:           0
Total symlinks:        0

Replicated Blocks:
Total size: 207181730 B
Total files: 1
Total blocks (validated): 2 (avg. block size 103590865 B)
Minimally replicated blocks: 2 (100.0 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks: 0 (0.0 %)
Mis-replicated blocks: 0 (0.0 %)
Default replication factor: 8
Average block replication: 8.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0 (0.0 %)

Erasure Coded Block Groups:
Total size: 0 B
Total files: 0
Total block groups (validated): 0
Minimally erasure-coded block groups: 0
Over-erasure-coded block groups: 0
Under-erasure-coded block groups: 0
Unsatisfactory placement block groups: 0
Average block group size: 0.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0
FSCK ended at Fri Sep 14 14:08:29 NZST 2018 in 1 milliseconds
```

Based on these results, is it possible for Spark to load and apply transformations in parallel for the year 2017? What about the year 2010?

- (b) Load and count the number of rows in daily for each of the years 2010 and 2017.
How many tasks were executed by each stage of each job?

```
daily = (
    sqlContext
    .read
    .format("com.databricks.spark.csv")
    .option("header", "false")
    .schema(schema_daily)
    .load("hdfs:///data/ghcnd/daily/2010.csv.gz"))

daily.count()
```

Result:

```
36946080
```

The number of rows in daily for each of the year 2010 is 36946080.

```
daily = (
    sqlContext
    .read
    .format("com.databricks.spark.csv")
    .option("header", "false")
    .schema(schema_daily)
    .load("hdfs:///data/ghcnd/daily/2017.csv.gz"))

daily.count()
```

Result:

21904999

The number of rows in daily for each of the year 2010 is **21904999**.

Total number of tasks completed by each executor:

Job 0 is to load and count the number of year 2010.

Job 1 is to load and count the number of year 2007.

1	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2018/09/11 01:04:12	32 s	2/2	2/2
0	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2018/09/11 00:57:05	49 s	2/2	2/2

Number of tasks executed by each stage of each job:

Here, for job 0, it has been separated into two stages:

Stage0: load the data for year 2010.

Stage1: count the number of rows for year 2010.

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

- ▶ Event Timeline
- ▶ DAG Visualization

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	count at NativeMethodAccessorImpl.java:0	2018/09/11 00:57:53	0.2 s	1/1			59.0 B	
0	count at NativeMethodAccessorImpl.java:0	2018/09/11 00:57:05	48 s	1/1	197.6 MB			59.0 B

Job 1 also does the same thing as job 0.

Details for Job 1

Status: SUCCEEDED
Completed Stages: 2

- ▶ Event Timeline
- ▶ DAG Visualization

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	count at NativeMethodAccessorImpl.java:0	2018/09/11 01:04:44	0.2 s	1/1			59.0 B	
2	count at NativeMethodAccessorImpl.java:0	2018/09/11 01:04:12	32 s	1/1	119.5 MB			59.0 B

You can check this by using your application console. which you can find in the web user interface (madslinux01:8080). You can either determine the number of tasks executed by each stage of each job or determine the total number of tasks completed by each executor after the job has completed.

Did the number of tasks executed correspond to the number of blocks in each input?
Yes.

(c) Load and count the number of rows in daily from 2010 to 2015.

Note that you can use any regular expressions in the path argument of the read command.

Now how many tasks were executed by each stage, and how does this number correspond to your input?

Find out how Spark partitions input files that are compressed.

```
daily = (
    sqlContext
    .read
    .format("com.databricks.spark.csv")
    .option("header", "false")
    .schema(schema_daily)
    .load("hdfs:///data/ghcnd/daily/201[0-5].csv.gz"))

daily.count()
```

Result:

207716098

Here using the regular expressions "201[0-5]" to stand for the year 2010 to 2015.
And the number of rows in daily from 2010 to 2015 is **207716098**.

Total number of tasks completed by each executor:

For this Job ,total 7 tasks completed .

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at NativeMethodAccessorImpl.java:0	2018/09/11 01:08:57	2.3 min	2/2	7/7

Number of tasks executed by each stage of each job:

When seperating it , we find that 6 task is for load the data, and one task is for count.

Details for Job 2

Status: SUCCEEDED
Completed Stages: 2

► Event Timeline
► DAG Visualization

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	count at NativeMethodAccessorImpl.java:0	+details	2018/09/11 01:11:14	0.1 s	1/1		354.0 B	
4	count at NativeMethodAccessorImpl.java:0	+details	2018/09/11 01:08:57	2.3 min	6/6	1130.2 MB		354.0 B

- (d) Based on parts (b) and (c), what level of parallelism can you achieve when loading and applying transformations to daily? Can you think of any way you could increase this level of parallelism either in Spark or by additional preprocessing?

Here , I have check the meaning of level of parallelism:

Level of Parallelism ----- Clusters will not be fully utilized unless you set the level of parallelism for each operation high enough. Spark automatically sets the number of “map” tasks to run on each file according to its size (though you can control it through optional parameters to `SparkContext.textFile`, etc), and for distributed “reduce” operations, such as `groupByKey` and `reduceByKey`, it uses the largest parent RDD’s number of partitions. You can pass the level of parallelism as a second argument (see the `spark.PairRDDFunctions` documentation), or set the config property `spark.default.parallelism` to change the default. In general, we recommend 2-3 tasks per CPU core in your cluster.

In spark , it has two different types of operators: **transformation** and **action**.

So , for loading and applying transformations to daily is the level that Spark automatically **sets the number of “map” tasks to run on each file according to its size**.

So , here we can set the **spark.default.parallelism** to reduce tasks ,and it can save time and increase the level of parallelism.

Q4

All of the data stored in HDFS under `hdfs:///data/ghcnd` has a replication factor of 4 and is available locally on every node in our cluster. As such you will always be able to load and apply transformations to multiple years of daily in parallel.

Again, you may want to use only part of the daily climate summaries while you are still developing your code so that you can use fewer resources to get preliminary results without waiting all day.

- (a) Count the number of rows in daily.

Note that this will take a while if you are only using 2 executors and 1 core per executor, and that the amount of driver and executor memory should not matter unless you actually try to cache or collect all of daily. You should never try to cache or collect all of daily.

```

schema_daily = StructType([
    StructField("ID1", StringType(), True),
    StructField("DATE", StringType(), True),
    StructField("ELEMENT", StringType(), True),
    StructField("VALUE", FloatType(), True),
    StructField("MEASUREMENT FLAG", StringType(), True),
    StructField("QUALITY FLAG", StringType(), True),
    StructField("SOURCE FLAG", StringType(), True),
    StructField("OBSERVATION TIME", StringType(), True)
])

daily = (
    sqlContext
    .read
    .format("com.databricks.spark.csv")
    .option("header", "false")
    .schema(schema_daily)
    .load("hdfs://data/ghcnd/daily")

daily.count()

```

Result:

```
2624027105
```

The number of rows in daily is **2624027105**.

- (b) Filter daily using the filter command to obtain the subset of observations containing the five core elements described in inventory.

```

core_elements = ["PRCP", "SNOW", "SNWD", "TMAX", "TMIN"]

daily_core_1 = daily.filter(daily.ELEMENT == "PRCP").distinct()
daily_core_2 = daily.filter(daily.ELEMENT == "SNOW").distinct()
daily_core_3 = daily.filter(daily.ELEMENT == "SNWD").distinct()
daily_core_4 = daily.filter(daily.ELEMENT == "TMAX").distinct()
daily_core_5 = daily.filter(daily.ELEMENT == "TMIN").distinct()
daily_core = daily_core_1.unionAll(daily_core_2)
daily_core = daily_core.unionAll(daily_core_3)
daily_core = daily_core.unionAll(daily_core_4)
daily_core = daily_core.unionAll(daily_core_5)
daily_core.show(5, False)

```

Result:

ID1	DATE	ELEMENT	VALUE	MEASUREMENT FLAG	QUALITY FLAG	SOURCE FLAG	OBSERVATION TIME
UK000047811	18730223	PRCP	-6660.0	null	X	E	null
GM000003218	19810210	PRCP	-999.0	null	X	I	null
GM000003218	19810220	PRCP	-999.0	null	X	I	null
GM000003218	19810227	PRCP	-999.0	null	X	I	null
GM000003218	19810301	PRCP	-999.0	null	X	I	null

How many observations are there for each of the five core elements?

```

daily_core_all = daily_core.groupBy('ELEMENT').agg((F.count("ELEMENT").alias("DISTINCT_CORE_ELEMENT")))

daily_core_all.show(5, False)

```

Result:

ELEMENT	DISTINCT_CORE_ELEMENT
SNOW	322003304
SNWD	261455306
PRCP	918490401
TMIN	360656728
TMAX	362528096

Which element has the most observations? Is this element consistent with Processing Q3?

From the result ,we can find that the “PRCP” has the most observations. And it is consistent with Processing Q3.

(d)Filter daily to obtain all observations of TMIN and TMAX for all stations in New Zealand, and save the result to your output directory.

```
dail_tmax = daily.filter(daily.ELEMENT == "TMAX").distinct()
dail_tmin = daily.filter(daily.ELEMENT == "TMIN").distinct()
daily_tmax_tmin = dail_tmax.unionAll(dail_tmin)
daily_tmax_tmin_nz = daily_tmax_tmin.filter(daily_tmax_tmin.ID1.startswith("NZ"))
daily_tmax_tmin_nz.show(5,False)
```

Save it from pyspark to output directory in HDFS.

```
daily_tmax_tmin_nz.select().write.save("hdfs:///user/sli171/outputs/ghcnd/daily_tmax_tmin_nz",format="csv")
```

Result:

ID1	DATE	ELEMENT	VALUE	MEASUREMENT FLAG	QUALITY FLAG	SOURCE FLAG	OBSERVATION TIME
NZM00093110	20090104	TMAX	210.0	null	null	S	null
NZ000933090	20090105	TMAX	204.0	null	null	S	null
NZ000936150	20090107	TMAX	327.0	null	null	S	null
NZ000933090	20090109	TMAX	240.0	null	null	S	null
NZM00093439	20090110	TMAX	202.0	null	null	S	null

only showing top 5 rows

How many observations are there?

```
daily_tmax_tmin_nz.count()
```

Result:

```
447017
```

how many years are covered by the observations?

```
daily_year_nz = daily_tmax_tmin_nz.select(daily_tmax_tmin_nz.ID1,
                                           (daily_tmax_tmin_nz.DATE[1:4]).alias("YEAR")
                                           )
daily_year_nz.show(5, False)
```

Result:

```
+-----+-----+
|ID1      |YEAR|
+-----+-----+
|NZM00093110|2009|
|NZ000933090|2009|
|NZ000936150|2009|
|NZ000933090|2009|
|NZM00093439|2009|
+-----+-----+
only showing top 5 rows
```

```
year_summary = daily_year_nz.groupBy('YEAR').agg((F.count("YEAR").alias("DISTINCT_YEAR")))
year_summary.show(5, False)
year_summary.count()
```

Result:

```
|YEAR|DISTINCT_YEAR|
+---+-----+
|2002|7838          |
|1988|7846          |
|1955|3650          |
|1961|4732          |
|1996|7290          |
+---+-----+
only showing top 5 rows

Out[39]: 78
```

78 years are covered by the observations.

Use `hdfs dfs -copyToLocal` to copy the output from HDFS to your local home directory, and count the number of rows in the part files using the `wc -l` bash command. This should match the number of observations that you counted using Spark.

Save data from spark to HDFS , and copy it from HDFS to local home directory.

```
##save:

daily_tmax_tmin_nz.write.format("csv").save("hdfs:///user/sli171/outputs/ghcnd/daily_tmax_tmin_nz")

##Copy to local:

hdfs dfs -copyToLocal hdfs:///user/sli171/outputs/ghcnd/daily_tmax_tmin_nz
```

Plot the time series of TMIN and TMAX on the same axis for each station in New Zealand using Python, R, or any other programming language you know well. Also, plot the average time series for the entire country.

- (e) Group the precipitation observations by year and country.
Compute the average rainfall in each year for each country, and save this result to your output directory.

```
daily_prdp = daily.filter(daily.ELEMENT == "PRCP").distinct()

daily_prdp_year_country = daily_prdp.select(daily_prdp.ID1,
                                             (daily_prdp.ID1[1:2]).alias("COUNTRY_CODE"),
                                             (daily_prdp.DATE[1:4]).alias("YEAR"),
                                             daily_prdp.VALUE
                                             )

daily_prdp_year_country.show(5,False)

daily_prdp_country = daily_prdp_year_country.groupBy('COUNTRY_CODE','YEAR').agg((F.mean("VALUE").alias("Average_Value")))
daily_prdp_country.show(5,False)
```

Result:

```
+-----+-----+-----+
|COUNTRY_CODE|YEAR|Average_Value|
+-----+-----+-----+
|TS          |1921|9.369272237196766|
|TS          |1900|4.405432595573441|
|TS          |1905|7.260869565217392|
|TS          |1935|10.421582234559335|
|US          |1905|27.541356091557624|
+-----+-----+-----+
only showing top 5 rows
```

Which country has the highest average rainfall in a single year across the entire dataset?

From the result ,we can find that **Colombia** has the highest average rainfall in a single year.

Find an elegant way to plot the cumulative rainfall for each country using Python, R, or any other programming language you know well. There are many ways to do this in Python and R specifically, such as using a choropleth to color a map according to average rainfall.