Arijit Bhattacharjee
Student ID 26649839

Processing

1.

a) Directory tree to represent how the data is structured
Hdfs:
```
|- - - - data
|            |- - - - ghcnd
|            |           |- - - - daily
|            |           |           |- - - - 1763.csv
|            |           |           |- - - - 1764.csv
|            |           |           :             :
|            |           |           :             :
|            |           |           |- - - - 2022.csv
|            |           |
|            |           |- - - - countries.txt
|            |           |- - - - inventory.txt
|            |           |- - - - states.txt
|            |           |- - - - stations.txt
```

b) *hdfs dfs -ls -h /data/ghcnd/daily* command is used to explore daily.
The starting year is 1763 and the ending year is 2022. So, daily contains records of 260 years.
The file size is comparatively much smaller (in KB) for starting years and the size gradually keeps on increasing over time and reaching a maximum size of 221 MB in 2010. Thereafter we observe a slight reduction in file size and file size reaching around 150 MB from 2018 onwards. The records for 2022 are incomplete hence the file size is comparatively lower than previous years. For example

Year	Filesize
1763	3.3 K
2010	221 M
2022	84 M

c) *hdfs dfs -du -h /data/ghcnd/* command is used to explore the size of data.

```
15.8 G  126.1 G  /data/ghcnd/daily
3.6 K   28.6 K   /data/ghcnd/ghcnd-countries.txt
31.8 M  254.7 M  /data/ghcnd/ghcnd-inventory.txt
1.1 K   8.5 K    /data/ghcnd/ghcnd-states.txt
10.0 M  80.1 M   /data/ghcnd/ghcnd-stations.txt
```

The total size of all of the data is around 15.8 GB. The size of daily is more than 99.9% of the total size.

2.

a) Schema for daily

schema_daily = StructType([
StructField("ID", StringType(), True), #Station code
StructField("DATE", DateType(), True), #Observation date formatted as YYYYMMDD
StructField("ELEMENT", StringType(), True), #Element type indicator
StructField("VALUE", DoubleType(), True), #Data value for ELEMENT
StructField("MEASUREMENT_FLAG", StringType(), True), # Measurement Flag
StructField("QUALITY_FLAG", StringType(), True), #Quality Flag
StructField("SOURCE_FLAG", StringType(), True), #Source Flag
StructField("OBSERVATION_TIME", TimestampType(), True), #Observation time formatted as HHMM
])

b) After loading a subset of daily/2022 we can observe that the fields ID, ELEMENT, VALUE, MEASUREMENT_FLAG and SOURCE_FLAG are loaded with expected values. However, we observe all values are null for DATE, SOURCE_FLAG and OBSERVATION_TIME. These could be because of missing values or the defined datatype being not compatible with the data.
However, by changing the datatype of DATE to StringType() we can see that the values for the DATE column are loaded properly.

c) Stations, states, countries and inventory are loaded into spark using the spark.read.text() followed by parsing the fixed width text formatting using substr(), casting appropriate column as DoubleType() or IntegerType(), and renaming the columns using alias().
Number of rows in each metadata table
Stations = 122047
States = 74
Countries = 219
Inventory = 725754

We also define a function clean_metadata() to carry out cleaning operations to make the text data more uniform and pass the dataframes of metadata through it. This would help us with counting null values.

113961 stations don't have a WMO ID.

3.

a) Extract country code and store the output as COUNTRY_CODE
stations_renamed = stations_renamed.withColumn("COUNTRY_CODE", F.col("ID")[0:2])

ID	LATITUDE	LONGITUDE	ELEVATION	STATE_CODE	STATION_NAME	GSN_FLAG	HCN_CRN_FLAG	WMO_ID	COUNTRY_CODE
ACW00011604	17.1167	-61.7833	10.1	null	ST JOHNS COOLIDGE...	null	null	null	AC
ACW00011647	17.1333	-61.7833	19.2	null	ST JOHNS	null	null	null	AC
AE000041196	25.333	55.517	34.0	null	SHARJAH INTER. AIRP	GSN	null	41196	AE

b) Left join countries on stations. All rows from the stations are returned regardless of match found on the countries dataset. When the join expression doesn't match, it

assigns null for that record and drops records from countries where the match is not found.
Countries_renamed and stations_renamed are renamed tables for countries and stations respectively.

```
stations_with_country_name = (
  stations_renamed
  .join(
    countries_renamed,
    on="COUNTRY_CODE",
    how="left"
  )
)
stations_with_country_name.show(10)
```

c) Left join states on stations.

```
stations_with_country_and_state_name = (
  stations_with_country_name
  .join(
    states_renamed,
    on="STATE_CODE",
    how="left"
  )
)
stations_with_country_and_state_name.show(10)
```

However, it is observed that few state codes are outside the USand in other countries like Canada. We have retained these rows.

```
stations_with_country_and_state_name.select('STATE_CODE','STATE_NAME','COUNTRY_CODE','COUNTRY_NAME').where((F.col('COUNTRY_CODE')!='US') & (F.col('STATE_NAME')!='null')).distinct().show(5)
```

d) For this part of the question we will use collect_set() on “ELEMENT” and then use array functions on the result to determine counts of CORE_ELEMENTS and OTHER_ELEMENTS in a single select. A temporary table named ‘temp’ is created.

We group by ‘ID’ i.e station id and aggregate on -
i) ‘ELEMENTS’ collected as list to give all elements
ii) ‘ELEMENTS’ collected as set to give the distinct elements
iii) minimum of ‘FIRSTYEAR’ to give start year
iv) maximum of ‘LASTYEAR’ to give end year

Then we select and operate on the columns for values
i) F.col(“ELEMENTS_DISTINCT”) is an array type column. F.size returns the length of the array or map stored in the column. It will give the count of the distinct elements when operated on the collected set for each row.

ii) Earlier we had defined core elements as a list that can be iterated upon (defined separately so that it’s easily scalable). We will be using F.lit() to create an array of constants and F.array to create a new array column. Using these commands we will create a constant column of values ‘PRCP’,’TMIN’ etc. and select all of them into an array, which would be an array of values ‘PRCP’,’TMIN’ etc.
iii) F.array_intersect(col1,col2) returns an array of the elements in the intersection of col1 and col2, without duplicates - we will use it for core elements where col1 is distinct elements and col2 is obtained from step ii) above.
iv) F.array_except(col1,col2) returns an array of the elements in col1 but not in col2, without duplicates - we use it for other elements. Same values of col1 and col2 used as step iii) above but this time it will give other elements.
v) We use F.size to return the length of the array stored in the respective columns (obtained from above steps) to get the number of core and other elements.

ID	START_YEAR	END_YEAR	ELEMENTS	ELEMENTS_DISTINCT	NUM_ELEMENTS	CORE_ELEMENTS	NUM_CORE_ELEMENTS
ACW00011604	1949	1949	[TMAX, TMIN, PRCP...	[TMAX, TMIN, WSFG...	11	[TMAX, TMIN, PRCP...	5
ACW00011647	1957	1970	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	7	[TMAX, TMIN, PRCP...	5
AE000041196	1944	2022	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	4	[TMAX, TMIN, PRCP]	3
AEM00041194	1983	2022	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	4	[TMAX, TMIN, PRCP]	3
AEM00041217	1983	2022	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	4	[TMAX, TMIN, PRCP]	3
AEM00041218	1994	2022	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	4	[TMAX, TMIN, PRCP]	3
AF000040930	1973	1992	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	5	[TMAX, TMIN, PRCP...	4
AFM00040938	1973	2021	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	5	[TMAX, TMIN, PRCP...	4
AFM00040948	1966	2021	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	5	[TMAX, TMIN, PRCP...	4
AFM00040990	1973	2020	[TMAX, TMIN, PRCP...	[TMAX, TMIN, PRCP...	5	[TMAX, TMIN, PRCP...	4

only showing top 10 rows

OTHER_ELEMENTS	NUM_OTHER_ELEMENTS
[WSFG, WDFG, PGTM...	6
[WT16, WT03]	2
[TAVG]	1
[TAVG]	1
[TAVG]	1
[TAVG]	1
[TAVG]	1
[TAVG]	1
[TAVG]	1
[TAVG]	1

As these are off the shelf functions, spark can optimise these. So computation efficiency should not be an issue here.

Station wise data for first and last year is collated in inventory.parquet folder and shared with supporting documents. As there are records for 122010 stations, it is not feasible to display the data here.

Each station has collected 725754 different elements overall.

Number of core elements that each station has collected overall is 336814 and the number of other elements is 388940.

20300 stations collected all 5 core elements.

16159 stations collected only precipitation and no other element

e) Left join to create the enriched stations data (Name of table: stations_with_country_state_name_inventory) and save the data using following commands
#Copy enriched stations data to hdfs
stations_with_country_state_name_inventory.write.parquet("/user/abh89/spark/outputs/ghcnd/stations_enriched.parquet")
#Copy from hdfs to local
!hdfs dfs -copyToLocal /user/abh89/spark/outputs/ghcnd/stations_enriched.parquet ~/spark/outputs/ghcnd/stations_enriched.parquet

f) Left join
While looking for any stations in the subset of daily that are not in enriched stations at all, the idea is to look for null values in START_YEAR and END_YEAR in the joined table (as this is not an expected missing value for a match in left join, if there is no match then there would be a gap). As there is no null value we can say that there are no stations in the subset of daily that are not in stations_enriched at all.

Broadcast join can be effective here as the stations data is small. However we are not sure that stations would be bounded in size over time. Also we would need to use groupby and aggregate on many occasions so shuffling the data using shuffle join makes sense. Shuffle join would be expensive as the daily data is huge. Stations metadata has to be copied to every row of daily.

Left anti join will return non matched records thereby helping us to determine if there are any stations in daily that are not in stations.

Analysis

1.

a) Total number of stations = 122047
Stations active in 2021 = 42588 (Start year not 2022 and end year > 2020)
Stations in GSN = 991
Stations in HCN = 1218
Stations in CRN = 0
Stations in GSN and HCN = 14

b) Groupby COUNTRY_CODE and aggregate on count of station ids i.e ID to create a temporary table. Join temporary table and countries and update column name using the withColumnRenamed command.
Save in output directory using the following commands
countries_updated.repartition(1).write.csv("/user/abh89/spark/outputs/ghcnd/countries.csv")
!hdfs dfs -copyToLocal /user/abh89/spark/outputs/ghcnd/countries.csv ~/spark/outputs/ghcnd/countries.csv

Groupby STATE_CODE and aggregate on count of station ids i.e ID to create a temporary table. Join temporary table and states and update column name using the withColumnRenamed command.
statess_updated.repartition(1).write.csv("/user/abh89/spark/outputs/ghcnd/states.csv")
!hdfs dfs -copyToLocal /user/abh89/spark/outputs/ghcnd/states.csv ~/spark/outputs/ghcnd/states.csv

c) Stations in southern hemisphere = 25337 (Latitude < 0)

First step is to filter the State code and country code of US territories from stations data. We select rows where the country code is not US and the state name is not null and the country name contains 'United States' (to exclude countries like Canada etc.).

STATE_CODE	STATE_NAME	COUNTRY_CODE	COUNTRY_NAME
VI	VIRGIN ISLANDS	VQ	Virgin Islands [U...
UM	U.S. MINOR OUTLYI...	LQ	Palmyra Atoll [Un...
UM	U.S. MINOR OUTLYI...	JQ	Johnston Atoll [U...
UM	U.S. MINOR OUTLYI...	WQ	Wake Island [Unit...
AS	AMERICAN SAMOA	AQ	American Samoa [U...
GU	GUAM	GQ	Guam [United States]
UM	U.S. MINOR OUTLYI...	MQ	Midway Islands [U...
MP	NORTHERN MARIANA ...	CQ	Northern Mariana ...
PR	PUERTO RICO	RQ	Puerto Rico [Unit...

Then we join the number of stations on COUNTRY_CODE (obtained from part b above).

Number of stations in US territories is 354

2.

a) We will be using Haversine distance to calculate geographical distances. It assumes the shape of the Earth is a sphere.[1]
We take stations in New Zealand as a small subset of data which is filtered from the enriched stations table. Next we select only the required columns like station id, station name, latitude and longitude.

Then we create a standard python function, where we use the radius of the earth as 6371km and return the absolute value of the distance rounded to 2dp. Next we create a udf to use it on our spark dataframe.
Then we took NZ stations metadata and cross join it with itself to allow for column operations, renaming columns in the process.

STATION_ID_A	STATION_NAME_A	LATITUDE_A	LONGITUDE_A	STATION_ID_B	STATION_NAME_B	LATITUDE_B	LONGITUDE_B
NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	NZM00093110	AUCKLAND AERO AWS	-37.0	174.8
NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	NZ000936150	HOKITIKA AERODROME	-42.717	170.983
NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	NZM00093678	KAIKOURA	-42.417	173.7
NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	NZ000093844	INVERCARGILL AIRPOR	-46.417	168.333
NZM00093110	AUCKLAND AERO AWS	-37.0	174.8	NZ000093994	RAOUL ISL/KERMADEC	-29.25	-177.917

only showing top 5 rows

b) We clean repeated rows and then apply our udf to NZ station pairs to add a new column ABS_DISTANCE. Finally we cast the ABS_DISTANCE column as a double.

STATION_ID_A	STATION_NAME_A	LATITUDE_A	LONGITUDE_A	STATION_ID_B	STATION_NAME_B	LATITUDE_B	LONGITUDE_B	ABS_DISTANCE
NZ000093417	PARAPARAUMU AWS	-40.9	174.983	NZM00093439	WELLINGTON AERO AWS	-41.333	174.8	50.53
NZM00093439	WELLINGTON AERO AWS	-41.333	174.8	NZ000093417	PARAPARAUMU AWS	-40.9	174.983	50.53
NZM00093678	KAIKOURA	-42.417	173.7	NZM00093439	WELLINGTON AERO AWS	-41.333	174.8	151.07
NZM00093439	WELLINGTON AERO AWS	-41.333	174.8	NZM00093678	KAIKOURA	-42.417	173.7	151.07
NZM00093781	CHRISTCHURCH INTL	-43.489	172.532	NZ000936150	HOKITIKA AERODROME	-42.717	170.983	152.26
NZ000936150	HOKITIKA AERODROME	-42.717	170.983	NZM00093781	CHRISTCHURCH INTL	-43.489	172.532	152.26
NZM00093781	CHRISTCHURCH INTL	-43.489	172.532	NZM00093678	KAIKOURA	-42.417	173.7	152.46
NZM00093678	KAIKOURA	-42.417	173.7	NZM00093781	CHRISTCHURCH INTL	-43.489	172.532	152.46
NZM00093678	KAIKOURA	-42.417	173.7	NZ000093417	PARAPARAUMU AWS	-40.9	174.983	199.53
NZ000093417	PARAPARAUMU AWS	-40.9	174.983	NZM00093678	KAIKOURA	-42.417	173.7	199.53

only showing top 10 rows

PARAPARAUMU AWS and WELLINGTON AERO AWS are geographically closest stations in New Zealand.

The output is saved to hdfs
nz_station_distance.write.csv("/user/abh89/spark/outputs/ghcnd/newzealand_stations_distance.csv")
#Copy to local
!hdfs dfs -copyToLocal /user/abh89/spark/outputs/ghcnd/newzealand_stations_distance.csv ~/spark/outputs/ghcnd/newzealand_stations_distance.csv

3.

a) Default block size of hdfs is 128MB
Number of blocks required for 2022 is 1
Number of blocks required for 2021 is 2
2021 has average block size of 79799197 B or 76.1MB

Spark can load and apply transformation in parallel for 2022 as there is only a single block.

However, spark can't load and apply transformation in parallel for 2021 because there are multiple blocks and gzip compressed data is not splittable. During the load the blocks need to be collected on one executor.

b)

```
2021 count = 35917254
2022 count = 19648456

2021 num partitions = 1
2022 num partitions = 1
```

number of task = number of partition

The number of tasks executed corresponds to the number of blocks for 2022 but the number of tasks executed doesn't correspond to the number of blocks for 2021.

c) Total number of count of observation is 303501016 from 2014 to 2022
Number of partitions = 9

years = [2014,2015,2016,2017,2018,2019,2020,2021,2022]
daily_selected = spark.read.csv([f"/data/ghcnd/daily/{year}.csv.gz" for year in years], schema=schema)
print(f"daily_selected count   = {daily_selected.count()}")
print(f"daily_selected num partitions = {daily_selected.rdd.getNumPartitions()}")

Year wise split:

```
2022 count = 19648456
2021 count = 35917254
2020 count = 36167120
2019 count = 35941498
2018 count = 36326971
2017 count = 34854073
2016 count = 35326496
2015 count = 34899014
2014 count = 34420134

2022 num partitions = 1
2021 num partitions = 1
2020 num partitions = 1
2019 num partitions = 1
2018 num partitions = 1
2017 num partitions = 1
2016 num partitions = 1
2015 num partitions = 1
2014 num partitions = 1
```

9 tasks were executed even though multiple years have multiple blocks. One task is executed for every partition. The number of partitions we have is the upper limit on how many tasks we can do in parallel - 9 tasks can be carried out in parallel. So even if we have more executors (upto 32 can be used in our cluster), only 9 tasks can be carried out in parallel and our cluster would be underutilized.

Input files that are compressed e.g gzip compression format is not splittable and each file must be loaded in its entirety by a single executor (to uncompress the data successfully) resulting in one partition per file (each block can't be loaded locally in parallel).

d) Number of files is 260 (1 for every year) so the maximum number of tasks that can run in parallel is 260. However, the number of executors (32 in our case) could limit the maximum number of tasks that can be run in parallel to 32.
We can combine smaller files together (coalesce or repartition) that could increase the number of tasks executed in parallel.

4.

a) Number of rows in daily = 3018826504

b) Filter by iterating over the list of core elements in a for loop.

daily_core = (daily_all
.filter(functools.reduce(operator.or_, [F.col("ELEMENT") == x for x in core_elements]))
.select("ID","DATE","ELEMENT","VALUE","MEASUREMENT_FLAG","QUALITY_FLAG","SOURCE_FLAG","OBSERVATION_TIME"))

For count of each element groupby core element and aggregate on count of ID

```
+------------+----------+
|CORE_ELEMENT|     COUNT|
+------------+----------+
|        SNOW| 344268930|
|        SNWD| 290998195|
|        TMIN| 445687425|
|        PRCP|1048156273|
|        TMAX| 447084093|
+------------+----------+
```

PRCP has the most observations

c) We will filter observations containing only TMIN and TMAX from daily. We will use collect_set() on "ELEMENT" and then use array functions on the result to determine counts of T_MAX and T_MIN in a single select.
The idea behind this is that each row will now have a ID, Date and count of TMIN and TMAX (each id and date group will have a maximum of 1 count of TMAX/TMIN and a minimum of 0).
For observations of TMIN that do not have a corresponding observation of TMAX, we equate ('NUM_TMAX' == 0)& ('NUM_TMIN' == 1).

8848299 observations of TMIN do not have a corresponding observation of TMAX
27678 unique stations contributed to these observations.

d) We know that the first two characters of the station code denote the country code, so we will extract the country code directly from the daily code table(subset of core element observation in daily) rather than using joins. We filter by element TMAX and TMIN and country code as NZ to create the required table. We also divide the value by 10 to arrive at temperature in degrees celsius. The date column is also split into year, month and day before saving the table.
Number of observations = 474654 and 83 years (1940-2022) are covered by the observations.

```
+------------+----------+--------+------+
|COUNTRY_CODE|START_YEAR|END_YEAR| COUNT|
+------------+----------+--------+------+
|          NZ|      1940|    2022|474654|
+------------+----------+--------+------+
```

Save in output directory using the following commands
daily_minmax_subset.write.csv("/user/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv")
!hdfs dfs -copyToLocal /user/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv ~/spark/outputs/ghcnd/newzealand_stations_temperature.csv

No of observations using command
*!wc -l ~/spark/outputs/ghcnd/newzealand_stations_temperature.csv/**
is also 474654

```
#Count number of rows
!wc -l ~/spark/outputs/ghcnd/newzealand_stations_temperature.csv/*
```

```
   20759 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00000-5159e788-ed16-4188-bb1d-76e43d84f
   41327 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00001-5159e788-ed16-4188-bb1d-76e43d84f
   33897 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00002-5159e788-ed16-4188-bb1d-76e43d84f
   51253 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00003-5159e788-ed16-4188-bb1d-76e43d84f
   53263 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00004-5159e788-ed16-4188-bb1d-76e43d84f
   54500 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00005-5159e788-ed16-4188-bb1d-76e43d84f
   39732 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00006-5159e788-ed16-4188-bb1d-76e43d84f
   39422 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00007-5159e788-ed16-4188-bb1d-76e43d84f
   48356 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00008-5159e788-ed16-4188-bb1d-76e43d84f
   35247 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00009-5159e788-ed16-4188-bb1d-76e43d84f
   11448 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00010-5159e788-ed16-4188-bb1d-76e43d84f
   10856 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00011-5159e788-ed16-4188-bb1d-76e43d84f
   10489 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00012-5159e788-ed16-4188-bb1d-76e43d84f
   22166 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00013-5159e788-ed16-4188-bb1d-76e43d84f
    1939 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/part-00014-5159e788-ed16-4188-bb1d-76e43d84f
       0 /users/home/abh89/spark/outputs/ghcnd/newzealand_stations_temperature.csv/_SUCCESS
  474654 total
```

Plotting time series
Load files from local using pandas.
As the data in the table is spread across day, month and year and the values of TMIN and TMAX are also spread across the timeframe, getting the data for plot would require grouping the data by year,month,day and aggregating on the mean of the values. To

achieve this we pivot the table and get the corresponding average of TMIN and TMAX against each year,month,day.
For plotting the graph for each individual station we loop over a list of stations and save the plot in local. For day wise data of each station (split by year), we loop over years in addition to looping over stations. Plots are saved locally.
Plot for stations have an overview chart that shows temperature variations across years for every station in New Zealand. Detailed day wise variation for each station is also saved in respective folders separated by years.

e) We filter by element PRCP on the daily core to create the required table. We also divide the value by 10 to arrive at rainfall in millimeters. The date column is also split into year, month and day.
For getting the average rainfall by country and year, we groupby country code and year and aggregate on count of ID and mean of rainfall values.

```
+------------+----+-----+------------+------------+-----------------+-------------------+
|COUNTRY_CODE|YEAR|COUNT|MIN_RAINFALL|MAX_RAINFALL|   AVERAGE_RAINFALL|       COUNTRY_NAME|
+------------+----+-----+------------+------------+-----------------+-------------------+
|          EK|2000|    1|       436.1|       436.1|            436.1|  Equatorial Guinea|
|          DR|1975|    1|       341.4|       341.4|            341.4| Dominican Republic |
|          LA|1974|    2|         0.0|       496.1|           248.05|              Laos |
|          BH|1978|    7|         0.0|       490.0| 224.4714285714286|             Belize|
|          NN|1979|   10|         0.0|       493.0|            196.7|       Sint Maarten|
|          CS|1974|    2|         0.0|       364.0|            182.0|        Costa Rica |
|          BH|1979|   11|         0.0|       495.0|175.55454545454543|             Belize|
|          NS|1973|    3|         0.0|       257.0|            171.0|           Suriname|
|          UC|1978|   26|         0.0|       496.1|167.50384615384615|            Curacao|
|          BH|1977|    7|         0.0|       491.0|154.17142857142858|             Belize|
|          HO|1978|   49|         0.0|       500.1| 146.9612244897959|          Honduras |
|          UC|1977|   52|         0.0|       496.1|144.25384615384615|            Curacao|
|          NN|1978|   23|         0.0|       490.0| 129.2869565217391|       Sint Maarten|
|          HO|1977|   36|         0.0|       500.1| 128.4138888888889|          Honduras |
|          TD|1978|   13|         0.0|       496.1|            126.5|Trinidad and Tobago |
|          GY|1976|    3|         0.0|       364.0|121.33333333333333|            Guyana |
|          UC|1979|   15|         0.0|       491.0|           116.82|            Curacao|
|          TS|1973|    4|         0.0|       156.0|            116.2|           Tunisia |
|          BM|2006|    2|         0.0|       230.4|            115.2|              Burma|
|          EK|2001|    1|       110.0|       110.0|            110.0|  Equatorial Guinea|
+------------+----+-----+------------+------------+-----------------+-------------------+
only showing top 20 rows
```

Equatorial Guinea has the highest average rainfall across the entire dataset for a single year. The result doesn't look sensible as it is only a single observation. It is possible that the data for a time period is entered against a single observation.

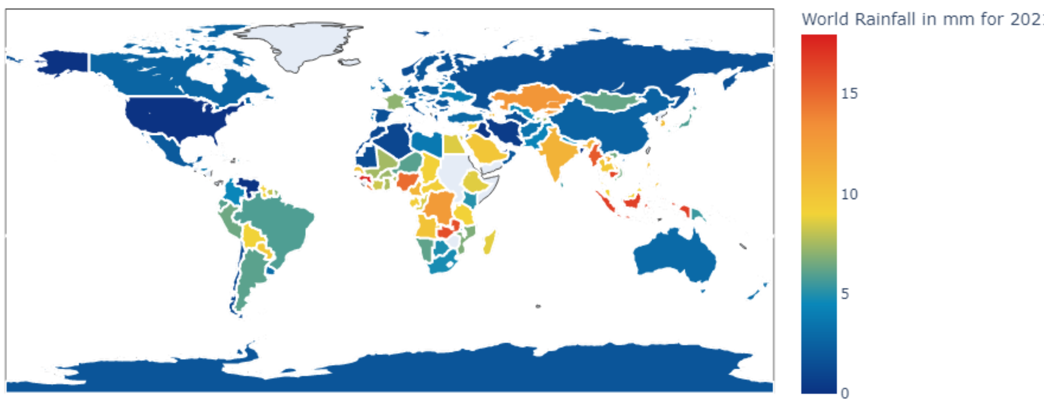
```
data_rainfall[data_rainfall['AVERAGE_RAINFALL']==data_rainfall['AVERAGE_RAINFALL'].max()]
```

	COUNTRY_CODE	YEAR	COUNT	MIN_RAINFALL	MAX_RAINFALL	AVERAGE_RAINFALL	COUNTRY_NAME
0	EK	2000	1	436.1	436.1	436.1	Equatorial Guinea

This result is consistent with the previous analysis.

#Save rainfall data to hdfs
rainfall.repartition(1).write.csv("/user/abh89/spark/outputs/ghcnd/rainfall.csv")
#Save from hdfs to local
!hdfs dfs -copyToLocal /user/abh89/spark/outputs/ghcnd/rainfall.csv ~/spark/outputs/ghcnd/rainfall.csv

Plotting choropleth map
Load the files using pandas. Filter observations for 2021. We would use the Graph Objects module in the Plotly library for this.[2]



Bangladesh, Venezuela and the United Arab Emirates have 0 average rainfall for 2021. Also, few countries like Sudan, Somalia, Zimbabwe and North Korea don't have any precipitation data.

References

1. https://medium.com/@nikolasbielski/using-a-custom-udf-in-pyspark-to-compute-haversine-distances-d877b77b4b18
2. https://levelup.gitconnected.com/plotting-choropleth-maps-in-python-b74c53b8d0a6