BDAT 1008 - 04 Data Collection and Curation Assignment 2

Shubham Chawla 200493036

Question 1 A sample schema for orders_data is given below. Import the data. How many unique cities are there?

In [73]:

```
# Importing Necessary Libraries
from pyspark.sql.types import *
from pyspark.sql import functions as F
from pyspark.sql import DataFrameWriter as W
from math import radians, cos, sin, asin, sqrt

# Spark Session
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
spark.conf.set('spark.sql.shuffle.partitions', 6)
spark.conf.set('num-executors', 16)
spark
```

Out[73]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v3.2.1`
Master	`local[*]`
AppName	`pyspark-shell`

In [74]:

```
%ls orders_data
```

```
 Volume in drive C is OS
 Volume Serial Number is 3C52-E078

 Directory of C:\Users\shubh\Desktop\1008 Data Collection & Curation\A2\orders_data

2022-07-10  05:59 PM    <DIR>          .
2022-07-17  06:58 PM    <DIR>          ..
2022-07-10  05:59 PM            14,173 orders_1.csv
2022-07-10  05:59 PM            14,413 orders_10.csv
2022-07-10  05:59 PM            14,392 orders_100.csv
2022-07-10  05:59 PM            14,390 orders_11.csv
2022-07-10  05:59 PM            14,432 orders_12.csv
2022-07-10  05:59 PM            14,287 orders_13.csv
2022-07-10  05:59 PM            14,354 orders_14.csv
2022-07-10  05:59 PM            14,465 orders_15.csv
2022-07-10  05:59 PM            14,442 orders_16.csv
2022-07-10  05:59 PM            14,592 orders_17.csv
2022-07-10  05:59 PM            14,439 orders_18.csv
2022-07-10  05:59 PM            14,455 orders_19.csv
2022-07-10  05:59 PM            14,485 orders_2.csv
2022-07-10  05:59 PM            14,493 orders_20.csv
2022-07-10  05:59 PM            14,461 orders_21.csv
2022-07-10  05:59 PM            14,449 orders_22.csv
2022-07-10  05:59 PM            14,518 orders_23.csv
2022-07-10  05:59 PM            14,416 orders_24.csv
2022-07-10  05:59 PM            14,464 orders_25.csv
2022-07-10  05:59 PM            14,492 orders_26.csv
2022-07-10  05:59 PM            14,432 orders_27.csv
2022-07-10  05:59 PM            14,391 orders_28.csv
2022-07-10  05:59 PM            14,414 orders_29.csv
2022-07-10  05:59 PM            14,460 orders_3.csv
2022-07-10  05:59 PM            14,489 orders_30.csv
2022-07-10  05:59 PM            14,488 orders_31.csv
2022-07-10  05:59 PM            14,419 orders_32.csv
2022-07-10  05:59 PM            14,451 orders_33.csv
2022-07-10  05:59 PM            14,406 orders_34.csv
2022-07-10  05:59 PM            14,378 orders_35.csv
2022-07-10  05:59 PM            14,449 orders_36.csv
2022-07-10  05:59 PM            14,406 orders_37.csv
2022-07-10  05:59 PM            14,470 orders_38.csv
2022-07-10  05:59 PM            14,460 orders_39.csv
2022-07-10  05:59 PM            14,156 orders_4.csv
2022-07-10  05:59 PM            14,565 orders_40.csv
2022-07-10  05:59 PM            14,655 orders_41.csv
2022-07-10  05:59 PM            14,438 orders_42.csv
2022-07-10  05:59 PM            14,350 orders_43.csv
2022-07-10  05:59 PM            14,477 orders_44.csv
2022-07-10  05:59 PM            14,445 orders_45.csv
2022-07-10  05:59 PM            14,414 orders_46.csv
2022-07-10  05:59 PM            14,506 orders_47.csv
2022-07-10  05:59 PM            14,524 orders_48.csv
2022-07-10  05:59 PM            14,486 orders_49.csv
```

```
2022-07-10  05:59 PM            14,313 orders_5.csv
2022-07-10  05:59 PM            14,427 orders_50.csv
2022-07-10  05:59 PM            14,502 orders_51.csv
2022-07-10  05:59 PM            14,419 orders_52.csv
2022-07-10  05:59 PM            14,477 orders_53.csv
2022-07-10  05:59 PM            14,427 orders_54.csv
2022-07-10  05:59 PM            14,470 orders_55.csv
2022-07-10  05:59 PM            14,420 orders_56.csv
2022-07-10  05:59 PM            14,534 orders_57.csv
2022-07-10  05:59 PM            14,442 orders_58.csv
2022-07-10  05:59 PM            14,482 orders_59.csv
2022-07-10  05:59 PM            14,308 orders_6.csv
2022-07-10  05:59 PM            14,482 orders_60.csv
2022-07-10  05:59 PM            14,495 orders_61.csv
2022-07-10  05:59 PM            14,476 orders_62.csv
2022-07-10  05:59 PM            14,441 orders_63.csv
2022-07-10  05:59 PM            14,471 orders_64.csv
2022-07-10  05:59 PM            14,450 orders_65.csv
2022-07-10  05:59 PM            14,416 orders_66.csv
2022-07-10  05:59 PM            14,384 orders_67.csv
2022-07-10  05:59 PM            14,545 orders_68.csv
2022-07-10  05:59 PM            14,407 orders_69.csv
2022-07-10  05:59 PM            14,473 orders_7.csv
2022-07-10  05:59 PM            14,423 orders_70.csv
2022-07-10  05:59 PM            14,571 orders_71.csv
2022-07-10  05:59 PM            14,433 orders_72.csv
2022-07-10  05:59 PM            14,488 orders_73.csv
2022-07-10  05:59 PM            14,480 orders_74.csv
2022-07-10  05:59 PM            14,537 orders_75.csv
2022-07-10  05:59 PM            14,465 orders_76.csv
2022-07-10  05:59 PM            14,421 orders_77.csv
2022-07-10  05:59 PM            14,332 orders_78.csv
2022-07-10  05:59 PM            14,402 orders_79.csv
2022-07-10  05:59 PM            14,465 orders_8.csv
2022-07-10  05:59 PM            14,460 orders_80.csv
2022-07-10  05:59 PM            14,457 orders_81.csv
2022-07-10  05:59 PM            14,478 orders_82.csv
2022-07-10  05:59 PM            14,419 orders_83.csv
2022-07-10  05:59 PM            14,398 orders_84.csv
2022-07-10  05:59 PM            14,429 orders_85.csv
2022-07-10  05:59 PM            14,331 orders_86.csv
2022-07-10  05:59 PM            14,510 orders_87.csv
2022-07-10  05:59 PM            14,345 orders_88.csv
2022-07-10  05:59 PM            14,364 orders_89.csv
2022-07-10  05:59 PM            14,434 orders_9.csv
2022-07-10  05:59 PM            14,393 orders_90.csv
2022-07-10  05:59 PM            14,449 orders_91.csv
2022-07-10  05:59 PM            14,496 orders_92.csv
2022-07-10  05:59 PM            14,474 orders_93.csv
2022-07-10  05:59 PM            14,465 orders_94.csv
2022-07-10  05:59 PM            14,388 orders_95.csv
2022-07-10  05:59 PM            14,369 orders_96.csv
```

```
2022-07-10  05:59 PM            14,482 orders_97.csv
2022-07-10  05:59 PM            14,480 orders_98.csv
2022-07-10  05:59 PM            14,424 orders_99.csv
             100 File(s)      1,444,064 bytes
               2 Dir(s)  78,929,788,928 bytes free
```

In [75]:

```
# Importing the data and applying sample schema
from pyspark.sql.types import *

file_location = "orders_data\orders_*.csv"

ordersSchema = StructType([
  StructField("Order_ID", DoubleType(), True),
  StructField("Country", StringType(), True),
  StructField("Province", StringType(), True),
  StructField("City", StringType(), True),
  StructField("Latitude", DoubleType(), True),
  StructField("Longitude", DoubleType(), True),
  StructField("TimeStamp", StringType(), True),
  StructField("Sales_Volume", DoubleType(), True)])

print(type(ordersSchema))
```

```
<class 'pyspark.sql.types.StructType'>
```

In [76]:

```
# creating DataFrame
data = (
  spark
    .read
    .schema(ordersSchema)
    .csv(file_location)
)

# create table for SQL analytics
data.createOrReplaceTempView("orders")

# Checking Size of pyspark dataframe
print((data.count(), len(data.columns)))

data.show(5)
```

```
(20000, 8)
+--------+-------+--------+-------------+---------+---------+-------------------+------------+
|Order_ID|Country|Province|         City| Latitude|Longitude|          TimeStamp|Sales_Volume|
+--------+-------+--------+-------------+---------+---------+-------------------+------------+
|231542.0| Canada|      AB|      Calgary|-113.9835|-113.9389|2022/04/22 08:28:49|       41.49|
|473450.0| Canada|      AB|     Edmonton|-113.4467|-113.3654|2022/04/22 05:04:24|       48.52|
|376604.0| Canada|      AB| Medicine Hat|-110.5798|-110.4884|2022/04/22 18:14:14|       60.79|
|440105.0| Canada|      AB|Sherwood Park|-113.2427| -113.242|2022/04/22 11:27:20|       77.81|
|483058.0| Canada|      AB|     Beaumont|-113.3783|-113.2894|2022/04/22 21:04:24|       12.06|
+--------+-------+--------+-------------+---------+---------+-------------------+------------+
only showing top 5 rows
```

In [77]:

```
# Checking for duplicate order ids
if data.count() > data.dropDuplicates(['Order_ID']).count():
    print('Data has {} duplicate OrderIDs'.format(data.count() - data.dropDuplicates(['Order_ID']).count()))
```

```
Data has 648 duplicate OrderIDs
```

In [78]:

```
# Removing Duplicate OrderIDs
duplicates_removed_df = data.dropDuplicates(['Order_ID'])
# create table for SQL analytics - duplicates removed
duplicates_removed_df.createOrReplaceTempView("orders_duplicates_removed")
# Checking Size of dataframe after removing duplicates
print((duplicates_removed_df.count(), len(duplicates_removed_df.columns)))
```

```
(19352, 8)
```

In [79]:

```
# Unique Cities Count without removing duplicates
df_sql = spark.sql("SELECT count(distinct(City)) AS `Unique Cities Count` FROM orders")
df_sql.show()
```

```
+-------------------+
|Unique Cities Count|
+-------------------+
|                619|
+-------------------+
```

In [80]:

```
# Unique Cities Count after removing duplicate Order_ID
df_sql = spark.sql("SELECT count(distinct(City)) AS `Unique Cities Count` FROM orders_duplicates_removed")
df_sql.show()
```

```
+-------------------+
|Unique Cities Count|
+-------------------+
|                612|
+-------------------+
```

Question 2 For each province, show the latest time of the order that was made

In [83]:

```
df_sql = spark.sql("""
select Province, max(TimeStamp) AS `Latest Time of Order`
from orders_duplicates_removed
group by Province
order by Province
""")
df_sql.show()
```

```
+--------+--------------------+
|Province|Latest Time of Order|
+--------+--------------------+
|      AB| 2022/04/23 01:13:04|
|      BC| 2022/04/23 01:05:14|
|      MB| 2022/04/23 00:25:42|
|      NB| 2022/04/23 00:47:52|
|      NL| 2022/04/23 00:36:20|
|      NS| 2022/04/23 00:19:17|
|      NT| 2022/04/22 17:33:18|
|      ON| 2022/04/23 01:18:38|
|      PE| 2022/04/22 23:23:21|
|      QC| 2022/04/23 01:17:12|
|      SK| 2022/04/23 01:19:30|
|      YT| 2022/04/22 20:48:47|
+--------+--------------------+
```

Question 3 A point of interest (POI) is a specific point location that people take interest. Import the POI data set (POI.csv).

In [84]:

```
# Importing data using inferred schema
# Location and type of the file being imported
path = 'POI.csv'
file_type = "csv"

# CSV OPTIONS
infer_schema = "true" #By using this option spark will automatically go through the csv file and infer the schema of each column.
first_row_is_header = "true"
```

```
delimiter = ","

# Importing POI.csv file
df_POI = spark.read.format(file_type) \
  .option("inferSchema",infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(path)
```

In [85]:

```
df_POI.columns
```

Out[85]:

```
['POI_ID', ' Latitude', 'Longitude']
```

In [86]:

```
#renaming column Latitude to remove an extra space in the beginning
df_POI = df_POI.withColumnRenamed(" Latitude","Latitude")
df_POI.columns
```

Out[86]:

```
['POI_ID', 'Latitude', 'Longitude']
```

In [87]:

```
# create view for SQL analytics
df_POI.createOrReplaceTempView("POI")
df_POI.show(5)
```

```
+------+---------+-----------+
|POI_ID| Latitude|  Longitude|
+------+---------+-----------+
| POI-1|45.521629| -74.085634|
| POI-2|53.596345|-114.465122|
| POI-3|44.897823| -62.987528|
+------+---------+-----------+
```

Question 4 For each Order_ID and POI pair, get the distance between the Order_ID and the POI based on the geographic Latitude and Longitude.

In [88]:

```
# Joining views orders and POI (A = orders, B = POI)
combined_df = spark.sql(
    """select int(v1.Order_ID), v2.POI_ID, v1.Latitude AS `Lat_A`, v1.Longitude AS `Long_A`, v2.Latitude AS `Lat_B`, v2.Longitude AS `Long_B`
    from orders_duplicates_removed v1, POI v2
    """
)
combined_df.show(10)
```

```
+--------+------+---------+---------+---------+-----------+
|Order_ID|POI_ID|    Lat_A|   Long_A|    Lat_B|     Long_B|
+--------+------+---------+---------+---------+-----------+
|  250410| POI-1|-113.2142|-113.1836|45.521629| -74.085634|
|  250410| POI-2|-113.2142|-113.1836|53.596345|-114.465122|
|  250410| POI-3|-113.2142|-113.1836|44.897823| -62.987528|
|  286817| POI-1|-111.4565|-111.4168|45.521629| -74.085634|
|  286817| POI-2|-111.4565|-111.4168|53.596345|-114.465122|
|  286817| POI-3|-111.4565|-111.4168|44.897823| -62.987528|
|  298450| POI-1|-113.3683|-113.3579|45.521629| -74.085634|
|  298450| POI-2|-113.3683|-113.3579|53.596345|-114.465122|
|  298450| POI-3|-113.3683|-113.3579|44.897823| -62.987528|
|  480255| POI-1| -79.3772| -79.3679|45.521629| -74.085634|
+--------+------+---------+---------+---------+-----------+
only showing top 10 rows
```

In [90]:

```
from IPython.display import Image
Image(url= "formula.png", width=600, height=400)
```

Out[90]:

In [91]:

```
# Calculating distance between two points using Haversine Distance formula

import pyspark.sql.functions as F
combined_df = combined_df.withColumn("a", (
        F.pow(F.sin(F.radians(F.col("Lat_B") - F.col("Lat_A")) / 2), 2) +
        F.cos(F.radians(F.col("Lat_A"))) * F.cos(F.radians(F.col("Lat_B"))) *
        F.pow(F.sin(F.radians(F.col("Long_B") - F.col("Long_A")) / 2), 2)
```

```
    )).withColumn("Distance", F.atan2(F.sqrt(F.col("a")), F.sqrt(-F.col("a") + 1)) * 12742.018)  #2*uE = 2 * 6371.009

combined_df = combined_df.drop('a')
combined_df = combined_df.withColumn("Distance", F.round(combined_df["Distance"], 3))
combined_df.show()
```

```
+--------+------+---------+---------+---------+----------+---------+
|Order_ID|POI_ID|    Lat_A|   Long_A|    Lat_B|    Long_B| Distance|
+--------+------+---------+---------+---------+----------+---------+
|  250410| POI-1|-113.2142|-113.1836|45.521629| -74.085634|16731.229|
|  250410| POI-2|-113.2142|-113.1836|53.596345|-114.465122| 18546.88|
|  250410| POI-3|-113.2142|-113.1836|44.897823| -62.987528| 16216.34|
|  286817| POI-1|-111.4565|-111.4168|45.521629| -74.085634|16702.623|
|  286817| POI-2|-111.4565|-111.4168|53.596345|-114.465122|18345.494|
|  286817| POI-3|-111.4565|-111.4168|44.897823| -62.987528|16232.481|
|  298450| POI-1|-113.3683|-113.3579|45.521629| -74.085634|16731.879|
|  298450| POI-2|-113.3683|-113.3579|53.596345|-114.465122|18564.407|
|  298450| POI-3|-113.3683|-113.3579|44.897823| -62.987528|16213.075|
|  480255| POI-1| -79.3772| -79.3679|45.521629| -74.085634|13892.397|
|  480255| POI-2| -79.3772| -79.3679|53.596345|-114.465122|14961.485|
|  480255| POI-3| -79.3772| -79.3679|44.897823| -62.987528|13859.726|
|  433267| POI-1|  -82.978|  -82.895|45.521629| -74.085634|14296.757|
|  433267| POI-2|  -82.978|  -82.895|53.596345|-114.465122|15286.753|
|  433267| POI-3|  -82.978|  -82.895|44.897823| -62.987528|14261.037|
|  200925| POI-1| -79.3595| -79.3471|45.521629| -74.085634|13890.402|
|  200925| POI-2| -79.3595| -79.3471|53.596345|-114.465122|14959.962|
|  200925| POI-3| -79.3595| -79.3471|44.897823| -62.987528|13857.713|
|  346582| POI-1|-114.3696|-114.3627|45.521629| -74.085634|16736.781|
|  346582| POI-2|-114.3696|-114.3627|53.596345|-114.465122|18676.975|
+--------+------+---------+---------+---------+----------+---------+
only showing top 20 rows
```

Question 5 For each Order_ID, identify the POI with the shortest distance. Retain only 1 record for each Order_ID. (Check: Your end result should have the same record count as your orders_data dataset.)

In [101…

```
# create view for SQL analytics
combined_df.createOrReplaceTempView("orders_POI")

# Finding POI with shortest distance for each Order_ID by using window function
shortest_df = spark.sql(
    """WITH cte AS (
        SELECT Order_ID, POI_ID, Distance,
            RANK() OVER ( PARTITION BY Order_ID
            ORDER BY Distance ASC
            ) AS r
        FROM orders_POI
       )
       SELECT Order_ID, POI_ID, Distance AS `Shortest Distance`
       FROM cte
       WHERE r = 1
       ORDER BY Order_ID;
    """
)
shortest_df.show()
```

```
+--------+------+-----------------+
|Order_ID|POI_ID|Shortest Distance|
+--------+------+-----------------+
|  200021| POI-3|        16204.454|
|  200026| POI-3|        13901.091|
|  200041| POI-3|         16209.39|
|  200099| POI-3|        13889.543|
|  200137| POI-3|        13954.091|
|  200154| POI-3|        13873.513|
|  200171| POI-3|        16167.789|
|  200183| POI-3|        13188.207|
|  200220| POI-3|        13856.457|
|  200221| POI-3|        13880.355|
|  200231| POI-3|        13857.506|
|  200242| POI-3|        13877.028|
|  200308| POI-3|        16213.982|
|  200311| POI-3|        13840.551|
|  200313| POI-3|        13857.452|
|  200335| POI-3|        13621.158|
|  200343| POI-3|        13883.572|
|  200347| POI-3|        13194.087|
|  200374| POI-3|        12933.142|
|  200398| POI-3|        14923.097|
+--------+------+-----------------+
only showing top 20 rows
```

In [102…

```
# Checking size of dataframe after retaining only single record for each unique Order_ID
print((shortest_df.count(), len(shortest_df.columns)))
```

```
(19352, 3)
```

In [103…

```
# Checking order_data dataset size with duplicates removed
print((duplicates_removed_df.count(), len(duplicates_removed_df.columns)))
```

```
(19352, 8)
```

Question 6 Based on #5, for each POI, get the average, standard deviation and max of the (shortest) distances, as well as the

count of the orders.

In [107…

```
# create view for SQL analytics
shortest_df.createOrReplaceTempView("v6")

# Using Aggregate Functions
descriptive_df = spark.sql(
    """
    SELECT POI_ID, round(avg(`Shortest Distance`),3) AS Average_Distance,
    round(stddev(`Shortest Distance`),3) AS Stddev_Distance,
    max(`Shortest Distance`) AS `Max_D`,
    count(Order_ID) AS `Order_Count`
    FROM v6
    Group by POI_ID
    """
)
descriptive_df.show()
```

```
+------+----------------+---------------+--------+-----------+
|POI_ID|Average_Distance|Stddev_Distance|   Max_D|Order_Count|
+------+----------------+---------------+--------+-----------+
| POI-3|       14872.704|       1310.943|16235.19|      19352|
+------+----------------+---------------+--------+-----------+
```

Question 7 For each POI, based on the max distance and orders count from #6, calculate the density using the formula: density =orders_count / (π*(max_distance)^2)

In [109…

```
# create view for SQL analytics
descriptive_df.createOrReplaceTempView("v7")

# Casting Density as Decimal Value
density_df = spark.sql(
    """
    SELECT POI_ID, Order_Count, Max_D, CAST(Order_Count/(3.14*pow(Max_D,2)) AS Decimal(25,15)) as Density
    FROM v7

    """
)
density_df.show()
```

```
+------+-----------+--------+-----------------+
|POI_ID|Order_Count|   Max_D|          Density|
+------+-----------+--------+-----------------+
| POI-3|      19352|16235.19|0.000023381989235|
+------+-----------+--------+-----------------+
```