CASE STUDY
PYSPARK

Use Case

Analysis of UBER dataset. This dataset consists of records on pickups and drops, fares charged and the pickup and drop latitudes and longitudes.

Tasks

a. Use the Haversine formula to calculate the distance from the latitudes of the pickup and drop points. Include the formula in a Python package and create the necessary user defined functions in Spark. Store the data as a csv file.

b. Find the date range from the newly created csv file.

c. Aggregate data based on week to depict weekly distance of travel

d. Identify the vendor covering the highest miles.

e. Calculate the total revenue made over a period of choice.

Solution

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import pyspark
import findspark
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import udf
from pyspark.sql.functions import isnan, when, count, col
from pyspark.sql.functions import min,max
from pyspark.sql.functions import weekofyear, sum
import pyspark.sql.types as types
import math
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

Setting Scala Parameters

```
sc = SparkContext()
sqlContext = pyspark.SQLContext(sc)
spark = pyspark.sql.SparkSession(sc)
```

Reading data

```
df0 = spark.read.csv('/Users/sakshambhatia/Downloads/Data_For_Assignment/4/uber_data_nyc_2016-06_3m_partitioned.csv/par
                    ,header='true',inferSchema='true')
```

```
df1 = spark.read.csv('/Users/sakshambhatia/Downloads/Data_For_Assignment/4/uber_data_nyc_2016-06_3m_partitioned.csv/par
                    ,header='true',inferSchema='true')
```

```
df2 = spark.read.csv('/Users/sakshambhatia/Downloads/Data_For_Assignment/4/uber_data_nyc_2016-06_3m_partitioned.csv/par
                    ,header='true',inferSchema='true')
```

```
df3 = spark.read.csv('/Users/sakshambhatia/Downloads/Data_For_Assignment/4/uber_data_nyc_2016-06_3m_partitioned.csv/par
                    ,header='true',inferSchema='true')
```

Uniting the dataset

```
df = df0.union(df1.union(df2.union(df3)))
```

```
df.count()
```

```
3299999
```

Defining the UDF and adding a row

```
def haversine(lon1,lat1,lon2,lat2):
    dLon = (lon2-lon1)*math.pi/180.0
    dLat = (lat2-lat1)*math.pi/180.0

    lat1 = (lat1)*math.pi/180.0
    lat2 = (lat2)*math.pi/180.0

    a = (pow(math.sin(dLat/2),2)+pow(math.sin(dLon/2),2)*math.cos(lat1)*math.cos(lat2));
    rad = 6371
    c = 2*math.asin(math.sqrt(a))
    return (rad*c*0.621371)
```

```
haversine_udf = udf(haversine,types.DoubleType())
uber=df.withColumn("distance_miles",haversine_udf("pickup_longitude","pickup_latitude","dropoff_longitude","dropoff_lat
```

```
uber.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
 |-- total_amount: double (nullable = true)
 |-- distance_miles: double (nullable = true)
```

The new Column is added

```
uber.describe().show()
```

```
+-------+------------------+-------------------+-----------------+-------------------+-----------------+-------------
-----+------------------+
|summary|          VendorID|   pickup_longitude|  pickup_latitude|  dropoff_longitude| dropoff_latitude|      total_a
mount|    distance_miles|
+-------+------------------+-------------------+-----------------+-------------------+-----------------+-------------
-----+------------------+
|  count|           3299999|            3299999|          3299999|            3299999|          3299999|           32
99999|           3299999|
|   mean|1.5279289478572569| -73.05836057064273|40.24678831059782|  -73.1286791673134|40.28652937530498|16.6462403353
31093|22.167555833907294|
| stddev|0.4992194401191109|  8.175606882912328|4.503776121420567|  7.858149695712172|4.328784275308626|127.299855544
32749|327.50130236430664|
|    min|                 1|-115.16835784912108|              0.0|-115.17537689208984|              0.0|            -
195.3|               0.0|
|    max|                 2|                0.0|51.09830093383789| 106.24687957763672|50.31213760375977|         1874
42.26| 6790.291519257411|
+-------+------------------+-------------------+-----------------+-------------------+-----------------+-------------
-----+------------------+
```

Checking the dataset for null values

```
g=uber.select('VendorID','pickup_longitude','pickup_latitude',
                  'dropoff_longitude','dropoff_latitude','total_amount','distance_miles')
g.select([count(when(isnan(c), c)).alias(c) for c in g.columns]).show()
```

```
+--------+----------------+---------------+-----------------+----------------+------------+--------------+
|VendorID|pickup_longitude|pickup_latitude|dropoff_longitude|dropoff_latitude|total_amount|distance_miles|
+--------+----------------+---------------+-----------------+----------------+------------+--------------+
|       0|               0|              0|                0|               0|           0|             0|
+--------+----------------+---------------+-----------------+----------------+------------+--------------+
```

Saving the dataset as a .csv file

```
uber.write.csv('/Users/sakshambhatia/Documents/fullUberDataSet.csv')
```

Creating a table for sql queries

```
uber.createOrReplaceTempView('uber')
```

Solving the queries:
b)
By sql:

```
spark.sql('select min(tpep_pickup_datetime) as dateRange from uber union select max(tpep_pickup_datetime) from uber').s
```

```
+-------------------+
|          dateRange|
+-------------------+
|2016-06-01 12:30:00|
|2016-06-12 21:59:19|
+-------------------+
```

By Dataframe:

```
uber.select(min("tpep_pickup_datetime").alias('start_date')).show()
uber.select(max("tpep_pickup_datetime").alias('end_date')).show()
```

```
+--------------------+
|          start_date|
+--------------------+
|2016-06-01 12:30:00|
+--------------------+

+--------------------+
|            end_date|
+--------------------+
|2016-06-12 21:59:19|
+--------------------+
```

c)
By sql:

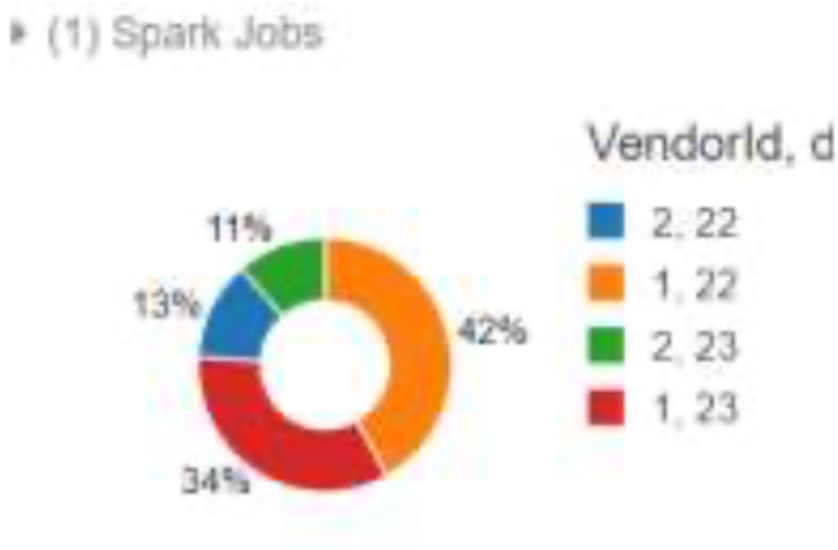
```
spark.sql("select weekofyear(tpep_pickup_datetime) as week,sum(distance_miles) as total_distance from uber group by wee
```

```
+----+--------------------+
|week|      total_distance|
+----+--------------------+
|  22|3.7738647911456354E7|
|  23| 3.541426417288098E7|
+----+--------------------+
```

By Dataframe:

```
uber.groupBy(weekofyear("tpep_pickup_datetime").alias("date_by_week")).agg(sum("distance_miles")).orderBy("date_by_weel
```

```
+------------+--------------------+
|date_by_week| sum(distance_miles)|
+------------+--------------------+
|          22|3.7738647911456354E7|
|          23| 3.541426417288098E7|
+------------+--------------------+
```



d)
By sql:

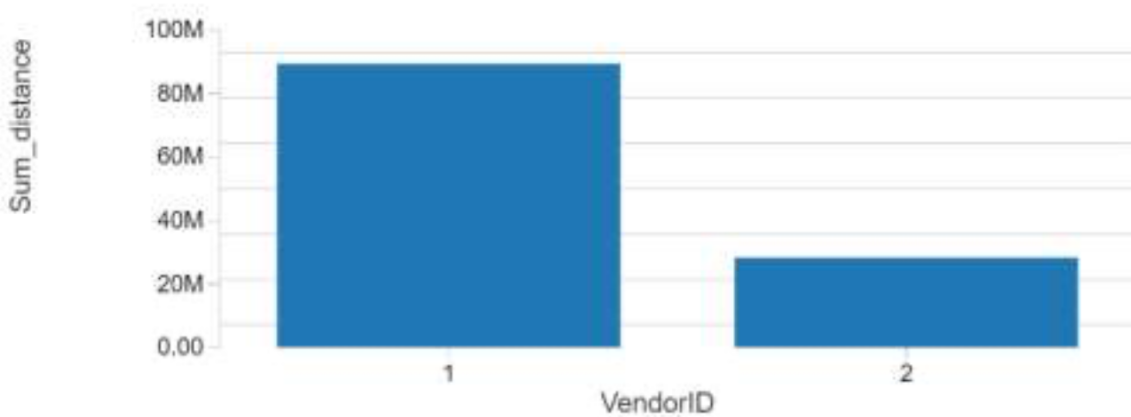
```
spark.sql('select VendorID from uber group by vendorID order by sum(distance_miles) desc limit 1').show()
```

```
+--------+
|VendorID|
+--------+
|       1|
+--------+
```

By Dataframe:

```
(uber.select("VendorID","distance_miles").groupBy('VendorID').agg(sum('distance_miles').alias('Sum_distance'))).select(
```

```
+--------+
|VendorID|
+--------+
|       1|
+--------+
only showing top 1 row
```

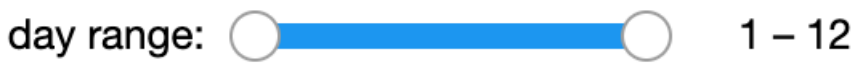


e)

By sql and iwidgets:

```
range_slider = widgets.IntRangeSlider(
    value=[1, 12],
    min=1, max=12., step=1,
    description='day range:',

)
range_slider
```




```
query=("select sum(total_amount) as total_revenue from uber where dayofmonth(tpep_pickup_datetime) between {} and {}"
       .format(range_slider.value[0],range_slider.value[1]))
```

```
spark.sql(query).show()
```

```
+-------------------+
|      total_revenue|
+-------------------+
|5.493257646035227E7|
+-------------------+
```

By Dataframe:

Closing the scala

```
sc.stop()
```