CS4433 Project 3

Initial Skills

William Yang

- PySpark:
    - Before: 0/10
    - After: 4/10
- Spark MapReduce:
    - Before: 0/10
    - After: 6/10
- Spark SQL:
    - Before: 0/10
    - After: 6/10

Gus Teran

- PySpark:
    - Before: 0/10
    - After: 5/10
- Spark MapReduce:
    - Before: 0/10
    - After: 5/10
- Spark SQL:
    - Before: 0/10
    - After: 7/10

Relative Contributions

William Yang:
Data generation file for problem 1, query 1 for problem 1, contents of README

Gus Teran:
Data generation file for problem 2, query 2 for problem 1

Both team members worked together and contributed equally to the tasks for problem 2 (excluding data generation)

Solution Logic Descriptions

Problem 1

Data Generation:

In the python script to generate data for problem 1, an object called ‘Point’ was defined, which held a point’s id and XY coordinates. Additionally, there were two functions called generate_people() and generate_infected().

Generate_people() takes the number of people to generate as an input, then increments through a for loop (with the range being the provided number of people). In the for loop, it creates a Point object with an incrementing ID and gives it a randomly generated XY coordinate. It then adds the Point object to a list of points. After the for loop is finished, the function then creates a Pandas dataframe using the list of points, and returns it.

Generate_infected() takes the dataframe of people and the number of infected people. It then returns a new Pandas dataframe created by sampling the provided people dataframe for the number of infected people requested.

By default, the script generates 1 million people, and then creates two infected files, one containing only 50 infected people and another which contains 5% of the number of people in the people dataframe.

Finally, the script saves these dataframes as CSV files under the name ‘PEOPLE.csv’, ‘INFECTED-small.csv’, and ‘INFECTED-large.csv’.

Query 1:

The logic for query 1 consists of a singular map job then a filter. We can do this because we know the infected file is very small, and thus can be loaded entirely into memory. Thus, we do so and send it to the map function along with the lines of the people file.

In the map function, we check the distance of the point from the people file to each infected point in memory. If the distance is smaller than 6, that means that the point is a close contact, and the map function outputs the point id.

After the map function, we simply filter out empty/null lines in the output, and then save the output file.

Query 2:

Query 2 maps the people and infected cases to a grid, finds close contacts within the grid, and counts the close contacts. The query loads the files from the csv and maps each person and each infective case the grids within a 12 foot square. The grids are 100 ft squares.

We then reduce the close contacts by folding the points and infected cases by the grid into one list for each grid key. Then using a mapping function to map the list of points and infected cases into a list of <infection id, list of people that are close contacts for that infection>.

We fold all of the close contacts for each infected id, remove the duplicates using a set, and sum up the number of close contacts

Problem 2

Data Generation

We define the customers and purchases as a class in python. We then created functions that generate rows of these customers and purchases using the specified data ranges for each. We use the random class to be able to provide random values as needed. We turn the list of rows into a pandas dataframe and

Task 1:

After loading the datafile into a dataframe and defining its schema, it is a simple matter of filtering the schema for only rows where the value of the total transactions is less than or equal to 600.

Task 2:

To get the minimum, median, and maximum transaction totals from the dataframe generated in task 1, we grouped the purchases by the number of items purchased and used an aggregate function. For minimum we used the 'min' value, for maximum we used the 'max' value, and for median we used the 50th percentile value.

Task 3:

The group by and aggregate functions from task 2 were saved into separate dataframes and printed into terminal to show the client.

Task 4:

We joined the customers and the purchases dataframes (from task 1) to get the customer data fro each purchase. We filtered the joined data first to have customer age less than or equal to 25, and then greater than or equal to 18. We grouped the data by the customer id and their age before summing the transaction number of items and transaction total for each customer. This

left us with a dataframe of the customer id, the age, the sum of their transaction number of items, and the sum of their transaction total costs.

Task 5:

We did a cartesian join of the dataframe containing both customer and transaction information from task 4 with itself. Then, it was a simple matter of filtering the resulting dataframe to only contain rows where the first customer's age is smaller than the second customer's, the first customer's total transaction cost amount were larger than the second customer's, and the first customer's total number of transactions were smaller than the second customer's.

Task 6

The filtered data from task 5 was saved as a dataframe with columns for the customer ids, the ages, the sum of the number of items, and the sum of the costs for each filtered pair. We display this schema and this dataframe.