**Analysis of AirBnB data**

I used SQL functions for my assignment – although I did familiarize myself with map/reduce/filter etc. in the process. My task was then mostly transforming the data into a format I could use the correct SQL functions on. I have not put a lot of emphasis on explaining why I use SQL functions like AVG and SUM, as I think most are quite familiar with them. I split each task into a separate .scala file - the solution could be more efficient if the DataFrames where reused between the tasks, but I thought being able to run each task individually was more important.

About task 6:

I managed to find a library function “pointInPolygon” and get my DataFrame on the form

[Neighbourhood name, [WrappedArray[WrappedArray[WrappedArray[.…]]]

I spent several hours trying to figure out how to get the coordinates out of the wrapped array, but eventually ran out of time. I have provided the code for as far as I got. My procedure would be for each listing, check if the latitude and longitude is within a polygon and then use the neighborhood name.

I also had issues exporting outputs to csv and therefore used println and copied this to a text file.

**2a**

CREATE TABLE listings

...

id int NOT NULL PRIMARY KEY

…

CREATE TABLE reviews

...

FOREIGN KEY (listing\_id) REFERENCES listings(id)

…

CREATE TABLE reviews

...

FOREIGN KEY (listing\_id) REFERENCES listings(id)

…

**2b**

I used spark.sql("select count(distinct " + column + ") from listings") and looped through the columns. The output for this task is available in the outputs.txt file.

**2c**

3 cities: New York, Seattle and San Francisco

I used spark.sql("select distinct city from listings")

**2d**

id: primary key used for further analysis by joining with other tables as foreign key

price: booking price per night in USD, min $0 max none

number\_of\_reviews: self-explanatory, min 0, max none

latitude, longitude: location information

**Task 3**

I transformed the original DataFrame where price was a string to a new DataFrame with price as an int so I could use SQL functions like avg and sum using a user defined function (udf):

spark.udf.register("myConvertCurrency", (input: String) => java.text.NumberFormat.getCurrencyInstance(java.util.Locale.US).parse(input).intValue.toInt)

with

listingsRaw.withColumn("price", callUDF("myConvertCurrency", listingsRaw("price")))

This transformation was also used to solve all subsequent tasks.

For each city, I created a view “citySubset” for better reuse as such:

spark.sql("select price, reviews\_per\_month, room\_type from listings where city = '" + city + "'")

**3a**

|  |  |
| --- | --- |
| **City** | **Average price** |
| San Francisco | $251 |
| Seattle | $131 |
| New York | $149 |

I used spark.sql("select avg(price) from citySubset")

**3b**

|  |  |  |  |
| --- | --- | --- | --- |
| **City** | **Shared room** | **Entire home/apt** | **Private room** |
| San Francisco | $88 | $341 | $135 |
| Seattle | $52 | $159 | $77 |
| New York | $69 | $208 | $90 |

I obtained the distinct room\_types as such:

spark.sql("select distinct room\_type from listings")

Then I used

spark.sql("select avg(price) from citySubset where room\_type = '" + roomType + "'")

**3c**

|  |  |
| --- | --- |
| **City** | **Average number of reviews (per listing)** |
| San Francisco | 1.68 |
| Seattle | 2.08 |
| New York | 1.38 |

I used

spark.sql("select avg(reviews\_per\_month) from citySubset")

**3d**

|  |  |
| --- | --- |
| **City** | **Estimated number of booked nights per year** |
| San Francisco | 196512 |
| Seattle | 113723 |
| New York | 923823 |

I defined a udf

spark.udf.register("myNightsPerYear", (input: String) => ((if(input == null) 0.0 else input.toDouble) / 0.7) \* 12)

which I applied to my “citySubset” to obtain a new column “nights\_per\_year”

citySubset.withColumn("nights\_per\_year", callUDF("myNightsPerYear", citySubset("reviews\_per\_month"))).createOrReplaceTempView("citySubsetNightsPerYear")

I then used

spark.sql("select sum(nights\_per\_year) from citySubsetNightsPerYear")

**3e**

|  |  |
| --- | --- |
| **City** | **Estimated amount of money spent per year ($)** |
| San Francisco | 3.38 \* 10^7 |
| Seattle | 1.25 \* 10^7 |
| New York | 1.26 \* 10^8 |

I reused “citySubsetNightsPerYear” from the previous task and used

spark.sql("select sum(nights\_per\_year \* price) from citySubsetNightsPerYear")

**Task 4**

I read the csv file and first created a view “listingsPerHost”, containing number of listings per host:

spark.sql("select count(\*) as count from listings group by host\_id")

**4a**

Average number of listings per host: 1.26

I used

spark.sql("select avg(count) from listingsPerHost")

**4b**

Percentage of hosts with more than 1 listing: 15%

I used

spark.sql("select count from listingsPerHost where count > 1").count.toFloat / listingsPerHost.count

since listingsPerHost.count is equal to the number of unique hosts

**4c**

|  |  |
| --- | --- |
| **City** | **Hosts (host\_name, income)** |
| San Francisco | Max ($3950000)  Ramil ($3920000)  Matt ($1913800) |
| Seattle | Jordan ($3420683)  Sea To Sky Rentals ($3330909)  Daniela ($2487022) |
| New York | Jessica & Doug ($7421883)  John ($4811400)  124 ($4303309) |

I created a view “countsPerListings” by joining listings and calendar on the listings.id = calendar.listing\_id where available was set to “t” (I assume this means booked), grouping by the listing\_id which then represents amount of nights a listing was booked:

spark.sql("select count(\*) as count, listing\_id from calendar JOIN listings ON listings.id = calendar.listing\_id where available = 't' and city = '" + city + "'" + "group by listing\_id")

I then found the income for each host by grouping on host\_id and using sum(price \* countsPerListings.count), ordering by the income descending and limiting the output to 3:

spark.sql("select first(host\_name), sum(price \* countsPerListings.count) as income from listings join countsPerListings on listings.id = countsPerListings.listing\_id group by host\_id order by income desc limit 3")

**Task 5**

I think the task text could be a bit clearer here – I’m not quite sure what the relationship between a review and nights spent is so I’m going to assume that an entry in the reviews table represents 3 nights spent for the respective listing.

I first created a view “reviewsListings”, joining reviews and listings tables and selecting appropriate columns.

**5a**

|  |  |
| --- | --- |
| **City** | **Guests (reviewer\_name, nights\_spent)** |
| San Francisco | Emily (87)  Zafar (81)  Claire (75) |
| Seattle | Amanda (213)  Kathryn (102)  David (81) |
| New York | J. B. (79)  Andy (41)  Adrienne (33) |

I used

spark.sql("select first(reviewer\_name), (count(\*) \* 3) as count from reviewsListings where city = '" + city + "' group by reviewer\_id order by count desc limit 3")

Grouping by reviewer\_id gives number of listings belonging to a reviewer.

**5b**

Guest who spent the most money: Claire ($57132)

I used

spark.sql("select reviewer\_id, first(reviewer\_name), sum(price) \* 3 as spent from reviewsListings group by reviewer\_id order by spent desc limit 1")