As in the first assignment, I used Spark SQL and DataFrames for my solutions. My solutions are in Scala and packaged to .jar using SBT. My solutions are based around transforming columns of the DataFrames using user defined functions (UDFs) and then using the appropriate SQL statements.

1 Important Terms of a Listing/Neighborhood

My approach for this task is intuitive, but might not be the fastest. The resource consuming operation in my solution is finding the total number of documents with term *t* in it. My TF-IDF approach is as follows:

1. Trim listings for empty descriptions
    - Using df.filter($"description".isNotNull)
2. Normalize description and make it into an array of words (new column)
    - If neighborhoods: aggregate all descriptions belonging to the neighborhood. This made me run out of memory on my local machine, and I had to use a sample.
3. Collect words to check
    - If single listing, use the array of words for the listing with the given id
    - If neighborhood, concatenate the array of words of all listings in that neighborhood
4. For each (distinct) word in the array, calculate:
    - Occurrences in document (count in the collected words)
    - Number of documents with the word
        - Using SQL function array_contains()
    - TF-IDF using the values found
5. Sort the TF-IDF values found and take the top 100

I assumed that for a given listing id the “document” is each listing, while for neighborhoods it is each neighborhood (this means that for listings the total number of documents is the number of listings, while for neighborhoods it is the number of neighborhoods).

2 Finding Alternative Listings

This task was somewhat straightforward with the use of SQL functions.

1. Convert price column of listings to an Integer
2. Find all listings vacant on the date and in the price range
   - SQL: *"select id, name, amenities, latitude, longitude, price FROM listings JOIN calendar ON listings.id = calendar.listing_id WHERE available = 'f' and room_type = '" + currentRoomType + "' AND date = '" + currentDate + "' and price < '" + currentMaxPrice + "'"*
3. Calculate the distance (create new column) for these listings and remove those out of range
   - Using a UDF and creating a new column:
     df.withColumn("distance", distanceUdf($"latitude", $"longitude"));
4. Count common amenities for each of these listings and sort by it, limit by given top *n*
   - Using a UDF and creating a new column:
     val commonAmenitiesUdf = udf ((input: Seq[String]) => input.intersect(currentAmenities).length);

     df.withColumn("number_of_common_amenities", commonAmenitiesUdf($"amenities"))

Since the assignment states that input should be as defined in the PDF, there is no path parameter for this program, input and output folder will be the root folder (where the program is run).

Visualization

I had to include latitude and longitude in the exported .tsv temporarily to make this work. I used listing 12607303, date 2016-12-29, distance 10 km and limited by 10 results. The output for running the program with these parameters is provided.

URL: *https://jameyer.carto.com/builder/18516592-1622-11e7-bca0-0e3a376473ab/embed*