

commercial_data_analysis

October 25, 2020

0.0.1 Commercial Data Analysis

0.0.2 University of California, Santa Barbara

0.0.3 PSTAT 135/235: Big Data Analytics

0.0.4 Last Updated: May 30, 2020

0.0.5 INSTRUCTIONS

In this assignment, you will work with a dataset containing information about businesses. Each record is a business location. Follow the steps below, writing and running the code in blocks, and displaying the solutions.

Each question part is worth 1 POINT, for a total of 15 POINTS.

```
[33]: from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("comm") \
    .getOrCreate()
```

```
[6]: # note that read.json can read a zipped JSON directly
df = spark.read.json('part-00000-a159c41a-bc58-4476-9b78-c437667f9c2b-c000.json.
    ↪gz')
```

1. (1 PT) Read in the dataset and show the number of records

```
[7]: df.count()
```

```
[7]: 154679
```

2. (1 PT) Print the schema

```
[8]: df.printSchema()
```

```
root
 |-- address: struct (nullable = true)
 |    |-- city: string (nullable = true)
```

```

|   |-- coordinates: struct (nullable = true)
|   |   |-- lat: double (nullable = true)
|   |   |-- lon: double (nullable = true)
|   |-- country: string (nullable = true)
|   |-- county: string (nullable = true)
|   |-- full_address: string (nullable = true)
|   |-- highway_number: string (nullable = true)
|   |-- is_headquarters: boolean (nullable = true)
|   |-- is_parsed: boolean (nullable = true)
|   |-- post_direction: string (nullable = true)
|   |-- pre_direction: string (nullable = true)
|   |-- secondary_number: string (nullable = true)
|   |-- state: string (nullable = true)
|   |-- street: string (nullable = true)
|   |-- street_address: string (nullable = true)
|   |-- street_number: string (nullable = true)
|   |-- street_type: string (nullable = true)
|   |-- type_of_address: string (nullable = true)
|   |-- zip: string (nullable = true)
|   |-- zip_suffix: string (nullable = true)
|-- business_tags: struct (nullable = true)
|   |-- no: array (nullable = true)
|   |   |-- element: string (containsNull = true)
|   |-- tags: array (nullable = true)
|   |   |-- element: struct (containsNull = true)
|   |   |   |-- name: string (nullable = true)
|   |   |   |-- value: string (nullable = true)
|   |-- yes: array (nullable = true)
|   |   |-- element: string (containsNull = true)
|-- hours: struct (nullable = true)
|   |-- any_day_is_24: boolean (nullable = true)
|   |-- friday_close: string (nullable = true)
|   |-- friday_lb: long (nullable = true)
|   |-- friday_open: string (nullable = true)
|   |-- friday_total_seconds: long (nullable = true)
|   |-- hours: struct (nullable = true)
|   |   |-- Friday: string (nullable = true)
|   |   |-- Monday: string (nullable = true)
|   |   |-- Saturday: string (nullable = true)
|   |   |-- Sunday: string (nullable = true)
|   |   |-- Thursday: string (nullable = true)
|   |   |-- Tuesday: string (nullable = true)
|   |   |-- Wednesday: string (nullable = true)
|   |-- monday_close: string (nullable = true)
|   |-- monday_lb: long (nullable = true)
|   |-- monday_open: string (nullable = true)
|   |-- monday_total_seconds: long (nullable = true)
|   |-- saturday_close: string (nullable = true)

```

```

|   |-- saturday_lb: long (nullable = true)
|   |-- saturday_open: string (nullable = true)
|   |-- saturday_total_seconds: long (nullable = true)
|   |-- sunday_close: string (nullable = true)
|   |-- sunday_lb: long (nullable = true)
|   |-- sunday_open: string (nullable = true)
|   |-- sunday_total_seconds: long (nullable = true)
|   |-- thursday_close: string (nullable = true)
|   |-- thursday_lb: long (nullable = true)
|   |-- thursday_open: string (nullable = true)
|   |-- thursday_total_seconds: long (nullable = true)
|   |-- tuesday_close: string (nullable = true)
|   |-- tuesday_lb: long (nullable = true)
|   |-- tuesday_open: string (nullable = true)
|   |-- tuesday_total_seconds: long (nullable = true)
|   |-- wednesday_close: string (nullable = true)
|   |-- wednesday_lb: long (nullable = true)
|   |-- wednesday_open: string (nullable = true)
|   |-- wednesday_total_seconds: long (nullable = true)
|   |-- week_total_hours_pretty: string (nullable = true)
|   |-- week_total_minutes_pretty: string (nullable = true)
|   |-- week_total_seconds: long (nullable = true)
|-- id: string (nullable = true)
|-- menu: struct (nullable = true)
|   |-- price_range: string (nullable = true)
|   |-- url: string (nullable = true)
|-- reviews: array (nullable = true)
|   |-- element: struct (containsNull = true)
|   |   |-- content: string (nullable = true)
|   |   |-- date: string (nullable = true)
|   |   |-- dislikes: long (nullable = true)
|   |   |-- gender: string (nullable = true)
|   |   |-- id: string (nullable = true)
|   |   |-- language: string (nullable = true)
|   |   |-- likes: long (nullable = true)
|   |   |-- source: string (nullable = true)
|   |   |-- stars: long (nullable = true)
|   |   |-- tags: array (nullable = true)
|   |   |   |-- element: string (containsNull = true)
|   |   |-- url: string (nullable = true)
|   |   |-- user: string (nullable = true)
|   |   |-- user_id: string (nullable = true)
|-- urls: struct (nullable = true)
|   |-- domain: string (nullable = true)
|   |-- domains: array (nullable = true)
|   |   |-- element: string (containsNull = true)
|   |-- email: string (nullable = true)
|   |-- url: string (nullable = true)

```

```

|     |-- urls: array (nullable = true)
|     |     |-- element: string (containsNull = true)
|-- webpage: struct (nullable = true)
|     |-- content: string (nullable = true)
|     |-- count: long (nullable = true)
|     |-- elapsed: double (nullable = true)
|     |-- success: boolean (nullable = true)
|     |-- timestamp: string (nullable = true)
|     |-- title: string (nullable = true)
|     |-- url: string (nullable = true)
|     |-- urlhash: string (nullable = true)
|     |-- validurl: string (nullable = true)

```

3. (1 PT) Show the first 5 records

hint: reaching deeper fields in json hierarchy can be done like this:

```
df.select('address.street_number')
```

[9]: `df.show(5)`

```

+-----+-----+-----+-----+
+---+-----+-----+-----+-----+
|          address|    business_tags|          hours|
id|menu|          reviews|          urls|          webpage|
+-----+-----+-----+-----+
+---+-----+-----+-----+
|[Woodburn, [45.15...|          null|
null|000023995a540868|null|
[]|[woodburn.k12.or...|[Educational Tech...|
|[Hialeah, [25.884...|[], [[has_atm, Y...|[, 1900,, 0830,
3...|0000821a1394916e|null|          null|[, [yelp.com],,, ...|
null|
|[Rochester, [43.1...|[], [[accepts_cr...|[, 1700,, 0830,
3...|000136e65d50c3b7|null|[[New (to me) qui...|[usps.com, [yelp...|[Welcome |
USPS G...|
|[West Palm Beach,...|          null|
null|00014329a70b9869|null|          null|          null|
null|
|[Eufaula, [35.283...|          null|[, 1700,, 0830,
3...|00031c0a83f00657|null|          null|[drsodomcoburnand...|[DRS.
COBURN, RIC...|
+-----+-----+-----+-----+
+---+-----+-----+-----+
only showing top 5 rows

```

4. (1 PT) Location

Count the number of records where the city is Houston

```
[10]: df.filter(df.address.city == 'Houston').count()
```

```
[10]: 1668
```

5. (1 PT) Hours

Count the number of records where closing time on Friday is 7pm

```
[11]: df.filter(df.hours.friday_close == '1900').count()
```

```
[11]: 3305
```

6. (1 PT) Location and Hours

Count the number of records where city is Houston and closing time on Friday is 7pm

```
[29]: df.filter((df.address.city == 'Houston') & (df.hours.friday_close == '1900')).  
      ↪count()
```

```
[29]: 42
```

7. (1 PT) Price Range

Price range is quoted in number of dollar signs. Count the number of records with price range greater than or equal to three.

```
[12]: df.filter(df.menu.price_range >= 3).count()
```

```
[12]: 115
```

8. (1 PT) COMPANY HEADQUARTERS

Show the distribution of the `address.is_headquarters` field:
how many locations are HQ / are NOT HQ / are null?

```
[13]: df.groupby('address.is_headquarters')  
      df.filter(df.address.is_headquarters == True).count()
```

```
[13]: 318
```

There are 318 business with the location being the HQ.

```
[14]: df.filter(df.address.is_headquarters == False).count()
```

```
[14]: 66736
```

There are 66736 business with the location *not* being the HQ.

```
[15]: df.filter(df.address.is_headquarters.isNull()).count()
```

```
[15]: 87625
```

There are 87625 business that left the field *Null*.

9. (1 PT) Webpage URLs

Register the dataframe as a temp table.

Next, use Spark SQL to select the webpage title where the webpage url (accessed under `webpage.url`) is *Target.com*.

Show the first record from your query, using `show(1, False)` to show the full text from the first record.

```
[16]: df.createOrReplaceTempView('dff')

sqlTarget = spark.sql("SELECT webpage.title FROM dff where webpage.url == 'Target.com'")
sqlTarget.show()
```

```
+-----+
|          title|
+-----+
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
|Target : Expect M...|
+-----+
```

10. (1 PT) Analysis on Ratings

The reviews contains information such as the number of stars for each review (the *rating*).

The ratings are stored in an array (`reviews.stars`) for each business location (you should check for yourself). Return the top five most common rating arrays. For example, an array might look like: `[5, 5]`

```
[17]: from pyspark.sql.functions import desc
sqlx = spark.sql("SELECT reviews.stars FROM dff")
```

```
sqlx.groupBy("stars").count().sort(desc("count")).show(5)
```

```
+-----+-----+
| stars|count|
+-----+-----+
|  null|74679|
|    []|42419|
|   [5]| 4258|
|    []| 3067|
|[5, 5]| 1610|
+-----+-----+
```

only showing top 5 rows

11. More work with Ratings

For this question, you will filter out null ratings and then compute the average rating for each business location (using the field: id).

a) (1 PT) Create a new dataframe retaining two fields: id, reviews.stars

```
[18]: sql = spark.sql("SELECT id, reviews.stars from dff")
      sql.show()
```

```
+-----+-----+
|          id|          stars|
+-----+-----+
|000023995a540868|      []|
|0000821a1394916e|     null|
|000136e65d50c3b7|    [4, 4]|
|00014329a70b9869|     null|
|00031c0a83f00657|     null|
|0003b7589a4e12a0|     [5]|
|00045f958e4bb02a|    [, , ,]|
|00055eb14ea38ca2|     null|
|00059519f0dba1b4|[, , , , , , , 1, 5, 2...|
|0005f8f0b2beeac4|     null|
|000673b2dac411ea|     null|
|0006d5aa170bae22|      []|
|00080a1e9b3607c0|     null|
|0008bc70f8ba62bf|      []|
|0008c39401d7ce3b|     null|
|0009cd9a574331a9|     null|
|000a1df4c8e0ecd2|    [, , 4, 5, 5, 4, 5]|
|000a863f1d02dd78|     null|
|000b4a9059161ee7|     null|
|000bf1e934ac9cb6|      []|
+-----+-----+
```

only showing top 20 rows

- b) (1 PT) Create a row for each rating
 hint: use the `withColumn()` and `explode()` functions
 you will need to import the `explode()` function by issuing:

```
from pyspark.sql.functions import explode
```

```
[24]: from pyspark.sql.functions import explode
a = sqly.withColumn('rating', explode(sqly.stars))
a.show()
```

```
+-----+-----+-----+
|          id|          stars|rating|
+-----+-----+-----+
|000136e65d50c3b7|          [4, 4]|      4|
|000136e65d50c3b7|          [4, 4]|      4|
|0003b7589a4e12a0|           [5]|      5|
|00045f958e4bb02a|          [,,]|    null|
|00045f958e4bb02a|          [,,]|    null|
|00045f958e4bb02a|          [,,]|    null|
|00045f958e4bb02a|          [,,]|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|    null|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|      1|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|      5|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|      2|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|      4|
|00059519f0dba1b4|[,,,,,,, 1, 5, 2...|      5|
+-----+-----+-----+
```

only showing top 20 rows

- c) (1 PT) Return a count of the number of ratings in this dataframe

```
[76]: a.count()
```

```
[76]: 600082
```

- d) (1 PT) Drop rows where the rating is null, and return a count of the number of non-null ratings

```
[72]: a.filter(a.rating.isNotNull()).count()
```


[72]: 538241

- e) (1 PT) Compute the average rating, grouped by `id`. After the average is computed, sort by `id` in ascending order and show the top 10 records.

hint:

this can all be done in one line using the `agg()` function

this `id` should be at the top: 000136e65d50c3b7|

```
[80]: from pyspark.sql import functions as F
a.groupby(a.id).agg(F.avg(a.rating)).orderBy(a.id).show(10)
```

```
+-----+-----+
|          id|    avg(rating)|
+-----+-----+
|000136e65d50c3b7|          4.0|
|0003b7589a4e12a0|          5.0|
|00045f958e4bb02a|         null|
|00059519f0dba1b4|3.333333333333335|
|0008bc70f8ba62bf|         null|
|000a1df4c8e0ecd2|          4.6|
|000c7b7a30623083|          5.0|
|000c9ffc8b89af03|          3.0|
|000de20baa847ecc|1.6666666666666667|
|001064359d9f162f|          5.0|
+-----+-----+
```

only showing top 10 rows