databricksHW_04

DSCI 417 –Homework 04

Sean Graham

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr

spark = SparkSession.builder.getOrCreate()
```

Load Diamond Data

```
my_schema = 'carat FLOAT, cut STRING, color STRING, clarity STRING, depth FLOAT, table FLOAT, price INTEGER, x FLOAT, y FLOAT, z FLOAT'

diamonds = spark.read.option('delimiter', '\t').option('header', True).schema(my_schema).csv('/FileStore/tables/diamonds.txt')

diamonds.printSchema()
```

```
root
 |-- carat: float (nullable = true)
 |-- cut: string (nullable = true)
 |-- color: string (nullable = true)
 |-- clarity: string (nullable = true)
 |-- depth: float (nullable = true)
 |-- table: float (nullable = true)
 |-- price: integer (nullable = true)
 |-- x: float (nullable = true)
```

```
 |-- y: float (nullable = true)
 |-- z: float (nullable = true)
```

Problem 1: Grouping By Cut

```
def rank_cut(level):
    numbers = {'Fair': 1, 'Good': 2, 'Very Good': 3, 'Premium': 4, 'Ideal': 5}
    return numbers[str(level)]

spark.udf.register('rank_cut', rank_cut)
```

```
Out[3]: <function __main__.rank_cut(level)>
```

```
(
    diamonds
    .groupBy('cut')
    .agg(
        expr('COUNT(*) AS n_diamonds'),
        expr('INT(MEAN(price)) AS avg_price'),
        expr('ROUND(MEAN(carat),2) AS avg_carat'),
        expr('ROUND(MEAN(depth),2) AS avg_depth'),
        expr('ROUND(MEAN(table),2) AS avg_table')
    )
    .sort(expr('rank_cut(cut)'), ascending=True)
    .show()
)
```

```
+---------+----------+---------+---------+---------+---------+
|      cut|n_diamonds|avg_price|avg_carat|avg_depth|avg_table|
+---------+----------+---------+---------+---------+---------+
|     Fair|      1610|     4358|     1.05|    64.04|    59.05|
|     Good|      4906|     3928|     0.85|    62.37|    58.69|
```

```
|Very Good|     12082|     3981|     0.81|    61.82|    57.96|
|  Premium|     13791|     4584|     0.89|    61.26|    58.75|
|    Ideal|     21551|     3457|      0.7|    61.71|    55.95|
+---------+----------+---------+---------+---------+---------+
```

Problem 2: Filtering based on Carat Size

```
for i in range(0, 6):
    bin = diamonds.filter((col('carat') >= i) & (col('carat') < (i+1))).count()
    print('The number of diamonds with carat size in range [' + str(i) + ', ' + str(i+1) + ') is ' + str(bin) + '.')
```

```
The number of diamonds with carat size in range [0, 1) is 34880.
The number of diamonds with carat size in range [1, 2) is 16906.
The number of diamonds with carat size in range [2, 3) is 2114.
The number of diamonds with carat size in range [3, 4) is 34.
The number of diamonds with carat size in range [4, 5) is 5.
The number of diamonds with carat size in range [5, 6) is 1.
```

Problem 3: Binning by Carat Size

```
def carat_bin(x):
    bins = ['[0,1)', '[1,2)', '[2,3)', '[3,4)', '[4,5)', '[5,6)']
    return bins[int(x)]

spark.udf.register('carat_bin', carat_bin)
```

```
Out[6]: <function __main__.carat_bin(x)>
```

```
(
    diamonds
    .withColumn('carat_bin', expr('carat_bin(carat)'))
    .groupBy('carat_bin')
    .agg(
        expr('COUNT(*) AS n_diamonds'),
        expr('INT(MEAN(price)) AS avg_price')
    )
    .sort('carat_bin', ascending = True)
    .show()
)
```

```
+---------+----------+---------+
|carat_bin|n_diamonds|avg_price|
+---------+----------+---------+
|    [0,1)|     34880|     1632|
|    [1,2)|     16906|     7288|
|    [2,3)|      2114|    14846|
|    [3,4)|        34|    14308|
|    [4,5)|         5|    16458|
|    [5,6)|         1|    18018|
+---------+----------+---------+
```

Load IMDB Data

```
movies = (spark.read.option('delimiter', '\t').option('header', True).option('inferSchema', True).csv('/FileStore/tables/imdb/movies.txt'))
movies.printSchema()
```

```
root
 |-- imdb_title_id: string (nullable = true)
 |-- title: string (nullable = true)
```

```
 |-- year: integer (nullable = true)
 |-- genre: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- country: string (nullable = true)
 |-- language: string (nullable = true)
```

```
names = (spark.read.option('delimiter', '\t').option('header', True).option('inferSchema', True).csv('/FileStore/tables/imdb/names.txt'))
names.printSchema()
```

```
root
 |-- imdb_name_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- birth_name: string (nullable = true)
 |-- height: string (nullable = true)
 |-- bio: string (nullable = true)
 |-- date_of_birth: string (nullable = true)
 |-- date_of_death: string (nullable = true)
```

```
title_principals = (spark.read.option('delimiter', '\t').option('header', True).option('inferSchema', True).csv('/FileStore/tables/imdb/title_principals.txt'))
title_principals.printSchema()
```

```
root
 |-- imdb_title_id: string (nullable = true)
 |-- ordering: integer (nullable = true)
 |-- imdb_name_id: string (nullable = true)
 |-- category: string (nullable = true)
 |-- characters: string (nullable = true)
```

```
ratings = (spark.read.option('delimiter', '\t').option('header', True).option('inferSchema', True).csv('/FileStore/tables/imdb/ratings.txt'))
ratings.printSchema()
```

```
root
```

```
 |-- imdb_title_id: string (nullable = true)
 |-- rating: double (nullable = true)
 |-- total_votes: integer (nullable = true)
```

```
print(movies.count())
print(names.count())
print(title_principals.count())
print(ratings.count())
```

```
85855
297710
835513
85855
```

Problem 4: Number of Appearances by Actor

```
(
    title_principals
    .filter(expr('category == "actor" OR category == "actress"'))
    .groupBy('imdb_name_id')
    .agg(
        expr('COUNT(*) AS appearances')
    )
    .join(other=names, on='imdb_name_id', how='left')
    .select('name', 'appearances')
    .sort('appearances', ascending = False)
    .show(16)
)
```

```
+-----------------+-----------+
|             name|appearances|
+-----------------+-----------+
```

```
|         Mohanlal|        163|
| Amitabh Bachchan|        142|
|        Mammootty|        140|
|     Eric Roberts|        133|
|       John Wayne|        132|
| Gérard Depardieu|        130|
|      Prakash Raj|        125|
|     Akshay Kumar|        115|
|   Michael Madsen|        107|
|         Andy Lau|        102|
|Catherine Deneuve|        101|
|      Anupam Kher|         99|
|     Brahmanandam|         99|
|       Ajay Devgn|         94|
|    Michael Caine|         94|
|  Christopher Lee|         93|
+-----------------+-----------+
only showing top 16 rows
```

Problem 5: Average Rating by Director

```
(
    title_principals
    .filter(expr('category == "director"'))
    .join(other=ratings, on='imdb_title_id', how='left')
    .groupBy('imdb_name_id')
    .agg(
        expr('COUNT(*) AS num_films'),
        expr('SUM(total_votes) AS total_votes'),
        expr('ROUND(MEAN(rating), 2) AS avg_rating')
    )
    .filter(expr('total_votes >= 1000000'))
    .filter(expr('num_films >= 5'))
    .join(other=names, on='imdb_name_id', how='left')
    .select('name', 'num_films', 'total_votes', 'avg_rating')
    .sort('avg_rating', ascending = False)
    .show(16, truncate=False)
)
```

```
+--------------------+---------+-----------+----------+
|name                |num_films|total_votes|avg_rating|
+--------------------+---------+-----------+----------+
|Christopher Nolan   |11       |11653144   |8.22      |
|Lee Unkrich         |5        |3329612    |8.14      |
|Hayao Miyazaki      |12       |2254496    |8.01      |
|Quentin Tarantino   |14       |9460772    |7.93      |
|Sergio Leone        |7        |1720654    |7.93      |
|Stanley Kubrick     |13       |4232356    |7.78      |
|David Fincher       |10       |6944421    |7.76      |
|Sam Mendes          |10       |3067512    |7.73      |
|Alejandro G. Iñárritu|7        |2067540    |7.61      |
|Wes Anderson        |9        |2173090    |7.61      |
|Peter Jackson       |13       |7304418    |7.58      |
|Brad Bird           |6        |2294748    |7.57      |
|Alfonso Cuarón      |8        |2078975    |7.54      |
```

```
|Andrew Stanton       |5        |2649551    |7.52      |
|Akira Kurosawa       |32       |1061519    |7.51      |
|Bong Joon Ho         |8        |1172684    |7.51      |
+---------------------+---------+-----------+----------+
```

Problem 6: Actors Appearing in Horror Films

```
horror_films = movies.filter(expr('genre LIKE "%Horror%"'))
print(horror_films.count())
```

```
9557
```

```
+----------------+---------+
|            name|num_films|
+----------------+---------+
| Christopher Lee|       56|
|   Peter Cushing|       47|
|   Boris Karloff|       46|
|  John Carradine|       43|
|     Bela Lugosi|       38|
|   Vincent Price|       34|
| Lance Henriksen|       33|
|    Eric Roberts|       29|
|  Lon Chaney Jr.|       28|
|       Tony Todd|       27|
|    Bill Moseley|       27|
|     Paul Naschy|       26|
|Donald Pleasence|       26|
|  Robert Englund|       23|
|     Brad Dourif|       23|
|       Sergey A.|       23|
+----------------+---------+
only showing top 16 rows
```