به‌نام خداوند بخشنده و مهربان

تمرین دوم: کار با داده‌های حجیم
درس: تحلیل سیستم داده‌های حجیم

استاد: محمدعلی نعمت‌بخش
دستیاران: فاطمه ابراهیمی، پریسا لطیفی، امیر سرتیپی

نام و نام‌خانوادگی: زلفا شفرئی
آدرس گیت: https://github.com/zolfaShefreie/spark_dataframes

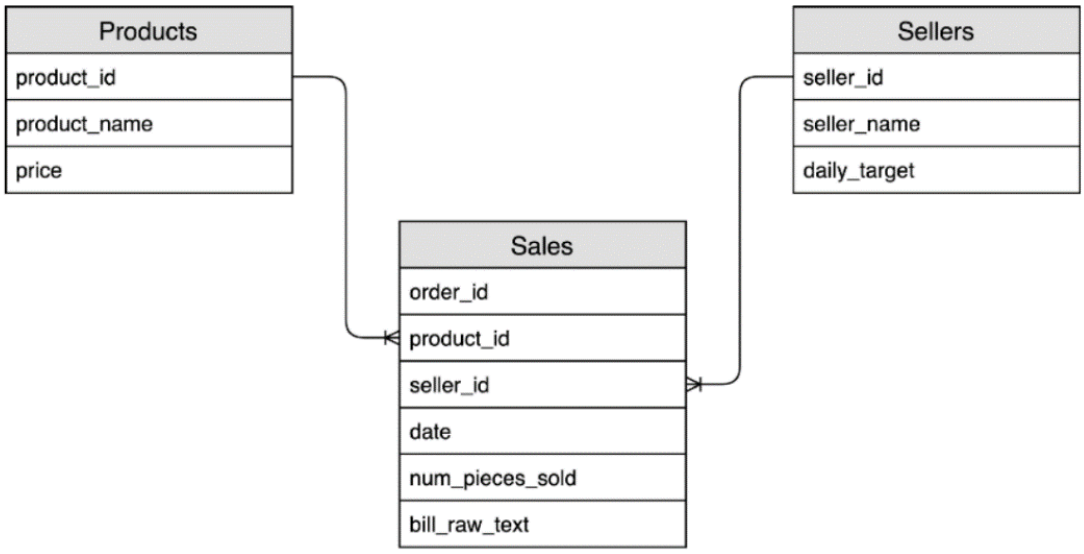
- **لطفا پاسخ تمارین حتما در سامانه‌ی کوئرا ارسال شود.**
- **لطفا پاسخ‌های خود را در خود سند سوال نوشته و در قالب یک فایل PDF ارسال کنید.**
- **نام سند ارسالی HW-{homework number}-{Name Family}-{student number}**
- **تمامی فایل‌های مورد نیاز این تمرین در این لینک قابل دسترس است.**
- **خروجی از هر مرحله‌ی تمرین را در سند خود بارگذاری کنید.**
- **کد + سند را در گیت بارگذاری کرده و لینک آن را در سند قرار دهید.**
- **لینک نوتبوک و مجموعه‌ی داده**

در این تمرین هدف ما آشنایی با دیتافریم‌ها و کار با داده‌های حجیم در موتور تحلیل spark است.

برای این منظور در ابتدا فایل دیتاست را به کمک قطعه کدی که در فایل نوت بوکی که در ادامه در اختیار شما قرار گرفته است، در دسترس خواهید داشت و سپس با توجه به مجموعه داده‌های در دسترس خود با کمک زبان برنامه نویسی پایتون به سوالات مطرح شده در قسمت مربوط به همان سوال پاسخ دهید.

مجموعه داده مورد استفاده در این تمرین، از پایگاه داده یک فروشگاه، که شامل اطلاعاتی در رابطه با محصولات، فروش و فروشندگان، تشکیل شده است. نمودار رابطه موجودیت این مجموعه داده که در شکل-۱ نمایش داده می‌شود، هر کدام شامل فیلدهای زیر می‌باشند:

- ✓ محصولات (products): {کد محصول (product_id)، نام محصول (product_name)، قیمت (price)}
- ✓ فروشنده (Sellers): {کد فروشنده (seller_id)، نام فروشنده (seller_name)، مقدار فروش روزانه هر فروشنده (daily_target)}
- ✓ فروش محصولات (سفارشات): {کد سفارش (order_id)، کد محصول (product_id)، کد فروشنده (seller_id)، تاریخ (date)، تعداد محصولات فروخته شده (num_pieces_sold)، متن صورتحساب (bill_raw_text)}



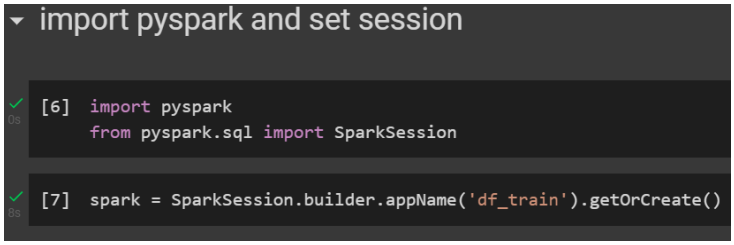
فایل فشرده این مجموعه داده در لینک زیر قابل دسترس خواهد بود که با کمک دستورات برنامه نویسی در محیط گوگل کولب فراخوانی شده و در گام اول از حالت فشرده خارج می‌شود تا بتوان به هر کدام از این جداول به طور مجزا دسترسی داشت.

سپس داده‌های هرکدام از جداول را بررسی کرده و از آن‌ها برای پاسخگویی به سوالات مطرح شده استفاده کنید.

ابتدا برای دسترسی به مجموعه‌داده، یک shourtcut از آن در drive خود ساخته شده است که می‌توان در google colab با استفاده از google.colab به آن دسترسی داشت. پس از آن نسخه زیپ فایل مجموعه‌داده از حالت فشرده خارج می‌شود.

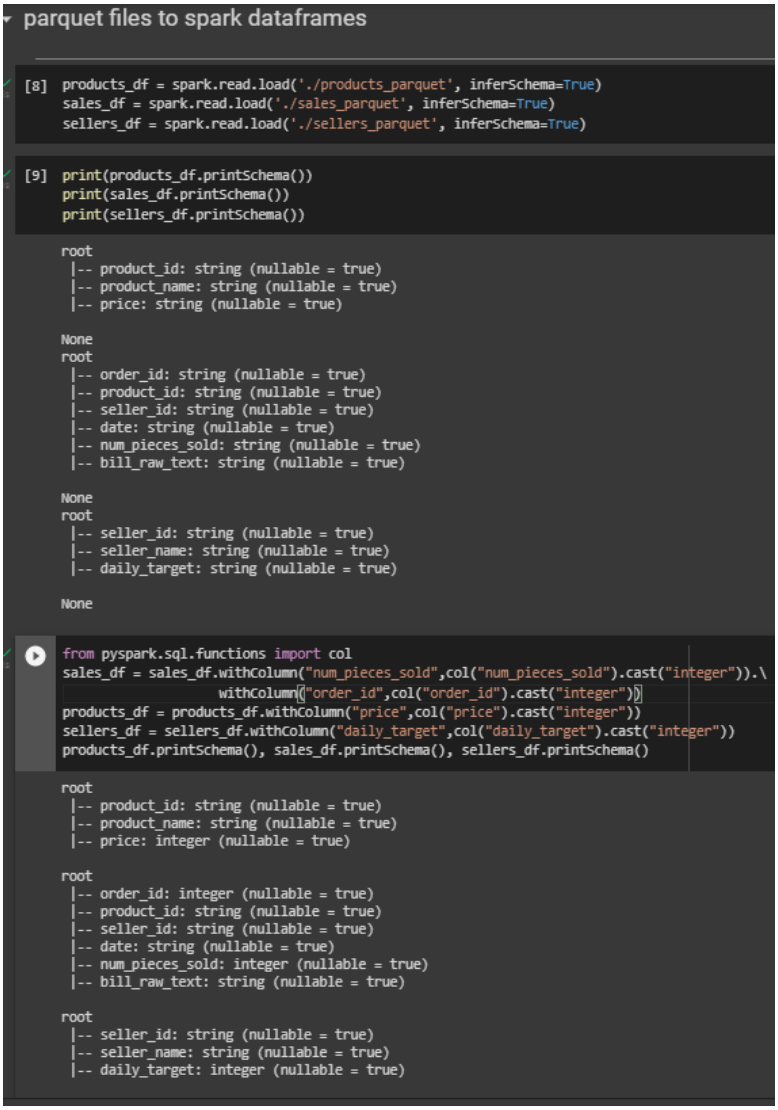
```
adding the dataset to my dirve and mounting here

[3]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

[4]  dataset_path = './drive/MyDrive/big_data/HW2_dataset.zip'

[5]  import zipfile
     with zipfile.ZipFile(dataset_path, 'r') as zip_ref:
         zip_ref.extractall("./")
```

شکل ۱: دسترسی به مجموعه‌داده



شکل ۲: ساخت سشن pyspark

سپس فایل‌های parquest به دیتافریم‌هایی از اسپارک تبدیل می‌شوند که با استفاده از دستور read.load صورت می‌گیرد.پس از لود در شمای دیتافریم‌ها مشاهده می‌شود که تمامی ستون‌ها به‌صورت string می‌باشد و حتی inferSchema اثرگذار نبوده است. نوع چندین ستون که در محاسبات سوالات به‌آن‌های نیاز داریم به integer تبدیل می‌شود که برای این کار دستور cast برای تبدیل و withColumn برای جایگذاری مقادیر در ستون استفاده می‌گردد.



شکل ۳: ساخت دیتافریم‌ها و تبدیل برخی از ستون

پاسخ سوالات به صورت زیر می‌باشد.

سوال ۱)

الف) تعداد سفارشات، تعداد محصولات و تعداد فروشندگان ذخیره شده در دیتاست را بدست آورید.

ب) تعداد محصولاتی که حداقل یکبار به فروش رسیده‌اند را بدست آورید.

ج) کدام یک از محصولات به فروش رسیده، بیشترین تکرار در سفارش‌ها را دارد؟

قسمت الف) تعداد سطرهای یک دیتافریم با استفاده از تابع count در دسترس است.

```
print(f"Products: {products_df.count()}")
print(f"Sales: {sales_df.count()}")
print(f"Sellers: {sellers_df.count()}")
```

```
Products: 75000000
Sales: 20000040
Sellers: 10
```

شکل ۴: پاسخ قسمت الف سوال ۱

قسمت ب) با استفاده از دستور distinct می‌توان موارد یونیک را برای محصولات در دیتافریم مربوط به سفارشات انتخاب کرد و با استفاده از تابع count آن‌ها را شمارد.

```
sales_df.select("product_id").distinct().count()
```

```
993429
```

شکل ۵: پاسخ قسمت ب سوال ۱

قسمت ج) برای بیشترین تکرار محصولات در سفارشات، یک گروه‌بندی با استفاده از تابع groupBy براساس شناسه‌ی محصولات انجام شده و با استفاده از count تعداد در هر گروه شمارش شده است. برای بدست آوردن بیشترین نیاز به sort کردن نتیجه به‌صورت نزولی می‌باشد که با استفاده از تابع limit پنج تا از بیشترین تکرار را انتخاب شده است. در آخر با استفاده از join و show اطلاعات بیشتری از از محصولات نمایش داده شده است.

```
from pyspark.sql.functions import desc
top_products = sales_df.groupBy("product_id").count().sort(desc("count")).limit(5)
top_products.join(products_df, ['product_id']).show()
```

```
+----------+--------+---------------+-----+
|product_id|   count|   product_name|price|
+----------+--------+---------------+-----+
|         0|19000000|      product_0|   22|
|   2316238|       3| product_2316238|   19|
|  36269838|       3|product_36269838|   87|
|  28592106|       3|product_28592106|   76|
|  31136332|       3|product_31136332|  149|
+----------+--------+---------------+-----+
```

شکل ۶: پاسخ قسمت ج سوال ۱

سوال ۲)

چند محصول متمایز در هر روز به فروش می‌رسد؟

برای پیدا کردن محصول متمایز سطرهای تکراری بر اساس date و product_id حذف شده‌اند. با استفاده از گروه‌بندی بر اساس تاریخ و تابع count به تعداد محصولات فروخته‌شده‌ی یکتا دسترسی داشت.

```
sales_df.dropDuplicates(['product_id', 'date']).groupBy('date').count().show()
```

```
+----------+------+
|      date| count|
+----------+------+
|2020-07-03|100017|
|2020-07-07| 99756|
|2020-07-01|100337|
|2020-07-08| 99662|
|2020-07-04| 99791|
|2020-07-10| 98973|
|2020-07-09|100501|
|2020-07-06|100765|
|2020-07-02| 99807|
|2020-07-05| 99796|
+----------+------+
```

شکل ۷: پاسخ سوال ۲

سوال ۳)

میانگین درآمد سفارشات در این دیتاست را محاسبه کنید.

برای بدست آوردن درآمد هر سفارشات باید قیمت هر کالا را با تعداد فروخته شده در هر سفارش ضرب کرد. برای اینکار نیاز به جوین کردن دو دیتافریم سفارشات و محصولات است. پس از آن با استفاده از withColumn یک ستون جدید ساخته می‌شود تا درآمد یک سفارش در آن ذخیره شود. با استفاده از agg بر روی این درآمدها میانگین گرفته می‌شود.

```
sales_df.join(products_df, sales_df.product_id==products_df.product_id).\
withColumn("total_price", col("price") * col("num_pieces_sold")).\
agg({'total_price': 'mean'}).show()
```

```
+------------------+
|  avg(total_price)|
+------------------+
|1246.1338560822878|
+------------------+
```

شکل ۸: پاسخ سوال ۳

سوال ٤)

به ازای هر فروشنده، میانگین درصد سهم یک سفارش در سهمیه روزانه فروشندگان چقدر است؟

(به عنوان مثال می‌توانیم بین جدول فروشنده و همچنین جدول فروش که نمایانگر ارتباط بین سفارشات، محصولات و فروشندگان می‌باشد، ارتباط برقرار کرده و سپس مقدار درصد سهمیه را برای هر سفارش خاص محاسبه کرده و پس از محاسبه میانگین سهمیه سفارش محصولات، مقدار بدست آمده در خروجی را براساس شماره فروشنده (seller_id) گروه‌بندی کنید.)

برای حل این سوال به دو صورت می‌توان پیش رفت:

- اگر daily_target را برای سهمیه‌ی روزانه‌ی فروشندگان در نظر گرفته شود تنها نیاز است درآمد در هر سفارش حساب شود و درصد آن بر اساس سهمیه روزانه فروشنده‌ی سفارش محاسبه و براساس فروشنده گروه‌بندی و میانگین گرفته شود. (طبق شکل ۹)
- بدون استفاده از daily_target باید درآمد فروشنده در بر اساس تاریخ محاسبه شود و براساس آن درصد سهمیه سفارش در آن روز حساب می‌شود. سپس براساس فروشنده گروه‌بندی و میانگین گرفته شود. (طبق شکل ۱۰)

calculate with **daily_target**

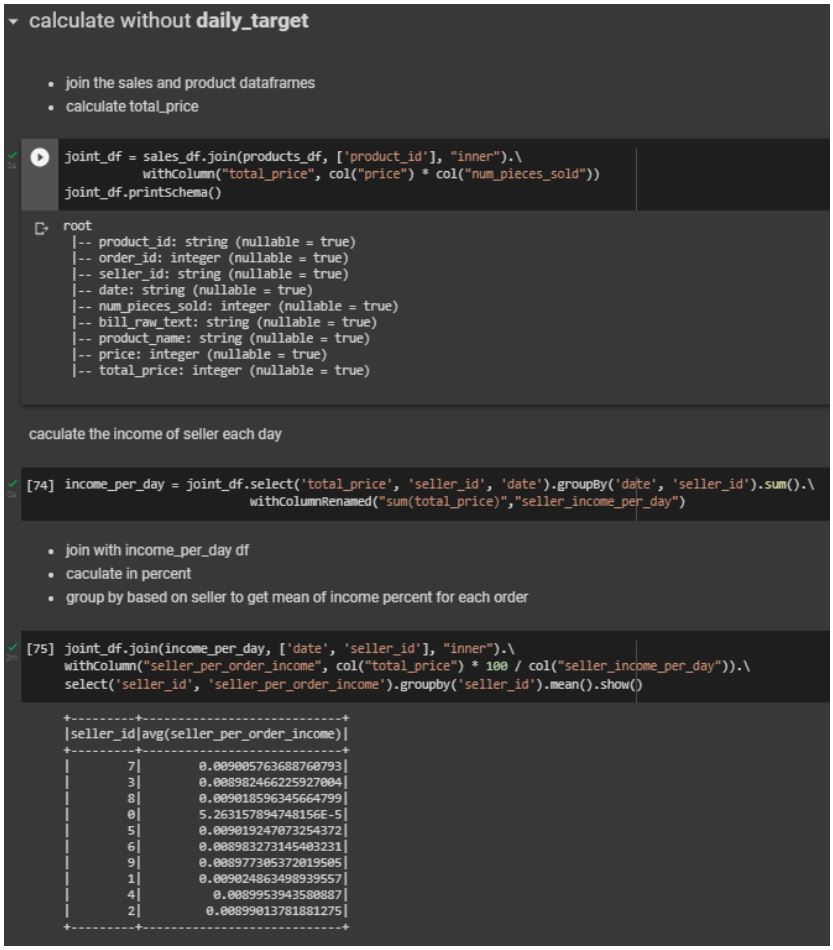
```
[69] joint_df = sales_df.join(products_df, ['product_id'], "inner").\
                withColumn("total_price", col("price") * col("num_pieces_sold")).\
                select('seller_id', 'order_id', 'date', 'total_price').join(sellers_df, ['seller_id'])
     joint_df.printSchema()
```

```
root
 |-- seller_id: string (nullable = true)
 |-- order_id: integer (nullable = true)
 |-- date: string (nullable = true)
 |-- total_price: integer (nullable = true)
 |-- seller_name: string (nullable = true)
 |-- daily_target: integer (nullable = true)
```

```
joint_df.withColumn('share_percent', col('total_price')*100/col('daily_target')).\
         select('seller_id', 'share_percent').groupBy('seller_id').mean().show()
```

```
+---------+-------------------+
|seller_id| avg(share_percent)|
+---------+-------------------+
|        7| 0.1960124630631757|
|        3| 1.2318678193054804|
|        8|  0.694606099856396|
|        5| 0.3170589299015149|
|        6|0.36093852517481845|
|        9| 0.2905299815673646|
|        1| 1.4844178645806256|
|        4|0.24841173704802313|
|        2| 0.5064829818617168|
|        0| 0.0444374897765487|
+---------+-------------------+
```

شکل ۹: پاسخ سوال ۴ با استفاده از daily_target



شکل ۱۰: پاسخ سوال ۴ بدون استفاده از daily_target

سوال ۵)

الف) دومین پرفروش‌ترین فروشنده و همچنین دومین کم فروش‌ترین را در بین فروشندگان بیابید.

ب) کدام فروشندگان محصول "product_id = 0" را بیابید.

توجه:

- ✓ در حین بررسی ممکن است به محصولی برخوردکنید که تنها توسط یک فروشنده به فروش رسیده باشد، در نتیجه این محصول به عنوان یک گروه یک گروه مجزا درنظر گرفته می‌شود.
- ✓ به عنوان مثال ممکن است، "product_0"، توسط بیش از یک فروشنده به فروش رسیده باشد ولی همه فروشندگان به مقدار مساوی از این محصول را فروخته‌اند، بنابراین همه فروشندگان را در یک گروه قرار داده و فرض می‌کنیم این محصول فقط توسط یک فروشنده به فروش رسیده است.
- ✓ حتی ممکن است در این بررسی فروشنده با کمترین میزان فروش، همان دومین فروشنده باشد، آنگاه این فروشنده به‌عنوان دومین فروشنده با کمترین میزان فروش معرفی می‌شود.

قسمت الف) برای یافتن پرفروش‌ترین و کم‌فروش‌ترین براساس تعداد فروخته شده از محصولات محاسبه می‌شود. برای این کار بر اساس فروشنده، دیتافریم سفارشات گروه‌بندی می‌شود و مجموع تعداد محصولات سفارش شده محاسبه می‌شود.

طبق صورت سوال درصورتی که دو فروشنده تعدادی یکسانی سفارش گرفته باشند در یک گروه قرار می‌گیرند پس نیاز به یک رتبه‌بندی براساس تعداد محصولات سفارش شده می‌باشد. برای اینکار از Window برای رتبه‌بندی استفاده می‌کنیم که رتبه‌ها بر اساس ترتیب تعداد محصولات سفارش شده قرار می‌گیرند. با استفاده از dense_rank().over(window) به هر یک از گروه‌ها براساس تعداد محصولات سفارش شده یک رتبه در ستون rank زده می‌شود. سپس با کمک where دومین پرفروش‌ترین و دومین کم‌فروش‌ترین پیدا خواهد شد.

```
[136] from pyspark.sql.window import Window
      from pyspark.sql.functions import row_number, dense_rank, when

[151] count_window = Window.orderBy(col('sum(num_pieces_sold)'))

[149] seller_rank_sum = sales_df.select('num_pieces_sold', 'seller_id').groupBy('seller_id').sum().\
                              withColumn('rank', dense_rank().over(count_window))
      max_value = seller_rank_sum.agg({'rank': 'max'}).first()['max(rank)']
      max_value

      10

      seller_rank_sum.where((col('rank') == 2) | (col('rank') == (max_value - 1))).show()

      +---------+--------------------+----+
      |seller_id|sum(num_pieces_sold)|rank|
      +---------+--------------------+----+
      |        1|             5598683|   2|
      |        9|             5634837|   9|
      +---------+--------------------+----+
```

شکل ۱۱: پاسخ سوال ۵ الف

قسمت ب) طبق بررسی‌های صورت گرفته شده هرکدام از محصولات فروخته شده تنها توسط یک فروشنده به فروش رسیده‌اند. این مسئله برای محصول با شناسه صفر نیز صادق است که تنها توسط فروشنده با شناسه‌ی صفر فروخته شده است.

```
sales_df.select('num_pieces_sold', 'seller_id', 'product_id').groupBy('seller_id', 'product_id').sum().show()

+---------+----------+--------------------+
|seller_id|product_id|sum(num_pieces_sold)|
+---------+----------+--------------------+
|        0|         0|           959445802|
|        6|  71598950|                  45|
|        9|  40745956|                  28|
|        4|  70025887|                  58|
|        9|  14623915|                  68|
|        9|  57841735|                  51|
|        9|  55876917|                  11|
|        3|  61166403|                  79|
|        3|  26750979|                  79|
|        4|   7475511|                  62|
|        5|  58722902|                  13|
|        6|  25666047|                  55|
|        8|  48878702|                  76|
|        6|  72166578|                  90|
|        6|  15442804|                  55|
|        4|   1427289|                  93|
|        3|  10498036|                  78|
|        5|  12714089|                  53|
|        5|  35669585|                  13|
|        9|  65272164|                  91|
+---------+----------+--------------------+
only showing top 20 rows
```

شکل ۱۲: بررسی کلی‌تر بر اساس هر محصول

اما برای یافتن کم فروش‌ترین و پرفروش‌ترین بدون درنظر گرفتن مسئله‌ی بالا می‌توان از دو پنجره با پارتیشن‌بندی product_id و رتبه‌بندی به‌صورت صعودی و نزولی استفاده کرد. پارتیشن‌بندی باعث می‌شود در هر پارتیشن رتبه‌بندی جدا فرض شود. در این مسئله با استفاده از where می‌توان رتبه‌بندی‌ها برای هر محصول را پیدا کرد. در این مسئله دومین پرفروش و کم‌فروش برای شناسه‌ی محصول صفر وجود ندارد.

```
count_window = Window.partitionBy('product_id').orderBy(col('sum(num_pieces_sold)'))
count_window_desc = Window.partitionBy('product_id').orderBy(desc('sum(num_pieces_sold)'))
```

```
sales_df.select('num_pieces_sold', 'seller_id', 'product_id').groupBy('seller_id', 'product_id').sum().\
withColumn('rank', dense_rank().over(count_window)).\
withColumn('rank_desc', dense_rank().over(count_window_desc)).\
where('product_id==0').\
where(((col('rank') == 2) |  (col('rank_desc') == 2))).show()
```

```
+---------+----------+--------------------+----+---------+
|seller_id|product_id|sum(num_pieces_sold)|rank|rank_desc|
+---------+----------+--------------------+----+---------+
+---------+----------+--------------------+----+---------+
```

شکل ۱۳: پاسخ سوال ۵ قسمت ب

سوال ٦)

در این قسمت ستونی به نام "hashed_bill" ایجاد کنید که به صورت زیر تعریف می‌شود:

- ✓ اگر شماره سفارش زوج (order_Id) باشد: تابع رمزنگار (Hash Function)، MD5 را به صورت متوالی روی قسمت "bill_raw_text" یک بار برای هر مقدار "A" موجود در متن اعمال کنید. (به عنوان مثال اگر متن صورتحساب به صورت "nbAAnllA" باشد، تابع hashing سه بار تکرار می‌شود.)

- ✓ اگر شماره سفارش فرد (order_id) باشد: تابع رمزنگار (Hash Function)، SHA256 را بر روی داده‌های درج شده در ستون "bill_raw_text" اعمال کنید.

در پایان وجود و یا عدم وجود موارد تکراری در ستون جدید را بررسی کنید.

برای شروط مسئله نیاز به when در withColumn می‌باشد که پارمتر اول آن یک شرط است اگر این شرط درست باشد پارامتر دوم به‌عنوان مقدار ستون به آن اختصاص می‌یابد. Otherwise نیز تنها به‌معنای درغیراین‌صورت استفاده می‌شود. برای بخش اول یعنی هنگام زوج بودن order_id، نیاز به‌یک تابعی ست که تعداد کاراکتر «A» را بشمارد و به تعداد آن md5 را برروی متن اجرا کند این تابع با توابع در پایتون نوشته شده است و می‌توان با udf تبدیل به یک تابع اسپارک شود تا در withColumn از آن استفاده شود. در قسمت دوم یعنی هنگام فرد بودن order_id نیاز به تابع جدیدی نیست و اسپارک تابع sha2 را برای توابع هش خانواده‌ی sha قرار داده است. این تابع یک ستون و یک عدد برای مشخص نمودن هرکدام از تابع‌ها می‌گیرد که در اینجا این عدد برابر ۲۵۶ می‌باشد. هم‌چنین با استفاده از groupBy برروی ستون جدید هیچ مقادیر تکراری یافت نشده است.

```
[64] from pyspark.sql.functions import udf, when, md5, sha2, desc

     import hashlib
     def md5_hash(raw_text):
       hashed_text = raw_text
       a_count = raw_text.count('A')
       for _ in range(a_count):
         hashed_text = hashlib.md5(hashed_text.encode()).hexdigest()
       return hashed_text

     even_hash_udf = udf(lambda z: md5_hash(z))
```

+ Code + Text

```
[66] hashed_df = sales_df.withColumn('hashed_bill',
                        when(col('order_id')%2==0, even_hash_udf(col('bill_raw_text'))).\
                        otherwise(sha2(col('bill_raw_text'), 256)))
     hashed_df.select('hashed_bill').groupBy('hashed_bill').count().where('count > 1').sort(desc("hashed_bill")).show()

     +-----------+-----+
     |hashed_bill|count|
     +-----------+-----+
     +-----------+-----+
```

شکل ۱۴: پاسخ سوال ۶