RetailDataAnalyticsWithPySpark Python



Schedule

spark File Edit View: Standard Permissions Run All Clear Comments Experiment Revision history

Cmd 1

# Retail Data Analytics Using Databricks

In the Python Data Analytics Project, we have done Data Analytics for London Gift Shop (LGS), a UK-based online store on the provided transaction dataset from 01/12/2009 to 09/12/2011 to answer all the business questions. In this project, we perform the same analysis using Python/PySpark. This notebook was developed using the Microsoft Azure Databricks interface.

Cmd 2

## Importing Data

We upload the online_retail_II.csv file containing the data from Python Data Analytics project and then it is used to construct a PySpark DataFrame with the purpose of data analysis.

Cmd 3

```
# File location and type
file_location = "/FileStore/tables/online_retail_II.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
retail_df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(retail_df.limit(5))
```

▸ (3) Spark Jobs

▸ retail_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 6 more fields]

Table Data Profile

| | Invoice | StockCode | Description | Quantity | InvoiceDate | Price | Customer ID | Country |
|---|---|---|---|---|---|---|---|---|
| 1 | 489434 | 85048 | 15CM CHRISTMAS GLASS BALL 20 LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.95 | 13085 | United Kingdom |
| 2 | 489434 | 79323P | PINK CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom |
| 3 | 489434 | 79323W | WHITE CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom |
| 4 | 489434 | 22041 | "RECORD FRAME 7"" SINGLE SIZE " | 48 | 2009-12-01T07:45:00.000+0000 | 2.1 | 13085 | United Kingdom |
| 5 | 489434 | 21232 | STRAWBERRY CERAMIC TRINKET BOX | 24 | 2009-12-01T07:45:00.000+0000 | 1.25 | 13085 | United Kingdom |

Showing all 5 rows.

Command took 7.32 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 4

```
#return the column labels
retail_df.columns
```

```
Out[3]: ['Invoice',
 'StockCode',
 'Description',
 'Quantity',
 'InvoiceDate',
 'Price',
 'Customer ID',
 'Country']
```

Command took 0.04 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 5

```
#determine the type of columns
retail_df.dtypes
```

```
Out[4]: [('Invoice', 'string'),
 ('StockCode', 'string'),
 ('Description', 'string'),
 ('Quantity', 'int'),
 ('InvoiceDate', 'timestamp'),
 ('Price', 'double'),
 ('Customer ID', 'double'),
 ('Country', 'string')]
```

Command took 0.03 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 6

## Total Invoice Amount

Cmd 7

```
#filter the dataframe for two columns
from pyspark.sql.functions import *
retail_filtered_df = retail_df.filter((col("price") > 0) & (col("quantity") > 0))
display(retail_filtered_df.limit(5))
```

▸ (1) Spark Jobs

▸ retail_filtered_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 6 more fields]

Table | Data Profile

| | Invoice | StockCode | Description | Quantity | InvoiceDate | Price | Customer ID | Country |
|---|---|---|---|---|---|---|---|---|
| 1 | 489434 | 85048 | 15CM CHRISTMAS GLASS BALL 20 LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.95 | 13085 | United Kingdom |
| 2 | 489434 | 79323P | PINK CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom |
| 3 | 489434 | 79323W | WHITE CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom |
| 4 | 489434 | 22041 | "RECORD FRAME 7"" SINGLE SIZE " | 48 | 2009-12-01T07:45:00.000+0000 | 2.1 | 13085 | United Kingdom |
| 5 | 489434 | 21232 | STRAWBERRY CERAMIC TRINKET BOX | 24 | 2009-12-01T07:45:00.000+0000 | 1.25 | 13085 | United Kingdom |

Showing all 5 rows.

Command took 0.49 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 8

```
#derive a new column for invoice_amount from existing columns price and quantity
retail_filtered_df = retail_filtered_df.withColumn("invoice_amount",col("price") * col("quantity"))
```

▸ retail_filtered_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 7 more fields]

Command took 0.05 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 9

```
#cache the dataframe in a new one for later reference
invoice_df = retail_filtered_df.cache()
display(invoice_df.limit(5))
```

▸ (1) Spark Jobs

▸ invoice_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 7 more fields]

Table | Data Profile

| | Invoice | StockCode | Description | Quantity | InvoiceDate | Price | Customer ID | Country | invoice_amount |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 489434 | 85048 | 15CM CHRISTMAS GLASS BALL 20 LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.95 | 13085 | United Kingdom | 83.4 |
| 2 | 489434 | 79323P | PINK CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom | 81 |
| 3 | 489434 | 79323W | WHITE CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom | 81 |
| 4 | 489434 | 22041 | "RECORD FRAME 7"" SINGLE SIZE " | 48 | 2009-12-01T07:45:00.000+0000 | 2.1 | 13085 | United Kingdom | 100.80000000000001 |
| 5 | 489434 | 21232 | STRAWBERRY CERAMIC TRINKET BOX | 24 | 2009-12-01T07:45:00.000+0000 | 1.25 | 13085 | United Kingdom | 30 |

Showing all 5 rows.

Command took 4.81 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 10

```
#determine the total amount for each invoice
retail_summary_df = retail_filtered_df.groupBy("Invoice").sum().select("invoice", "sum(invoice_amount)").orderBy("sum(invoice_amount)")
display(retail_summary_df.limit(5))
```

▸ (1) Spark Jobs

▸ retail_summary_df: pyspark.sql.dataframe.DataFrame = [invoice: string, sum(invoice_amount): double]

Table | Data Profile

| | invoice | sum(invoice_amount) |
|---|---|---|
| 1 | 528127 | 0.19 |
| 2 | 570554 | 0.38 |
| 3 | 567869 | 0.4 |
| 4 | 539441 | 0.42 |
| 5 | 502731 | 0.42 |

Showing all 5 rows.

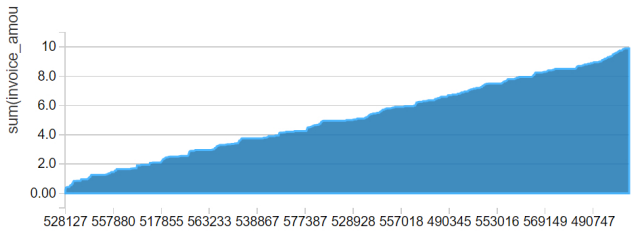
Command took 7.59 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:43 PM on spark

Cmd 11

```
#Area graph of Invoice versus sum(invoice_amount)
display(retail_summary_df)
```

▸ (2) Spark Jobs

Chart | Data Profile



Aggregated (by sum) in the backend.
Truncated results, showing first 1000 rows.

Plot Options...

Command took 1.54 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:54:23 PM on spark

Cmd 12

```
#double-check the invoice_df's columns
invoice_df.columns
```

Python

```
Out[10]: ['Invoice',
 'StockCode',
 'Description',
 'Quantity',
 'InvoiceDate',
 'Price',
 'Customer ID',
 'Country',
 'invoice_amount']
```

Command took 0.03 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 13

```
#get the summary statistics of of the column
#display(retail_summary_df.select("sum(invoice_amount)").describe())
display(retail_summary_df.select([min('sum(invoice_amount)'), avg('sum(invoice_amount)'), max('sum(invoice_amount)')]))
```

▸ (2) Spark Jobs

Table | Data Profile

| | min(sum(invoice_amount)) | avg(sum(invoice_amount)) | max(sum(invoice_amount)) |
|---|---|---|---|
| 1 | 0.19 | 523.3037611158243 | 168469.6 |

Showing all 1 rows.

Command took 5.88 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 14

```
#approximate median value
retail_summary_df.approxQuantile("sum(invoice_amount)", [0.5], 0)[0]
```

▸ (2) Spark Jobs

```
Out[12]: 304.3100000000002
```

Command took 4.96 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 15

```
#calculate the mode of the column
record_counts = retail_summary_df.groupBy("sum(invoice_amount)").count()
mode = record_counts.orderBy(desc("count")).limit(1)
display(mode)
```

▸ (2) Spark Jobs

▸ record_counts: pyspark.sql.dataframe.DataFrame = [sum(invoice_amount): double, count: long]

▸ mode: pyspark.sql.dataframe.DataFrame = [sum(invoice_amount): double, count: long]

Table | Data Profile

| | sum(invoice_amount) | count |
|---|---|---|
| 1 | 15 | 118 |

Showing all 1 rows.

Command took 4.82 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 16

## Monthly Placed and Canceled Orders

Cmd 17

```
#add the yyyymm column to to the new_retail_df
new_retail_df = retail_df.cache()
new_retail_df = new_retail_df.withColumn("yyyymm",date_format("InvoiceDate","yyyyMM").cast("integer"))
display(new_retail_df.limit(5))
```

▸ (1) Spark Jobs

▸ new_retail_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 7 more fields]

Table | Data Profile

| | Invoice | StockCode | Description | Quantity | InvoiceDate | Price | Customer ID | Country | yyyymm |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 489434 | 85048 | 15CM CHRISTMAS GLASS BALL 20 LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.95 | 13085 | United Kingdom | 200912 |
| 2 | 489434 | 79323P | PINK CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom | 200912 |
| 3 | 489434 | 79323W | WHITE CHERRY LIGHTS | 12 | 2009-12-01T07:45:00.000+0000 | 6.75 | 13085 | United Kingdom | 200912 |
| 4 | 489434 | 22041 | "RECORD FRAME 7"" SINGLE SIZE " | 48 | 2009-12-01T07:45:00.000+0000 | 2.1 | 13085 | United Kingdom | 200912 |
| 5 | 489434 | 21232 | STRAWBERRY CERAMIC TRINKET BOX | 24 | 2009-12-01T07:45:00.000+0000 | 1.25 | 13085 | United Kingdom | 200912 |

Showing all 5 rows.

Command took 4.59 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 18

```
#calculate all the orders included placed and cancelled
all_orders = new_retail_df.groupby("yyyymm").agg(expr('count(distinct Invoice)').alias('all_orders'))
all_orders = all_orders.orderBy("yyyymm")
display(all_orders.limit(5))
```

▸ (1) Spark Jobs

▸ all_orders: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, all_orders: long]

Table Data Profile

| | yyyymm | all_orders |
|---|---|---|
| 1 | 200912 | 2330 |
| 2 | 201001 | 1633 |
| 3 | 201002 | 1969 |
| 4 | 201003 | 2367 |
| 5 | 201004 | 1892 |

Showing all 5 rows.

Command took 8.05 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 19

```
#calculate the number of cancelled orders
cancelled_orders = new_retail_df.filter(col("Invoice").like("C%"))
cancelled_orders = cancelled_orders.groupby("yyyymm").agg(expr('count(distinct Invoice)').alias('cancelled_orders'))
cancelled_orders = cancelled_orders.orderBy("yyyymm")
display(cancelled_orders.limit(5))
#placed_orders = total_orders - 2 * cancelled_orders
```

▸ (1) Spark Jobs

▸ cancelled_orders: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, cancelled_orders: long]

Table Data Profile

| | yyyymm | cancelled_orders |
|---|---|---|
| 1 | 200912 | 401 |
| 2 | 201001 | 300 |
| 3 | 201002 | 240 |
| 4 | 201003 | 407 |
| 5 | 201004 | 304 |

Showing all 5 rows.

Command took 2.79 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:44 PM on spark

Cmd 20

```
#count the number of orders that actually went through each month
monthly_orders = all_orders.join(cancelled_orders, (cancelled_orders.yyyymm == all_orders.yyyymm))
monthly_orders = monthly_orders.withColumn("placed_orders",monthly_orders.all_orders - 2*monthly_orders.cancelled_orders)
display(monthly_orders.select(all_orders.yyyymm,"all_orders","cancelled_orders","placed_orders").orderBy(all_orders.yyyymm).limit(5))
```

▸ (3) Spark Jobs

▸ monthly_orders: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, all_orders: long ... 3 more fields]

Table Data Profile

| | yyyymm | all_orders | cancelled_orders | placed_orders |
|---|---|---|---|---|
| 1 | 200912 | 2330 | 401 | 1528 |
| 2 | 201001 | 1633 | 300 | 1033 |
| 3 | 201002 | 1969 | 240 | 1489 |
| 4 | 201003 | 2367 | 407 | 1553 |
| 5 | 201004 | 1892 | 304 | 1284 |

Showing all 5 rows.

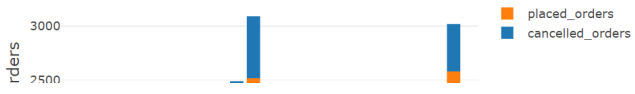
Command took 7.55 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:45 PM on spark

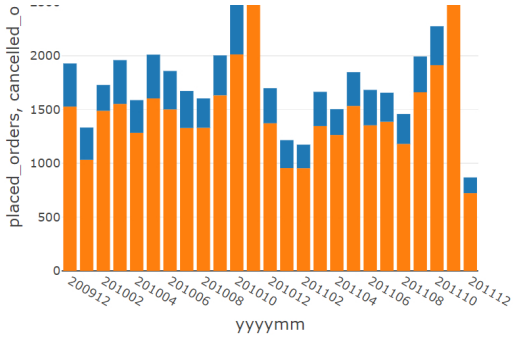
Cmd 21

```
#plot the # of placed orders versus canceled orders
display(monthly_orders.select(all_orders.yyyymm,"all_orders","cancelled_orders","placed_orders").orderBy(all_orders.yyyymm))
```

▸ (3) Spark Jobs

Chart Data Profile






Plot Options...

Command took 7.45 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:45 PM on spark

Cmd 22

## Monthly Sales

Cmd 23

```
#create the monthly sales dataframe
monthly_sales_df = new_retail_df.withColumn("Revenue", col("Quantity") * col("Price"))
revenue_df = monthly_sales_df.select("yyyymm", "Revenue")
revenue_df = revenue_df.groupby("yyyymm").sum()
revenue_df = revenue_df.orderBy("yyyymm")
revenue_df = revenue_df.drop("sum(yyyymm)")
display(revenue_df.limit(5))
```

(1) Spark Jobs

monthly_sales_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 8 more fields]

revenue_df: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, sum(Revenue): double]

Table | Data Profile

| | yyyymm | sum(Revenue) |
|---|---|---|
| 1 | 200912 | 799847.1100000143 |
| 2 | 201001 | 624032.8919999956 |
| 3 | 201002 | 533091.4260000042 |
| 4 | 201003 | 765848.7609999765 |
| 5 | 201004 | 590580.4319999823 |

Showing all 5 rows.

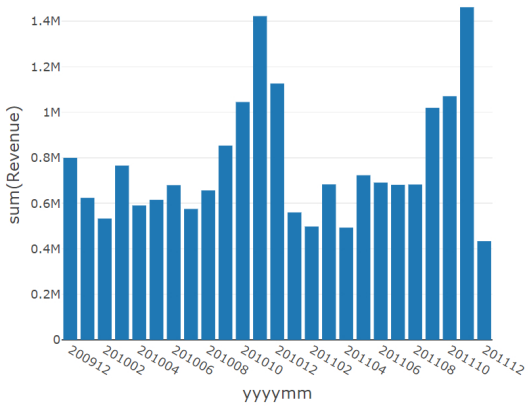
Command took 1.46 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:45 PM on spark

Cmd 24

```
#build the bar plot for monthly sales from 200912-201112.
display(revenue_df)
```

(1) Spark Jobs

Chart | Data Profile




Plot Options...

Command took 1.16 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:45 PM on spark

Cmd 25

## Monthly Active Users

Cmd 26

```
#compute # of active users (unique Customer ID) for each month
```

```
new_retail_df = new_retail_df.withColumnRenamed("Customer ID","CustomerID")
active_users_df = new_retail_df.select("yyyymm","CustomerID")
active_users_df = active_users_df.groupby("yyyymm").agg(expr('count(distinct CustomerID)').alias('ActiveCustomers'))
display(active_users_df.orderBy("yyyymm").limit(5))
```

▸ (1) Spark Jobs

▸ new_retail_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 7 more fields]

▸ active_users_df: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, ActiveCustomers: long]

Table | Data Profile

| | yyyymm | ActiveCustomers |
|---|---|---|
| 1 | 200912 | 1045 |
| 2 | 201001 | 786 |
| 3 | 201002 | 807 |
| 4 | 201003 | 1111 |
| 5 | 201004 | 998 |

Showing all 5 rows.

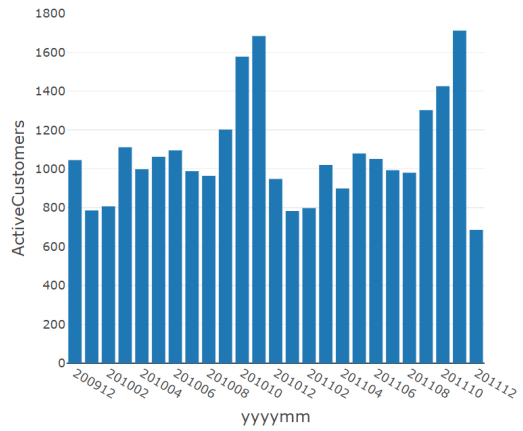
Command took 2.87 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:45 PM on spark

Cmd 27

```
#plot active users of each month
display(active_users_df.orderBy("yyyymm"))
```

▸ (1) Spark Jobs

Chart | Data Profile




Plot Options...

Command took 2.24 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:45 PM on spark

Cmd 28

## New and Existing Users

Cmd 29

```
#find out the first purchase for each user
first_purchase_for_user = new_retail_df.select("yyyymm","CustomerID")
first_purchase_for_user = first_purchase_for_user.groupBy("CustomerID").min()
first_purchase_for_user = first_purchase_for_user.withColumnRenamed("min(yyyymm)","user_starting_date")
first_purchase_for_user = first_purchase_for_user.withColumnRenamed("CustomerID","FirstCustomerID")
first_purchase_for_user = first_purchase_for_user.drop("min(CustomerID)")
display(first_purchase_for_user.orderBy("FirstCustomerID").limit(5))
```

▸ (1) Spark Jobs

▸ first_purchase_for_user: pyspark.sql.dataframe.DataFrame = [FirstCustomerID: double, user_starting_date: integer]

Table | Data Profile

| | FirstCustomerID | user_starting_date |
|---|---|---|
| 1 | null | 200912 |
| 2 | 12346 | 200912 |
| 3 | 12347 | 201010 |
| 4 | 12348 | 201009 |
| 5 | 12349 | 200912 |

Showing all 5 rows.

Command took 1.61 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:46 PM on spark

Cmd 30

```
#add the first users to the table of all the users
new_retail_df = new_retail_df.join(first_purchase_for_user, (new_retail_df.CustomerID == first_purchase_for_user.FirstCustomerID))
```

▸ new_retail_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 9 more fields]

Command took 0.02 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:46 PM on spark

Cmd 31

```
display(new_retail_df.orderBy("yyyymm").limit(5))
```

▸ (1) Spark Jobs

Table | Data Profile

| | Invoice | StockCode | Description | Quantity | InvoiceDate | Price | CustomerID | Country | yyyymm | FirstCustomerID |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 490285 | 21704 | BAG 250g SWIRLY MARBLES | 72 | 2009-12-04T12:59:00.000+0000 | 0.72 | 16656 | United Kingdom | 200912 | 16656 |
| 2 | 490285 | 21662 | VINTAGE GLASS COFFEE CADDY | 12 | 2009-12-04T12:59:00.000+0000 | 5.45 | 16656 | United Kingdom | 200912 | 16656 |
| 3 | 490285 | 84466 | TOP SECRET PEN SET | 72 | 2009-12-04T12:59:00.000+0000 | 0.95 | 16656 | United Kingdom | 200912 | 16656 |
| 4 | 490285 | 21661 | VINTAGE GLASS TEA CADDY | 12 | 2009-12-04T12:59:00.000+0000 | 5.45 | 16656 | United Kingdom | 200912 | 16656 |
| 5 | C491005 | 79340W | WHITE ORCHID FLOWER LIGHTS | -1 | 2009-12-08T17:40:00.000+0000 | 7.95 | 13112 | United Kingdom | 200912 | 13112 |

Showing all 5 rows.

Command took 3.08 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:46 PM on spark

Cmd 32

```
#merge new and existing dataframes
users_monthly_existing = new_retail_df.where(col("yyyymm") != col("user_starting_date"))
users_monthly_existing = users_monthly_existing.groupBy("yyyymm").agg(expr('count(distinct CustomerID) as ExistingUsers'))
users_monthly_existing = users_monthly_existing.withColumnRenamed("yyyymm","yyyymm_extra")

users_monthly_new = new_retail_df.where(col("yyyymm") == col("user_starting_date"))
users_monthly_new = users_monthly_new.groupBy("yyyymm").agg(expr('count(distinct CustomerID) as NewUsers'))

users_monthly = users_monthly_new.join(users_monthly_existing,users_monthly_new.yyyymm == users_monthly_existing.yyyymm_extra,"left")
users_monthly = users_monthly.drop("yyyymm_extra")
display(users_monthly.select("yyyymm","NewUsers","ExistingUsers").orderBy("yyyymm").limit(5))
```

▸ (1) Spark Jobs

▸ users_monthly_existing: pyspark.sql.dataframe.DataFrame = [yyyymm_extra: integer, ExistingUsers: long]

▸ users_monthly_new: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, NewUsers: long]

▸ users_monthly: pyspark.sql.dataframe.DataFrame = [yyyymm: integer, NewUsers: long ... 1 more field]

Table | Data Profile

| | yyyymm | NewUsers | ExistingUsers |
|---|---|---|---|
| 1 | 200912 | 1045 | null |
| 2 | 201001 | 394 | 392 |
| 3 | 201002 | 363 | 444 |
| 4 | 201003 | 436 | 675 |
| 5 | 201004 | 291 | 707 |

Showing all 5 rows.

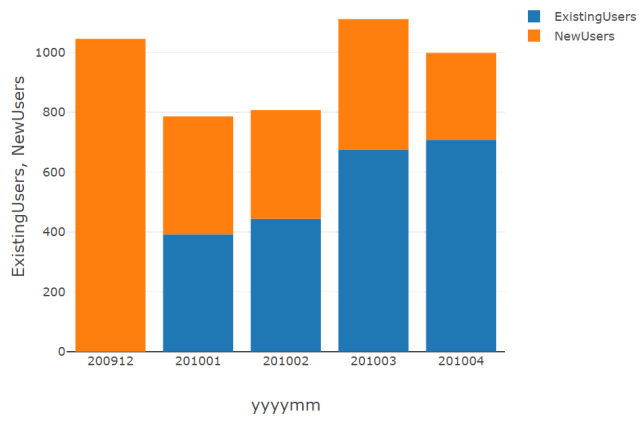
Command took 8.34 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:46 PM on spark

Cmd 33

```
#plot the # of new users versus existing users
display(users_monthly.select("yyyymm","NewUsers","ExistingUsers").orderBy("yyyymm").limit(5))
```

▸ (1) Spark Jobs

Chart | Data Profile




Plot Options...

Command took 7.47 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:46 PM on spark

Cmd 34

## RFM Analysis

RFM segmentation is a great method to divide customers into equal groups depending on three criteria:

- Recency - How recently did the customer purchase?
- Frequency - How often do they purchase?
- Monetary - How much do they spend?

Cmd 35

```
#data preparation for RFM
from pyspark.sql.types import *
new_retail_df = new_retail_df.withColumn("Quantity", new_retail_df["Quantity"].cast(IntegerType()))
new_retail_df = new_retail_df.withColumn("Price", new_retail_df["Price"].cast(DoubleType()))
new_retail_df = new_retail_df.withColumn("Date", to_date(unix_timestamp("InvoiceDate", "MM/dd/yyyy").cast("timestamp")))

#calculate difference in days
new_retail_df = new_retail_df.withColumn("Recency", expr("datediff('2011-12-09', Date)"))

#define Total column
new_retail_df = new_retail_df.withColumn("Total", round(new_retail_df["Price"] * new_retail_df["Quantity"], 2))
stat_summary_retail = new_retail_df.select("Quantity", "Price", "Total")
display(stat_summary_retail.summary())
```

▸ (1) Spark Jobs

▸ new_retail_df: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 12 more fields]

▸ stat_summary_retail: pyspark.sql.dataframe.DataFrame = [Quantity: integer, Price: double ... 1 more field]

Table | Data Profile

| | summary | Quantity | Price | Total |
|---|---|---|---|---|
| 1 | count | 824364 | 824364 | 824364 |
| 2 | mean | 12.414574144431343 | 3.676799578826833 | 20.195317080804063 |
| 3 | stddev | 188.9760990097584 | 70.24138768949332 | 308.68500104847647 |
| 4 | min | -80995 | 0.0 | -168469.6 |
| 5 | 25% | 2 | 1.25 | 4.25 |
| 6 | 50% | 5 | 1.95 | 11.25 |
| 7 | 75% | 12 | 3.75 | 19.5 |

Showing all 8 rows.

Command took 5.88 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:46 PM on spark

Cmd 36

```
#create the RFM table and determine Recency, Frequency and Monetary
rfm_table = new_retail_df.groupBy("CustomerId")\
                         .agg(min("Recency").alias("Recency"), \
                              count("Invoice").alias("Frequency"), \
                              round(sum("Total"), 2).alias("Monetary"))
display(rfm_table.limit(5))
```

▸ (1) Spark Jobs

▸ rfm_table: pyspark.sql.dataframe.DataFrame = [CustomerId: double, Recency: integer ... 2 more fields]

Table | Data Profile

| | CustomerId | Recency | Frequency | Monetary |
|---|---|---|---|---|
| 1 | 12467 | 386 | 18 | 0 |
| 2 | 12493 | 165 | 23 | 416.79 |
| 3 | 12671 | 606 | 45 | 2622.48 |
| 4 | 12737 | 498 | 2 | 3710.5 |
| 5 | 13094 | 21 | 38 | 2214.66 |

Showing all 5 rows.

Command took 1.17 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:47 PM on spark

Cmd 37

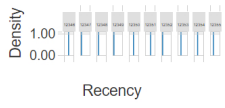
## Customer Distribution Based on Recency, Frequency and Monetary

Cmd 38

```
#Recency histogram
display(rfm_table.select("CustomerID", "Recency"))
```

▸ (6) Spark Jobs

Chart | Data Profile




Aggregated (by count) in the backend.
Truncated results, showing first 1000 rows.

Plot Options...

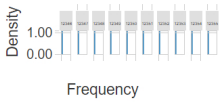
Command took 4.37 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:47 PM on spark

Cmd 39

```
#Frequency histogram
display(rfm_table.select("CustomerID", "Frequency"))
```

▸ (6) Spark Jobs

Chart | Data Profile




Aggregated (by count) in the backend.
Truncated results, showing first 1000 rows.

Plot Options...

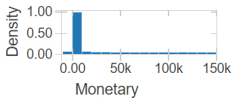
Command took 3.96 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:47 PM on spark

Cmd 40

```
#Monetary histogram
display(rfm_table.select("CustomerID", "Monetary"))
```

▸ (4) Spark Jobs

Chart | Data Profile




Showing sample based on the first 1000 rows.
Aggregate over all results.
Error plotting over all results: Show error.

Plot Options...

Command took 1.02 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:47 PM on spark

Cmd 41

## Computing Quantile of RFM values

We are assigning a score ranging from 1 to 5 to each customer, where 1 denotes the lowest score while 5 is the highest score using PySpark API called QuantileDiscretizer. Obviously, the customer who has 5s at all indicators is considered the best customer.

Cmd 42

```
from pyspark.ml.feature import QuantileDiscretizer
monetary_discretizer = QuantileDiscretizer().setNumBuckets(5).setInputCol("Monetary").setOutputCol("MonetaryScore").setRelativeError(0.0)
fitted_monetary_discretizer = monetary_discretizer.fit(rfm_table)
rfm_table = fitted_monetary_discretizer.transform(rfm_table)

frequency_discretizer = QuantileDiscretizer().setNumBuckets(5).setInputCol("Frequency").setOutputCol("FrequencyScore").setRelativeError(0.0)
fitted_frequency_discretizer = frequency_discretizer.fit(rfm_table)
rfm_table = fitted_frequency_discretizer.transform(rfm_table)

recency_discretizer = QuantileDiscretizer().setNumBuckets(5).setInputCol("Recency").setOutputCol("RecencyScore").setRelativeError(0.0)
fitted_recency_discretizer= recency_discretizer.fit(rfm_table)
rfm_table = fitted_recency_discretizer.transform(rfm_table)

display(rfm_table.limit(5))
```

▸ (4) Spark Jobs

▸ rfm_table: pyspark.sql.dataframe.DataFrame = [CustomerId: double, Recency: integer ... 5 more fields]

Table | Data Profile

| | CustomerId | Recency | Frequency | Monetary | MonetaryScore | FrequencyScore | RecencyScore |
|---|---|---|---|---|---|---|---|
| 1 | 12467 | 386 | 18 | 0 | 0 | 1 | 3 |
| 2 | 12493 | 165 | 23 | 416.79 | 1 | 1 | 2 |
| 3 | 12671 | 606 | 45 | 2622.48 | 3 | 2 | 4 |
| 4 | 12737 | 498 | 2 | 3710.5 | 4 | 0 | 4 |
| 5 | 13094 | 21 | 38 | 2214.66 | 3 | 2 | 1 |

Showing all 5 rows.

Command took 5.30 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:47 PM on spark

Cmd 43

## Need to Massage Quantile Discretizer Outputs to Produce Proper RFM Scores

Despite binning the customers by quantile, the labels given to each quantile is not the same value as the quantile's RFM score. Firstly, the QuantileDiscritzer outputs range from 0-4, when we

would like them to range from 1-5. Secondly, for recency, there is an inverse relationship between the quantile rank and its RFM score (i.e. larger quantile score = relatively higher recency value = lower recency score). This means we need to transform the RFM scores by adding new columns that are functions of the original ones generated by QuantileDiscritizer.

Cmd 44

```
#rename the fields produced by QuantileDiscritizer
rfm_table = (rfm_table.withColumnRenamed("MonetaryScore","OldMonetaryScor")
                      .withColumnRenamed("FrequencyScore","OldFrequencyScore")
                      .withColumnRenamed("RecencyScore","OldRecencyScore"))

#transform and drop old score values
rfm_table = rfm_table.withColumn("MonetaryScore",col("OldMonetaryScor")+1).drop("OldMonetaryScor")
rfm_table = rfm_table.withColumn("FrequencyScore",col("OldFrequencyScore")+1).drop("OldFrequencyScore")
udf_recency= udf(lambda x: -x + 5, FloatType())
rfm_table = rfm_table.withColumn("RecencyScore",udf_recency("OldRecencyScore")).drop("OldRecencyScore")
display(rfm_table.limit(5))
```

▸ (1) Spark Jobs

▸ rfm_table: pyspark.sql.dataframe.DataFrame = [CustomerId: double, Recency: integer ... 5 more fields]

Table Data Profile

| | CustomerId | Recency | Frequency | Monetary | MonetaryScore | FrequencyScore | RecencyScore |
|---|---|---|---|---|---|---|---|
| 1 | 12467 | 386 | 18 | 0 | 1 | 2 | 2 |
| 2 | 12493 | 165 | 23 | 416.79 | 2 | 2 | 3 |
| 3 | 12671 | 606 | 45 | 2622.48 | 4 | 3 | 1 |
| 4 | 12737 | 498 | 2 | 3710.5 | 5 | 1 | 1 |
| 5 | 13094 | 21 | 38 | 2214.66 | 4 | 3 | 4 |

Showing all 5 rows.

Command took 1.23 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:47 PM on spark

Cmd 45

## Concatenating RFM scores

Cmd 46

```
#concatenate the scores RFMScore = RecencyScore + FrequencyScore + MonetaryScore
rfm_table = (rfm_table.withColumn("RecencyScore",rfm_table.RecencyScore.cast(IntegerType()))
                      .withColumn("FrequencyScore",rfm_table.FrequencyScore.cast(IntegerType()))
                      .withColumn("MonetaryScore",rfm_table.MonetaryScore.cast(IntegerType())))

rfm_table = (rfm_table.withColumn("RFMScore",
                                  concat(rfm_table.RecencyScore.cast(StringType()),
                                         rfm_table.FrequencyScore.cast(StringType()),
                                         rfm_table.MonetaryScore.cast(StringType()))
                                 )
            )
display(rfm_table.limit(5))
```

▸ (1) Spark Jobs

▸ rfm_table: pyspark.sql.dataframe.DataFrame = [CustomerId: double, Recency: integer ... 6 more fields]

Table Data Profile

| | CustomerId | Recency | Frequency | Monetary | MonetaryScore | FrequencyScore | RecencyScore | RFMScore |
|---|---|---|---|---|---|---|---|---|
| 1 | 12467 | 386 | 18 | 0 | 1 | 2 | 2 | 221 |
| 2 | 12493 | 165 | 23 | 416.79 | 2 | 2 | 3 | 322 |
| 3 | 12671 | 606 | 45 | 2622.48 | 4 | 3 | 1 | 134 |
| 4 | 12737 | 498 | 2 | 3710.5 | 5 | 1 | 1 | 115 |
| 5 | 13094 | 21 | 38 | 2214.66 | 4 | 3 | 4 | 434 |

Showing all 5 rows.

Command took 0.68 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:48 PM on spark

Cmd 47

```
#find out who are the best customers
display(rfm_table.select("CustomerID").where("RFMScore == 555").limit(5))
```

▸ (2) Spark Jobs

Table Data Profile

| | CustomerID |
|---|---|
| 1 | 17884 |
| 2 | 15311 |
| 3 | 15750 |
| 4 | 15898 |
| 5 | 17659 |

Showing all 5 rows.

Command took 0.96 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:48 PM on spark

Cmd 48

```python
#segmenting of customers according to RecencyScore and FrequencyScore values
import re

def lookup(s):
    lookups  = [
        ('^[1-2][1-2]','Hibernating'),
        ('^[1-2][3-4]','At Risk'),
        ('^[1-2]5','Can\'t Lose'),
        ('^3[1-2]','About to Sleep'),
        ('^33','Need Attention'),
        ('^[3-4][4-5]','Loyal Customers'),
        ('^41','Promising'),
        ('^51','New Customers'),
        ('^[4-5][2-3]','Potential Loyalists'),
        ('^5[4-5]','Champions')
    ]
    for pattern, value in lookups:
        if re.search(pattern, s):
            return value
    return None

lookup_udf = udf(lookup, StringType())
```

Command took 0.04 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:48 PM on spark

Cmd 49

```python
#create the segment column in terms of lables
rfm_table = rfm_table.withColumn("Segment",lookup_udf(rfm_table.RFMScore))
display(rfm_table.limit(5))
```

▸ (1) Spark Jobs

▸ rfm_table: pyspark.sql.dataframe.DataFrame = [CustomerId: double, Recency: integer ... 7 more fields]

Table | Data Profile

| | CustomerId | Recency | Frequency | Monetary | MonetaryScore | FrequencyScore | RecencyScore | RFMScore | Segment |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 12467 | 386 | 18 | 0 | 1 | 2 | 2 | 221 | Hibernating |
| 2 | 12493 | 165 | 23 | 416.79 | 2 | 2 | 3 | 322 | About to Sleep |
| 3 | 12671 | 606 | 45 | 2622.48 | 4 | 3 | 1 | 134 | At Risk |
| 4 | 12737 | 498 | 2 | 3710.5 | 5 | 1 | 1 | 115 | Hibernating |
| 5 | 13094 | 21 | 38 | 2214.66 | 4 | 3 | 4 | 434 | Potential Loyalists |

Showing all 5 rows.

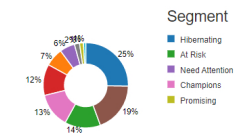
Command took 0.68 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:48 PM on spark

Cmd 50

```python
#Visulize the CustomerID versus Segment
display(rfm_table.select("CustomerID", "Segment"))
```

▸ (4) Spark Jobs

Chart | Data Profile




Showing sample based on the first 1000 rows.
Aggregate over all results.

Plot Options...

Command took 1.71 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:48 PM on spark

Cmd 51

```python
#get summary aggregations on RFM segments
final_rfm_table = (rfm_table.select("Segment", "Recency","Frequency","Monetary").groupby("Segment")
                                                                                .agg(count(rfm_table.Recency).alias('RecencyCount'),
                                                                                     avg(rfm_table.Recency).alias('RecencyMean'),
                                                                                     count(rfm_table.Frequency).alias('FrequencyCount'),
                                                                                     avg(rfm_table.Frequency).alias('FrequencyMean'),
                                                                                     count(rfm_table.Monetary).alias('MonetaryCount'),
                                                                                     avg(rfm_table.Monetary).alias('FrequenMonetaryMean')
                                                                                    )
                   )
display(final_rfm_table.limit(5))
```

▸ (4) Spark Jobs

▸ final_rfm_table: pyspark.sql.dataframe.DataFrame = [Segment: string, RecencyCount: long ... 5 more fields]

Table | Data Profile

| | Segment | RecencyCount | RecencyMean | FrequencyCount | FrequencyMean | MonetaryCount | FrequenMonetaryMean |
|---|---|---|---|---|---|---|---|
| 1 | Champions | 820 | 7.15 | 820 | 455.4207317073171 | 820 | 10659.325841463411 |
| 2 | Promising | 119 | 35.00840336134454 | 119 | 8.865546218487395 | 119 | 388.5754621848739 |
| 3 | At Risk | 812 | 392.2081280788177 | 812 | 74.35960591133005 | 812 | 1211.5383497536945 |
| 4 | About to Sleep | 427 | 108.95316159250585 | 427 | 18.416861826697893 | 427 | 542.1019672131147 |
| 5 | Hibernating | 1462 | 459.07113543091657 | 1462 | 15.61422708618331 | 1462 | 309.28761969904247 |

Showing all 5 rows.

Command took 5.35 seconds -- by homaalmasieh@gmail.com at 1/15/2022, 5:39:48 PM on spark

Shift+Enter to run