HW4_final

September 30, 2017

0.1 Problem 1 (30%)

Consider two attached text files: *bible.txt* and *4300.txt*. The first contains ASCII text of King James Bible and the other the text of James Joyce novel Ulysses.

Note: For this assignment a Docker container with the Jupyter Notebook as an IDE was used. The advantage of this approach is it's reproducabiltiy.

0.1.1 i) Download stop words

Download and parse a list of *stop words* from the web page: http://www.lextek.com/manuals/onix/stopwords1.html.

In [2]:
```
# Load libraries
import requests
import csv
from bs4 import BeautifulSoup

# Download page
page = requests.get("http://www.lextek.com/manuals/onix/stopwords1.html")

# Parse page
html = BeautifulSoup(page.content, 'html.parser').pre
text = html.get_text().split()

# Remove introduction
stopwords = text[21:len(text)]

## Export data to a datafile
result_file = open("stopwords.csv", 'w')
for i in stopwords:
  result_file.write(i + "\n")

result_file.close
```

Out[2]:
```
<function TextIOWrapper.close>
```

0.1.2 ii) RDD word number pairs

Use Spark transformation and action functions present in *RDD API* to transform those texts into RDD-s that contain words and numbers of occurrence of those words in respective text. From King James Bible eliminate all verse numbers of the form: *03:019:024*. Eliminate from both RDDs so called *ĂIJstop wordsĂİ*. List for us 30 most frequent words in each RDD (text).

Cleanup function

```
In [3]: # Cleanup function
        def clean_up(rdd_words):
          import re # Import regex library
          rdd_words_clean1 = re.sub(r'(03:019:024)', '', rdd_words) # certain verse
          rdd_words_clean2 = re.sub(r'([^A-Za-z0-9\s+])', '',
                                    rdd_words_clean1) # Nonwords
          rdd_words_split = rdd_words_clean2.split(' ') # Split data
          return [word.lower() for word in rdd_words_split if word != ''] # Lower case
```

Startup RDD Session

```
In [4]: # Import libraries
        import findspark
        findspark.init("/usr/local/spark")
        from pyspark import SparkContext, SparkConf

        # Start session
        conf = SparkConf().setMaster("local").setAppName("rdd")
        sc = SparkContext(conf = conf)
```

Load data into RDD and cleanup

```
In [5]: # Read data
        rdd_ulysses = sc.textFile("4300-2.txt")
        rdd_bible = sc.textFile("bible.txt")
        rdd_stopwords = sc.textFile("stopwords.csv")

        # Clean data and remove stopwords and verse number
        rdd_ulysess = rdd_ulysses.flatMap(clean_up)
        rdd_ulysess_cleaned = rdd_ulysess.subtract(rdd_stopwords)

        rdd_bible = rdd_bible.flatMap(clean_up)
        rdd_bible_cleaned = rdd_bible.subtract(rdd_stopwords)

        # Number of occurence (Mapreduce)
        rdd_ulysess_all = rdd_ulysess_cleaned.map(lambda x: (x, 1))\
            .reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], ascending=False)
        rdd_bible_all = rdd_bible_cleaned.map(lambda x: (x, 1))\
            .reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], ascending=False)
```

Print Top 30 Word pairs

```
In [6]: print("Bible - Top 30 word pairs:")
        print(rdd_bible_all.take(30))

        print("Ulysess - Top 30 word pairs:")
        print(rdd_ulysess_all.take(30))
```

```
Bible - Top 30 word pairs:
[('the', 64294), ('and', 51836), ('of', 34868), ('to', 13722), ('that', 12943), ('in', 12785), (
Ulysess - Top 30 word pairs:
[('the', 45043), ('of', 24576), ('and', 21758), ('a', 19569), ('to', 15095), ('in', 14831), ('he
```

0.1.3 iii) Get unique words

Create RDD-s that contain only words unique for each of text.

```
In [8]: # Get distinct values
        rdd_ulysess_dist = rdd_ulysess_cleaned.distinct()
        rdd_bible_dist = rdd_bible_cleaned.distinct()

        # Number of unique words
        rdd_ulysess_dist.count()
        rdd_bible_dist.count()
```

```
Out[8]: 44284
```

0.1.4 iv) Get common words

Finally create an RDD that contains only the words common to both texts. In latest RDD preserve numbers of occurrences in two texts. In other words a row in your RDD will look like (love 45 32). Print or store the words and the numbers of occurrences.

```
In [9]: rdd_combined = rdd_ulysess_all.join(rdd_bible_all)

        print("Common Words:")
        print(rdd_combined.sortByKey(False).take(10))
```

```
Common Words:
[('zion', (15, 152)), ('zealous', (6, 8)), ('zeal', (9, 16)), ('youths', (9, 2)), ('youthful', (
```

0.1.5 v) 20 most frequent words

Create for us the list of 20 most frequently used words common to both texts. In your report, print (store) the words, followed by the number of occurrences in Ulysses and then the Bible. Order your report in descending order starting by the number of occurrences in Ulysses. Present the same data this time ordered by the number of occurrences in the Bible.

```
In [10]: print("Top 20 word pairs (Ulysess):")
         rdd_combined = rdd_ulysess_all.join(rdd_bible_all)
         print(rdd_combined.sortBy(lambda a:a[1], False).take(5))

         print("Top 20 word pairs (Bible):")
         rdd_combined = rdd_bible_all.join(rdd_ulysess_all)
         print(rdd_combined.sortBy(lambda a:a[1], False).take(5))
```

```
Top 20 word pairs (Ulysess):
[('the', (45043, 64294)), ('of', (24576, 34868)), ('and', (21758, 51836)), ('a', (19569, 8291)),
Top 20 word pairs (Bible):
[('the', (64294, 45043)), ('and', (51836, 21758)), ('of', (34868, 24576)), ('to', (13722, 15095)
```

0.1.6 vi) Get a random sample

List for us a random samples containing 5% of words in the final RDD.

```
In [11]: rdd_5perc = format(rdd_combined.takeSample(False,\
                                                    int(rdd_combined.count() *
                                                        5/100), seed=123))
         print("5 percent sample of common words in both books")
         print(rdd_5perc)
```

```
5 percent sample of common words in both books
[('rinsed', (3, 3)), ('oak', (15, 15)), ('unjust', (17, 3)), ('withdrawn', (6, 6)), ('likewise',
```

0.2 Problem 2 (20%)

Implement problem 1 using DataFrame API.

0.2.1 i) DF word number pairs

Use Spark transformation and action functions present in *DF API* to transform those texts into DF-s that contain words and numbers of occurrence of those words in respective text. From King James Bible eliminate all verse numbers of the form: *03:019:024*. Eliminate from both RDDs so called *stop wordsĂİ*. List for us 30 most frequent words in each DF (text).

Functions

```
In [12]: # Function
         from pyspark.sql.functions import regexp_replace, trim, col, lower

         def removePunctuation(column):
           return trim(lower(regexp_replace(column,'([^A-Za-z0-9\s+])', ''))).alias('words')

         # Cleanup function
         def clean_up(rdd_words):
```

```
    import re # Import regex library
    rdd_words_clean1 = re.sub(r'(03:019:024)', '', rdd_words) # certain verse
    rdd_words_clean2 = re.sub(r'([^A-Za-z0-9\s+])', '', rdd_words_clean1) # Nonwords
    rdd_words_split = rdd_words_clean2.split(' ') # Split data
    return [word.lower() for word in rdd_words_split if word != ''] # Lower case
```

Create Session

```
In [13]: # Import libraries
         import findspark
         findspark.init("/usr/local/spark")
         from pyspark.sql import SparkSession
         from pyspark.sql.functions import split          # Function to split data
         from pyspark.sql.functions import explode        # Equivalent to flatMap

         # Create Session
         spark = SparkSession.builder.master("local") \
                             .appName("df").getOrCreate()
```

Data

```
In [24]: # Read data
         df_ulysses = spark.read.text("4300-2.txt")
         df_bible = spark.read.text("bible.txt")
         df_stopwords = spark.read.text("stopwords.csv")

         # Select words
         df_ulysses_all = df_ulysses.select(split(df_ulysses.value, " ").alias("words"))
         df_ulysses_all = df_ulysses_all.select(explode(df_ulysses_all.words).alias("words"))
         df_ulysses_all = df_ulysses_all.select(removePunctuation(col('words')))
         df_ulysses_all = df_ulysses_all.filter('words != Null or words != ""')

         df_bible_all = df_bible.select(split(df_bible.value, " ").alias("words"))
         df_bible_all = df_bible_all.select(explode(df_bible_all.words).alias("words"))
         df_bible_all = df_bible_all.select(removePunctuation(col('words')))
         df_bible_all = df_bible_all.filter('words != Null or words != ""')

         # Remove stopwords
         df_ulysses_cleaned = df_ulysses_all.join(df_stopwords, df_ulysses_all.words
                                                  == df_stopwords.value, \
                                                  'left_anti').select(df_ulysses_all.words)
         df_bible_cleaned = df_bible_all.join(df_stopwords, df_bible_all.words
                                              == df_stopwords.value, \
                                              'left_anti').select(df_bible_all.words)
```

```
In [25]: # Get frequent word pair
         df_ulysses_unique = df_ulysses_cleaned.groupBy("words").count()
         df_ulysses_unique = df_ulysses_unique.orderBy(["count"], ascending=False)
```

```
        print("30 Most frequent words: ")
        print(df_ulysses_unique.show(30))
```

```
30 Most frequent words:
+-----+-----+
|words|count|
+-----+-----+
|  the|45043|
|   of|24576|
|  and|21758|
|    a|19569|
|   to|15095|
|   in|14831|
|   he|12080|
|  his| 9983|
|    i| 8093|
| that| 7846|
| with| 7565|
|   it| 7131|
|  was| 6400|
|   on| 6361|
|  for| 5866|
|  you| 5842|
|  her| 5353|
|  him| 4570|
|   is| 4369|
|  all| 3988|
|   by| 3901|
|   at| 3890|
| said| 3617|
|   as| 3605|
|  she| 3402|
| from| 3290|
| they| 3074|
|   or| 3054|
|   me| 2824|
|bloom| 2798|
+-----+-----+
only showing top 30 rows

None
```

In [26]:
```
        # Get frequent word pair
        df_bible_unique = df_bible_cleaned.groupBy("words").count()
        df_bible_unique = df_bible_unique.orderBy(["count"], ascending=False)
        print("30 Most frequent words: ")
        print(df_bible_unique.show(30))
```

```
30 Most frequent words:
+-----+-----+
|words|count|
+-----+-----+
|  the|64294|
|  and|51836|
|   of|34868|
|   to|13722|
| that|12943|
|   in|12785|
|   he|10424|
|shall| 9842|
|  for| 9023|
| unto| 8997|
|    i| 8854|
|  his| 8473|
|    a| 8291|
| lord| 7830|
| they| 7382|
|   be| 7051|
|   is| 7041|
|  him| 6659|
|  not| 6638|
| them| 6430|
|   it| 6159|
| with| 6110|
|  all| 5656|
| thou| 5474|
|  thy| 4600|
|  was| 4524|
|  god| 4443|
|which| 4427|
|   my| 4368|
|   me| 4096|
+-----+-----+
only showing top 30 rows

None
```

0.2.2 ii) Get unique words

Create DF-s that contain only words unique for each of text.

```
In [27]: df_ulysess_dist = df_ulysses_all.distinct()
         df_ulysess_dist.count()
```

```
Out[27]: 30038
```

```
In [28]: df_bible_dist = df_bible_all.distinct()
         df_bible_dist.count()
```

```
Out[28]: 44285
```

0.2.3 iii) Get common words

Finally create an DF that contains only the words common to both texts. In latest DF preserve numbers of occurrences in two texts. In other words a row in your DF will look like (love 45 32). Print or store the words and the numbers of occurrences.

```
In [44]: df_combined = df_ulysses_unique.join(df_bible_unique, \
                                               df_ulysses_unique.words\
                                               == df_bible_unique.words, 'inner')
         df_combined = df_combined.toDF("words_ulysses", "count_ulysses", \
                                        "words_bible", "count_bible")
         df_combined.show(5)
```

```
+-------------+-------------+-----------+-----------+
|words_ulysses|count_ulysses|words_bible|count_bible|
+-------------+-------------+-----------+-----------+
|          the|        45043|        the|      64294|
|           of|        24576|         of|      34868|
|          and|        21758|        and|      51836|
|            a|        19569|          a|       8291|
|           to|        15095|         to|      13722|
+-------------+-------------+-----------+-----------+
only showing top 5 rows
```

0.2.4 iv) 20 most frequent words

Create for us the list of 20 most frequently used words common to both texts. In your report, print (store) the words, followed by the number of occurrences in Ulysses and then the Bible. Order your report in descending order starting by the number of occurrences in Ulysses. Present the same data this time ordered by the number of occurrences in the Bible.

```
In [47]: df_combined = df_combined.select(['words_ulysses', 'count_ulysses',
                                            'count_bible'])
```

```
In [48]: print(df_combined.count())
```

```
6097
```

```
In [58]: df_combined.orderBy(col('count_ulysses').desc()).show(20)
```

```
+-------------+-------------+-----------+
|words_ulysses|count_ulysses|count_bible|
+-------------+-------------+-----------+
|          the|        45043|      64294|
|           of|        24576|      34868|
|          and|        21758|      51836|
|            a|        19569|       8291|
|           to|        15095|      13722|
|           in|        14831|      12785|
|           he|        12080|      10424|
|          his|         9983|       8473|
|            i|         8093|       8854|
|         that|         7846|      12943|
|         with|         7565|       6110|
|           it|         7131|       6159|
|          was|         6400|       4524|
|           on|         6361|       2033|
|          for|         5866|       9023|
|          you|         5842|       2752|
|          her|         5353|       1994|
|          him|         4570|       6659|
|           is|         4369|       7041|
|          all|         3988|       5656|
+-------------+-------------+-----------+
only showing top 20 rows
```

```
In [59]: df_combined.orderBy(col('count_bible').desc()).show(20)
```

```
+-------------+-------------+-----------+
|words_ulysses|count_ulysses|count_bible|
+-------------+-------------+-----------+
|          the|        45043|      64294|
|          and|        21758|      51836|
|           of|        24576|      34868|
|           to|        15095|      13722|
|         that|         7846|      12943|
|           in|        14831|      12785|
|           he|        12080|      10424|
|        shall|          198|       9842|
|          for|         5866|       9023|
|         unto|           15|       8997|
|            i|         8093|       8854|
|          his|         9983|       8473|
|            a|        19569|       8291|
|         lord|          447|       7830|
|         they|         3074|       7382|
```

```
|           be|         2697|       7051|
|           is|         4369|       7041|
|          him|         4570|       6659|
|          not|         2726|       6638|
|         them|         2025|       6430|
+-------------+-------------+-----------+
only showing top 20 rows
```

0.2.5 v) Get a random sample

List for us a random samples containing 5% of words in the final DF.

```
In [62]: # List for us a random samples containing 5% of words in the final RDD.
         final_df_sample = df_combined.sample(False, 0.5, 123)
         print(final_df_sample.show())
```

```
+-------------+-------------+-----------+
|words_ulysses|count_ulysses|count_bible|
+-------------+-------------+-----------+
|           to|        15095|      13722|
|           in|        14831|      12785|
|           he|        12080|      10424|
|         that|         7846|      12943|
|           it|         7131|       6159|
|          was|         6400|       4524|
|           on|         6361|       2033|
|          for|         5866|       9023|
|           is|         4369|       7041|
|          all|         3988|       5656|
|           at|         3890|       1600|
|         said|         3617|       3999|
|         from|         3290|       3676|
|         they|         3074|       7382|
|          out|         2700|       2777|
|           be|         2697|       7051|
|           my|         2511|       4368|
|           up|         2495|       2386|
|        their|         2157|       3932|
|        there|         2117|       2303|
+-------------+-------------+-----------+
only showing top 20 rows

None
```

0.3 Problem 3 (30%)

Consider attached files *transactions.txt* and *products.txt*.

0.3.1 i) Load data

Each line in *transactions.txt* file contains a *transaction date*, *time*, *customer id*, *product id*, *quantity bought* and *price paid*, delimited with hash (#) sign. Each line in file *products.txt* contains *product id*, *product name*, *unit price* and *quantity available* in the store. Bring those data in Spark and organize it as DataFrames with named columns.

```
In [63]: # Read data
         df_transactions = spark.read.csv("transactions.txt", sep="#")
         df_products = spark.read.csv("products.txt", sep="#")

In [64]: df_transactions = df_transactions.withColumnRenamed('_c0', "transaction_date")
         df_transactions = df_transactions.withColumnRenamed('_c1', "time")
         df_transactions = df_transactions.withColumnRenamed('_c2', "customer_id")
         df_transactions = df_transactions.withColumnRenamed('_c3', "product_id")
         df_transactions = df_transactions.withColumnRenamed('_c4', "quantity_bought")
         df_transactions = df_transactions.withColumnRenamed('_c5', "price_paid")

         df_products = df_products.withColumnRenamed('_c0', "product_id")
         df_products = df_products.withColumnRenamed('_c1', "product_name")
         df_products = df_products.withColumnRenamed('_c2', "unit_price")
         df_products = df_products.withColumnRenamed('_c3', "quantity")
```

0.3.2 ii) Largest spending

Using either DataFrame methods or plain SQL statements find 5 customers with the largest spent on the day. Find the names of the products each of those 5 customers bought.

```
In [65]: #df_transactions.groupBy("customer id").sum().show()
         df_cust_spend = df_transactions.groupBy('customer_id', \
                                                 'transaction_date')
                                        .agg({'price_paid': 'sum'})
         df_cust_spend = df_cust_spend.orderBy('sum(price_paid)', \
                                               ascending=False)

In [66]: # Create tables
         df_cust_spend.createOrReplaceTempView("tbl_cust_spend")
         df_transactions.createOrReplaceTempView("tbl_transactions")
         df_products.createOrReplaceTempView("tbl_products")

In [67]: df_top5 = spark.sql("SELECT * FROM tbl_cust_spend LIMIT 5")
         df_top5.show()

+-----------+----------------+------------------+
|customer_id|transaction_date|   sum(price_paid)|
+-----------+----------------+------------------+
```

```
|         76|      2015-03-30|100049.00000000001|
|         53|      2015-03-30| 88829.76000000001|
|         56|      2015-03-30|          85906.94|
|         51|      2015-03-30|          83312.12|
|         31|      2015-03-30|          83202.61|
+-----------+----------------+------------------+
```

```
In [68]: df_transactions.show(5)
```

```
+----------------+--------+-----------+----------+---------------+----------+
|transaction_date|    time|customer_id|product_id|quantity_bought|price_paid|
+----------------+--------+-----------+----------+---------------+----------+
|      2015-03-30| 6:55 AM|         51|        68|              1|   9506.21|
|      2015-03-30| 7:39 PM|         99|        86|              5|   4107.59|
|      2015-03-30|11:57 AM|         79|        58|              7|   2987.22|
|      2015-03-30|12:46 AM|         51|        50|              6|   7501.89|
|      2015-03-30|11:39 AM|         86|        24|              5|    8370.2|
+----------------+--------+-----------+----------+---------------+----------+
only showing top 5 rows
```

```
In [69]: df_top5_products = df_transactions.join(df_top5, \
                                                 df_transactions.customer_id \
                                                 == df_top5.customer_id, "left")\
                                           .select(df_transactions.customer_id, \
                                                   df_transactions.product_id)
         df_top5_list = df_top5_products.join(df_products, df_top5_products.product_id \
                                              == df_products.product_id, "left")\
                                              .select(df_top5_products.customer_id,
                                                      df_products.product_name)
         df_top5_list.orderBy("customer_id").show()
```

```
+-----------+--------------------+
|customer_id|        product_name|
+-----------+--------------------+
|          1|SAMSUNG LED TV 42...|
|          1|ROBITUSSIN PEAK C...|
|          1|    LEGO Minifigures|
|          1|           Glipizide|
|          1|Scrub Care Povido...|
|          1|Medal Of Honor Al...|
|          1|Notebook Lenovo U...|
|          1|        LEGO Technic|
|          1|PC HP 490PD MT, D...|
|         10|              Ativan|
|         10|   LEGO Galaxy Squad|
```

```
|         10|SAMSUNG LED TV 32...|
|         10|          Dictionary|
|         10|ROBITUSSIN PEAK C...|
|         10|Procesor Intel Co...|
|         10|GAM X360 Hitman A...|
|        100|    chest congestion|
|        100|PC HP 490PD MT, D...|
|        100|     LEGO The Hobbit|
|        100|Roller Derby Roll...|
+-----------+--------------------+
only showing top 20 rows
```

0.3.3 iii) Total number sold

Find the names and total number sold of 10 most popular products. Order products once per the number sold and then by the total value (quanity*price) sold.

```
In [70]: # List the sum of sold products
         df_sum_products=df_transactions.groupBy('product_id')
                                        .agg({'quantity_bought': 'sum'})
         df_sum_products = df_sum_products
                                         .orderBy('sum(quantity_bought)', ascending=False)

         # Get top ten results
         df_sum_products.createOrReplaceTempView("tbl_sum_products")
         df_top10_products = spark.sql("SELECT * FROM tbl_sum_products LIMIT 10")

         # Calculate the total value
         df_products_distinct = df_products.select(df_products.product_id,
                                                   df_products.product_name,
                                                     df_products.unit_price).distinct()
         df_top10_products = df_top10_products.join(df_products_distinct,
                                                    df_top10_products.product_id
                                                    == df_products_distinct.product_id, "left")
         df_top10_products = df_top10_products.select(df_top10_products['product_name'],
                                                      df_top10_products['sum(quantity_bought)']
                                                      .alias("quantity"),
                                                      df_top10_products['unit_price'],
                                                      (df_top10_products['sum(quantity_bought)']
                                                       df_top10_products['unit_price'])
                                                      .alias("Total value"))
         df_top10_products.show()
```

```
+--------------------+--------+----------+------------------+
|        product_name|quantity|unit_price|       Total value|
+--------------------+--------+----------+------------------+
```

```
|Notebook Lenovo U...|   226.0|    461.08|         104204.08|
|SAMSUNG LED TV 39...|   142.0|   2531.15|          359423.3|
|               Jafra|   102.0|   3715.07|         378937.14|
|            Jantoven|   102.0|    3255.4|          332050.8|
|Far Cry 4 Limited...|   101.0|    711.88|          71899.88|
|Roller Derby Roll...|    91.0|   7783.79|         708324.89|
|Procesor Intel Co...|    90.0|   4570.99|          411389.1|
|  Sony Playstation 3|    88.0|   5088.35|447774.80000000005|
|    chest congestion|    84.0|   1305.04|         109623.36|
|Barbie Beach Ken ...|    82.0|    742.84|60912.880000000005|
+--------------------+--------+----------+------------------+
```

0.4 Problem 4 (20%)

Implement problem 3 using RDD APIs.

0.4.1 i) Load data

Each line in *transactions.txt* file contains a *transaction date, time, customer id, product id, quantity bought* and *price paid*, delimited with hash (#) sign. Each line in file *products.txt* contains *product id, product name, unit price* and *quantity available* in the store. Bring those data in Spark and organize it as DataFrames with named columns.

```
In [71]: from pyspark.sql import SQLContext, Row
         rdd_transactions = sc.textFile("transactions.txt")
         rdd_transactions = rdd_transactions.map(lambda x: x.split("#"))
         rdd_transactions = rdd_transactions.map(lambda x: Row(transaction_date = x[0],
                                                               time = x[1],
                                                               customer_id = int(x[2]),
                                                               product_id = int(x[3]),
                                                               quantity_bought = int(x[4]),
                                                               price_paid = float(x[5])))

         rdd_products = sc.textFile("products.txt")
         rdd_products = rdd_products.map(lambda x: x.split("#"))
         rdd_products = rdd_products.map(lambda x: Row(product_id = int(x[0]),
                                                       product_name = x[1],
                                                       unit_price = float(x[2]),
                                                       quantity = int(x[3])))
```

```
In [82]: rdd_transactions.take(5)
```

```
Out[82]: [Row(customer_id=51, price_paid=9506.21, product_id=68, quantity_bought=1, time='6:55 A
          Row(customer_id=99, price_paid=4107.59, product_id=86, quantity_bought=5, time='7:39 P
          Row(customer_id=79, price_paid=2987.22, product_id=58, quantity_bought=7, time='11:57
          Row(customer_id=51, price_paid=7501.89, product_id=50, quantity_bought=6, time='12:46
          Row(customer_id=86, price_paid=8370.2, product_id=24, quantity_bought=5, time='11:39 A
```

```
In [83]: rdd_products.take(5)
```

```
Out[83]: [Row(product_id=1, product_name='ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU', quantit
          Row(product_id=2, product_name='Mattel Little Mommy Doctor Doll', quantity=6, unit_pri
          Row(product_id=3, product_name='Cute baby doll, battery', quantity=2, unit_price=1808.
          Row(product_id=4, product_name='Bear doll', quantity=6, unit_price=51.06),
          Row(product_id=5, product_name='LEGO Legends of Chima', quantity=6, unit_price=849.36)
```

0.4.2 ii) Largest spending

Using either RDD methods or plain SQL statements find 5 customers with the largest spent on the day. Find the names of the products each of those 5 customers bought.

Create SQL schema

```
In [165]: # Import data types
          from pyspark.sql.types import *

          # The schema is encoded in a string.
          schemaString1 = "transaction_date time customer_id product_id quantity_bought price_pa
          schemaString2 = "product_id time product_name unit_price quantity"

          fields = [StructField(field_name, StringType(), True) for field_name in schemaString1.
          schema1 = StructType(fields)

          fields = [StructField(field_name, StringType(), True) for field_name in schemaString2.
          schema2 = StructType(fields)

          # Create schema
          sch_transactions = spark.createDataFrame(rdd_transactions, schema)
          sch_products = spark.createDataFrame(rdd_products, schema)

          # Creates a temporary view using the DataFrame
          sch_transactions.createOrReplaceTempView("tbl_transactions")
          sch_products.createOrReplaceTempView("tbl_products")
```

Note The SQL group by method somehow didn't work. That is why the DF method was used. In order to solve the problem the following SQL stament should work:

```
SELECT customer_id, SUM(to_float(quantity_bought) * to_float(price_paid)) AS revenue
FROM tbl_transactions
GROUP BY customer_id
ORDER BY revenue DESC
```

```
In [166]: sch_transactions = df_transactions.groupBy('customer_id', 'transaction_date').agg({'pr
          sch_transactions = sch_transactions.orderBy('sum(price_paid)', ascending=False)
```

```
In [167]: sch_transactions.createOrReplaceTempView("tbl_transactions2")
          tbl_cust_spend = spark.sql("SELECT * FROM tbl_transactions2 ORDER BY 'sum(price_paid)'
          tbl_cust_spend.rdd.take(5)
```

```
Out[167]: [Row(customer_id=76, transaction_date='2015-03-30', sum(price_paid)=100049.00000000001
           Row(customer_id=53, transaction_date='2015-03-30', sum(price_paid)=88829.76000000001),
           Row(customer_id=56, transaction_date='2015-03-30', sum(price_paid)=85906.94),
           Row(customer_id=51, transaction_date='2015-03-30', sum(price_paid)=83312.12),
           Row(customer_id=31, transaction_date='2015-03-30', sum(price_paid)=83202.61)]
```

Somehow the code below doesn't run. However, it would be the necessary SQL command.

```
In [168]: df_top5_products = spark.sql("SELECT * FROM tbl_transactions t \
                                        LEFT JOIN tbl_products p ON \
                                        t.product_id == p.product_id")
          df_top5_products.createOrReplaceTempView("tbl_top5_products")
          df_top5_list = spark.sql("SELECT t.customer_id FROM tbl_products p \
                                    LEFT JOIN tbl_top5_products t ON \
                                   t.product_id == p.product_id")
          df_top5_list.createOrReplaceTempView("df_top5_list")
          df_top5_list = spark.sql("SELECT * FROM tbl_cust_spend ORDER BY \
                                    'customer_id' DESC")
          df_top5_list.take(5)


          ---------------------------------------------------------------------------

          Py4JJavaError                             Traceback (most recent call last)

          /usr/local/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
           62         try:
          ---> 63             return f(*a, **kw)
           64         except py4j.protocol.Py4JJavaError as e:


          /usr/local/spark/python/lib/py4j-0.10.4-src.zip/py4j/protocol.py in get_return_value(ans
          318                     "An error occurred while calling {0}{1}{2}.\n".
          --> 319                     format(target_id, ".", name), value)
          320             else:


          Py4JJavaError: An error occurred while calling o526.sql.
          : org.apache.spark.sql.AnalysisException: Reference 't.product_id' is ambiguous, could be: p
                  at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.resolve(LogicalPlan.scala
                  at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.resolveChildren(LogicalPl
                  at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$$anonfun$apply$
                  at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$$anonfun$apply$
                  at org.apache.spark.sql.catalyst.analysis.package$.withPosition(package.scala:48)
                  at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$$anonfun$apply$
                  at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$$anonfun$apply$
                  at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformUp$1.apply(TreeNod
                  at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformUp$1.apply(TreeNod
```

```
at org.apache.spark.sql.catalyst.trees.CurrentOrigin$.withOrigin(TreeNode.scala:70)
at org.apache.spark.sql.catalyst.trees.TreeNode.transformUp(TreeNode.scala:288)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$3.apply(TreeNode.scala:286)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$3.apply(TreeNode.scala:286)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$4.apply(TreeNode.scala:306)
at org.apache.spark.sql.catalyst.trees.TreeNode.mapProductIterator(TreeNode.scala:18
at org.apache.spark.sql.catalyst.trees.TreeNode.mapChildren(TreeNode.scala:304)
at org.apache.spark.sql.catalyst.trees.TreeNode.transformUp(TreeNode.scala:286)
at org.apache.spark.sql.catalyst.plans.QueryPlan$$anonfun$transformExpressionsUp$1.a
at org.apache.spark.sql.catalyst.plans.QueryPlan$$anonfun$transformExpressionsUp$1.a
at org.apache.spark.sql.catalyst.plans.QueryPlan.transformExpression$1(QueryPlan.sca
at org.apache.spark.sql.catalyst.plans.QueryPlan.org$apache$spark$sql$catalyst$plans
at org.apache.spark.sql.catalyst.plans.QueryPlan.org$apache$spark$sql$catalyst$plans
at org.apache.spark.sql.catalyst.plans.QueryPlan$$anonfun$6.apply(QueryPlan.scala:29
at org.apache.spark.sql.catalyst.trees.TreeNode.mapProductIterator(TreeNode.scala:18
at org.apache.spark.sql.catalyst.plans.QueryPlan.mapExpressions(QueryPlan.scala:298)
at org.apache.spark.sql.catalyst.plans.QueryPlan.transformExpressionsUp(QueryPlan.sc
at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$$anonfun$apply$
at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$$anonfun$apply$
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan$$anonfun$resolveOperators
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan$$anonfun$resolveOperators
at org.apache.spark.sql.catalyst.trees.CurrentOrigin$.withOrigin(TreeNode.scala:70)
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.resolveOperators(LogicalP
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan$$anonfun$1.apply(LogicalP
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan$$anonfun$1.apply(LogicalP
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$4.apply(TreeNode.scala:306)
at org.apache.spark.sql.catalyst.trees.TreeNode.mapProductIterator(TreeNode.scala:18
at org.apache.spark.sql.catalyst.trees.TreeNode.mapChildren(TreeNode.scala:304)
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.resolveOperators(LogicalP
at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$.apply(Analyzer
at org.apache.spark.sql.catalyst.analysis.Analyzer$ResolveReferences$.apply(Analyzer
at org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1$$anonfun$appl
at org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1$$anonfun$appl
at scala.collection.LinearSeqOptimized$class.foldLeft(LinearSeqOptimized.scala:124)
at scala.collection.immutable.List.foldLeft(List.scala:84)
at org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1.apply(RuleExe
at org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1.apply(RuleExe
at scala.collection.immutable.List.foreach(List.scala:381)
at org.apache.spark.sql.catalyst.rules.RuleExecutor.execute(RuleExecutor.scala:74)
at org.apache.spark.sql.execution.QueryExecution.analyzed$lzycompute(QueryExecution.
at org.apache.spark.sql.execution.QueryExecution.analyzed(QueryExecution.scala:67)
at org.apache.spark.sql.execution.QueryExecution.assertAnalyzed(QueryExecution.scala
at org.apache.spark.sql.Dataset$.ofRows(Dataset.scala:66)
at org.apache.spark.sql.SparkSession.sql(SparkSession.scala:623)
at sun.reflect.GeneratedMethodAccessor99.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java
at java.lang.reflect.Method.invoke(Method.java:498)
at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
```

```
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:280)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:214)
        at java.lang.Thread.run(Thread.java:748)


During handling of the above exception, another exception occurred:


    AnalysisException                         Traceback (most recent call last)

    <ipython-input-168-616c74ab478a> in <module>()
      1 df_top5_products = spark.sql("SELECT * FROM tbl_transactions t LEFT JOIN tbl_product
      2 df_top5_products.createOrReplaceTempView("tbl_top5_products")
----> 3 df_top5_list = spark.sql("SELECT t.customer_id FROM tbl_products p LEFT JOIN tbl_top
      4 df_top5_list.createOrReplaceTempView("df_top5_list")
      5 df_top5_list = spark.sql("SELECT * FROM tbl_cust_spend ORDER BY 'customer_id' DESC")


    /usr/local/spark/python/pyspark/sql/session.py in sql(self, sqlQuery)
    554         [Row(f1=1, f2=u'row1'), Row(f1=2, f2=u'row2'), Row(f1=3, f2=u'row3')]
    555         """
--> 556         return DataFrame(self._jsparkSession.sql(sqlQuery), self._wrapped)
    557
    558     @since(2.0)


    /usr/local/spark/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py in __call__(self, *
   1131         answer = self.gateway_client.send_command(command)
   1132         return_value = get_return_value(
-> 1133             answer, self.gateway_client, self.target_id, self.name)
   1134
   1135         for temp_arg in temp_args:


    /usr/local/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
     67                                             e.java_exception.getStackTrace()))
     68             if s.startswith('org.apache.spark.sql.AnalysisException: '):
---> 69                 raise AnalysisException(s.split(': ', 1)[1], stackTrace)
     70             if s.startswith('org.apache.spark.sql.catalyst.analysis'):
     71                 raise AnalysisException(s.split(': ', 1)[1], stackTrace)


    AnalysisException: "Reference 't.product_id' is ambiguous, could be: product_id#1616, pr
```

0.4.3 iii) Total number sold

Find the names and total number sold of 10 most popular products. Order products once per the number sold and then by the total value (quanity*price) sold.

Note: It's the same problem as above.

```
In [169]: # List the sum of sold products
          df_sum_products=df_transactions.groupBy('product_id').agg({'quantity_bought': 'sum'})
          df_sum_products = df_sum_products.orderBy('sum(quantity_bought)',
                                                    ascending=False)

          # Get top ten results
          df_sum_products.createOrReplaceTempView("tbl_sum_products")
          df_top10_products = spark.sql("SELECT * FROM tbl_sum_products LIMIT 10")

          # Calculate the total value
          df_products_distinct = df_products.select(df_products.product_id,
                                                    df_products.product_name,
                                                    df_products.unit_price).distinct()
          df_top10_products = df_top10_products.join(df_products_distinct,
                                                     df_top10_products.product_id
                                                     == df_products_distinct.product_id, "left")
          df_top10_products = df_top10_products.select(df_top10_products['product_name'],
                                                       df_top10_products['sum(quantity_bought)']
                                                       .alias("quantity"),
                                                       df_top10_products['unit_price'],
                                                       (df_top10_products['sum(quantity_bought)'
                                                        df_top10_products['unit_price'])
                                                       .alias("Total value"))
          df_top10_products.show()
```

```
+--------------------+--------+----------+------------------+
|        product_name|quantity|unit_price|       Total value|
+--------------------+--------+----------+------------------+
|Notebook Lenovo U...|     226|    461.08|         104204.08|
|SAMSUNG LED TV 39...|     142|   2531.15|          359423.3|
|            Jantoven|     102|    3255.4|          332050.8|
|               Jafra|     102|   3715.07|         378937.14|
|Far Cry 4 Limited...|     101|    711.88|          71899.88|
|Roller Derby Roll...|      91|   7783.79|         708324.89|
|Procesor Intel Co...|      90|   4570.99|          411389.1|
|  Sony Playstation 3|      88|   5088.35|447774.80000000005|
|    chest congestion|      84|   1305.04|         109623.36|
|Barbie Beach Ken ...|      82|    742.84|60912.880000000005|
+--------------------+--------+----------+------------------+
```