**HU Extension Assignment 04 E63 Big Data Analytics**

Issued on: Sept 24, 2017 Due on Saturday by 4 PM EST, Sept 30, 2017

You can do these problems in the language of your choice: Python, Scala, Java or R.

**Problem 1.** 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’s novel Ulysses. 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 “stop words”. Please use the list of stop words on Web page: <http://www.lextek.com/manuals/onix/stopwords1.html>. . Create RDD-s that contain only words unique for each of text. 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). List for us 30 most frequent words in each RDD (text). Print or store the words and the numbers of occurrences. 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. List for us a random samples containing 5% of words in the final RDD. We are just practicing RDD transformations and actions. You could implement this problem in a command shell or as a standalone program.

**(30%)**

//create rdd,removed pattern of form 03:019:024,remove punctuations, removed empty ones,

val inputBible = sc.textFile("/Users/smukherjee5/harvard\_hw/bible.txt")

val regexpr = """\b\d{2}[-:]?\d{3}[-:]?\d{3}\b""".r

val filteredBible = inputBible.flatMap(\_.split(" ")).map(\_.trim).filter(x => !regexpr.pattern.matcher(x).matches).map(\_.replaceAll("[,.!?;\*^~]","")).filter(\_.nonEmpty)

//now create stop list. created a text file with stop words from the url, and used to create rdd of stop words

val stop = sc.textFile("/Users/smukherjee5/harvard\_hw/stopwords.txt")

//next create a set of the stopwords

val stopSet = stop.collect.toSet

//And removed stop words

val filteredBibleStopWords = filteredBible.map(\_.toLowerCase).filter(!stopSet.contains(\_))

// Now do similarly for 4300.txt

val input4300 = sc.textFile("/Users/smukherjee5/harvard\_hw/4300.txt")

//Sanitize data

val filtered4300 = input4300.flatMap(\_.split(" ")).map(\_.trim).map(\_.replaceAll("[,.!?:;\*^~]","")).filter(\_.nonEmpty)

//Remove stopwords

val filtered4300StopWords = filtered4300.map(\_.toLowerCase).filter(!stopSet.contains(\_))

//Now put their respective counts by first mapping each word to 1, and then do a reduceByKey to add these, this also ensures we have distinct

val filtered4300StopWordsCount= filtered4300StopWords.map((\_,1)).reduceByKey(\_+\_)

val filteredBibleStopWordsCount= filteredBibleStopWords.map((\_,1)).reduceByKey(\_+\_)

//Lastly join the two rdds to get the word and unique count per textfile

filtered4300StopWordsCount.join(filteredBibleStopWordsCount).collect()

res90: Array[(String, (Int, Int))] = Array((bone,(33,19)), (hem,(6,7)), (fuller,(6,1)), (crying,(30,31)), (breath,(171,42)), (fowl,(12,31)), (accomplished,(9,26)), (harlot's,(6,2)), (inquisition,(6,3)), (stern,(13,1)), (espoused,(3,5)), (erected,(9,1)), (adultery,(15,40)), (descending,(12,8)), (seventy,(9,61)), (cor,(3,1)), (goodliest,(6,2)), (converted,(15,9)), (astray,(11,22)), (viewed,(12,6)), (rolled,(78,12)), (didst,(12,122)), (whit,(9,5)), (win,(39,2)), (worshippers,(3,7)), (fail,(12,64)), (table,(210,75)), (bedad,(6,2)), (inherited,(3,6)), (instruments,(12,51)), (persecute,(3,25)), (handful,(15,9)), (astonished,(3,34)), (miserable,(3,3)), (brow,(60,2)), (accept,(33,27)), (abba,(3,3)), (4,(67,6)), (david's,(6,53)), (apply,(14,6)), (hour,(162,94)), (vow,(12,41)), (deceased,(33,2)),...

//Sorted with top 30 most frequent, false is used for sorting by desc

filtered4300StopWordsCount.sortBy(\_.\_2,false).take(30)

res110: Array[(String, Int)] = Array((bloom,2654), (stephen,1442), (time,1131), (eyes,963), (little,870), (street,870), (hand,855), (yes,855), (father,828), (day,741), (round,717), (night,687), (sir,657), (name,645), ((he,636), (head,627), (god,612), (life,583), (john,576), (look,570), (woman,558), (poor,546), (don't,537), (voice,513), (house,511), (tell,508), (course,495), (left,492), (hat,486), (white,486))

filteredBibleStopWordsCount.sortBy(\_.\_2,false).take(30)

res111: Array[(String, Int)] = Array((unto,8997), (lord,7828), (thou,5473), (thy,4600), (god,4437), (ye,3980), (thee,3825), (israel,2563), (son,2370), (king,2270), (hath,2264), (people,2144), (house,2022), (children,1801), (day,1731), (land,1718), (shalt,1616), (hand,1466), (saying,1445), (behold,1325), (saith,1261), (hast,1070), (sons,1067), (david,1014), (earth,986), (father,978), (jesus,972), (thine,938), (name,930), (thereof,905))

// Next we find the most frequent common words between the two files. I use sortBy for this and sort on \_2 (which is the count tuple)

val commonFiltered = filtered4300StopWordsCount.join(filteredBibleStopWordsCount)

commonFiltered.sortBy(\_.\_2,false).take(20)

res97: Array[(String, (Int, Int))] = Array((stephen,(1442,7)), (time,(1131,609)), (eyes,(963,476)), (little,(870,239)), (street,(870,35)), (hand,(855,1391)), (yes,(855,4)), (father,(828,928)), (day,(741,1654)), (round,(717,320)), (night,(687,282)), (sir,(657,11)), (name,(645,886)), ((he,(636,7)), (head,(627,346)), (god,(612,4217)), (life,(583,413)), (john,(576,139)), (look,(570,152)), (woman,(558,352)))

// In your report, print (store) the words, followed by the number of occurrences in Ulysses and then the Bible

//I do similar steps as above for Ulysses and then join to the Bible text file

val filteredUlysses =inputUlysses.flatMap(\_.split(" ")).map(\_.trim).map(\_.replaceAll("[,.!?:;\*^~]","")).filter(!stopSet.contains(\_)).filter(\_.nonEmpty)

val filteredUlyssesCount = filteredUlysses.map((\_,1)).reduceByKey(\_+\_)

filteredUlyssesCount.join(filteredBibleStopWordsCount).collect

res125: Array[(String, (Int, Int))] = Array((bone,(9,19)), (fuller,(2,1)), (crying,(9,31)), (breath,(51,42)), (fowl,(3,31)), (harlot's,(1,2)), (inquisition,(2,3)), (adultery,(5,40)), (goodliest,(2,2)), (viewed,(4,6)), (rolled,(24,12)), (win,(13,2)), (worshippers,(1,7)), (bedad,(2,2)), (instruments,(3,51)), (handful,(4,9)), (astonished,(1,34)), (miserable,(1,3)), (brow,(17,2)), (apply,(4,6)), (hour,(54,94)), (vow,(3,41)), (trial,(4,6)), (blown,(7,4)), (office,(48,50)), (tossed,(10,7)), (pour,(9,63)), (enemy,(6,106)), (floor,(27,18)), (throng,(2,2)), (birds,(14,24)), (bark,(5,1)), (robber,(6,5)), (happy,(27,28)), (worse,(30,26)), (feebleminded,(1,1)), (spoils,(2,5)), (surely,(9,284)), (men's,(11,24)), (glass,(35,9)), (pulled,(17,7)), (hospitality,(7,4)), (beside,(39,132)), (town,(25,13)),...

// Order your report in descending order starting by the number of occurrences in Ulysses. I use \_.\_2.\_1 to get the count for ulysses (time,("344-->\_1",623)

val joinUlyssesBible = filteredUlyssesCount.join(filteredBibleStopWordsCount)

joinUlyssesBible.sortBy((\_.\_2.\_1),false).collect

res127: Array[(String, (Int, Int))] = Array((time,(344,623)), (eyes,(277,501)), (street,(268,36)), (little,(261,242)), (day,(230,1731)), (hand,(224,1466)), (night,(207,307)), (name,(207,930)), (round,(199,320)), (sir,(198,12)), (life,(174,451)), (woman,(162,357)), (yes,(160,4)), (house,(157,2022)), (head,(154,364)), (course,(153,37)), (look,(151,155)), (am,(147,874)), (own,(147,596)), (world,(142,287)), (told,(142,282)), (poor,(137,205)), (tell,(136,217)), (voice,(135,505)), (father,(134,978)), (bit,(134,3)), (left,(130,348)), (fellow,(129,27)), (door,(127,190)), (till,(127,169)), (love,(123,310)), (white,(123,75)), (wife,(121,396)), (hear,(117,551)), (word,(112,699)), (home,(111,183)), (words,(110,544)), (water,(110,395)), (read,(109,76)), (heard,(108,641)), (morning,(108,227)), (comin...

// Present the same data this time ordered by the number of occurrences in the Bible. I am assuming this too is desc ??

joinUlyssesBible.sortBy((\_.\_2.\_2),false).collect

res128: Array[(String, (Int, Int))] = Array((unto,(5,8997)), (lord,(69,7828)), (thou,(39,5473)), (thy,(32,4600)), (god,(15,4437)), (ye,(12,3980)), (thee,(25,3825)), (son,(92,2370)), (king,(37,2270)), (hath,(12,2264)), (people,(62,2144)), (house,(157,2022)), (children,(44,1801)), (day,(230,1731)), (land,(76,1718)), (shalt,(1,1616)), (hand,(224,1466)), (saying,(56,1445)), (behold,(5,1325)), (saith,(9,1261)), (hast,(9,1070)), (sons,(18,1067)), (earth,(56,986)), (father,(134,978)), (name,(207,930)), (thereof,(3,905)), (forth,(45,904)), (neither,(20,879)), (am,(147,874)), (city,(30,870)), (brought,(79,863)), (days,(75,862)), (heart,(103,830)), (pass,(38,830)), (according,(15,792)), (whom,(44,763)), (nor,(53,755)), (bring,(41,725)), (offering,(6,724)), (set,(57,713)), (word,(112,699)), (sent,...

//Sample 5% withReplacement as false, I use sample on rdd to do this

//Ref:https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html

joinUlyssesBible.sample(false,.05).collect

res138: Array[(String, (Int, Int))] = Array((tossed,(10,7)), (bark,(5,1)), (surely,(9,284)), (truly,(9,42)), (bleating,(2,1)), (melt,(4,17)), (talk,(32,24)), (child's,(5,4)), (heavens,(6,133)), (suddenly,(34,41)), (plague,(5,98)), (stooped,(6,7)), (frozen,(6,1)), (blast,(5,8)), (shoes,(17,21)), (wonders,(3,55)), (thicker,(2,2)), (talking,(57,9)), (loathsome,(1,4)), (proclaimed,(1,16)), (itching,(3,1)), (requests,(1,1)), (hate,(28,87)), (rumour,(5,10)), (fairest,(1,3)), (likeness,(6,34)), (abstain,(2,6)), (gardens,(9,12)), (honestly,(2,3)), (shipwreck,(1,2)), (disposing,(1,1)), (bottle,(24,15)), (begotten,(2,24)), (lords,(7,42)), (science,(9,2)), (colt,(3,15)), (sevens,(1,2)), (hated,(3,60)), (drag,(2,2)), (baptism,(1,22)), (amounting,(1,1)), (steady,(2,1)), (wares,(1,5)), (times,(72,145...

filtered4300StopWordsCount.join(filteredBibleStopWordsCount).sample(false,.05).collect

res137: Array[(String, (Int, Int))] = Array((accomplished,(9,26)), (inherited,(3,6)), (deceased,(33,2)), (benjamin,(15,162)), (drive,(27,57)), (rear,(6,4)), (concerning,(45,245)), (stripes,(3,17)), (governor's,(6,1)), ("defects",(3,2)), (flood,(39,43)), (hearing,(45,39)), (flea,(6,2)), (pain,(57,25)), (seasoned,(3,2)), (ink,(48,4)), (worms,(6,8)), (ages,(48,4)), (shadowing,(6,3)), (deliverance,(3,16)), (ephod,(3,52)), (buckler,(3,11)), (valiantly,(3,6)), (prolonged,(21,9)), (heard,(381,641)), (drawn,(69,28)), (vale,(9,9)), (oration,(3,1)), (blow,(51,39)), (reasonable,(6,3)), (supplied,(15,2)), (measured,(18,46)), (mending,(3,2)), (ascending,(12,5)), (yearned,(9,1)), (speech,(93,49)), (thumb,(45,6)), (midnight,(33,14)), (latter,(42,42)), (procure,(6,2)), (provoke,(3,42)), (porch,(27,39))...

**Problem 2**. Implement problem 1 using DataFrame API. You could implement this problem in a command shell or as a standalone program.

**(20%)**

//create DF,remove punctuations, removed empty ones,

val inputBibleDF = spark.read.text("/Users/smukherjee5/harvard\_hw/bible.txt").as[String].flatMap(\_.split(" ")).filter(\_.nonEmpty).map(\_.replaceAll("[,.!?;\*^~-]","")).toDF("words")

val stopRegexDF =inputBibleDF.filter(!inputBibleDF("words").rlike("""\b\d{2}[-:]?\d{3}[-:]?\d{3}\b""")).toDF("words")

//create stop words dataframe

val stopDF = spark.read.text("/Users/smukherjee5/harvard\_hw/stopwords.txt").toDF("words")

//Next I do a left outer join to filter out the stop words

val allFiltered = stopRegexDF.join(stopDF,lower(stopRegexDF("words"))=== stopDF("words"),"left\_outer").where(stopDF("words").isNull).select(lower(stopRegexDF("words")))

// Now do similarly for 4300.txt

//Sanitize data

val input4300DF = spark.read.text("/Users/smukherjee5/harvard\_hw/4300.txt").as[String].flatMap(\_.split(" ")).filter(\_.nonEmpty).map(\_.replaceAll("[,.!?;\*^~-]","")).toDF("words")

//Remove stopwords

val allFiltered4300 = input4300DF.join(stopDF,lower(input4300DF("words")) === stopDF("words"),"left\_outer").where(stopDF("words").isNull).select(lower(input4300DF("words")))

//Now put their respective counts using groupby

val allFiltered4300Count = allFiltered4300.groupBy("lower(words)").count.select("lower(words)","count")

val allFilteredBibleCount = allFiltered.groupBy("lower(words)").count.select("lower(words)","count")

//Lastly join the two DFs to get the word and unique count per textfile

allFilteredBibleCount.join(allFiltered4300Count, allFilteredBibleCount("lower(words)")===allFiltered4300Count("lower(words)")).select(allFilteredBibleCount("lower(words)"),allFiltered4300Count("count"),allFilteredBibleCount("count")).show(10)

+------------+-----+-----+

|lower(words)|count|count|

+------------+-----+-----+

| art| 144| 492|

| blossom| 3| 5|

| brands| 9| 1|

| cures| 9| 1|

| doubts| 6| 1|

| embrace| 24| 8|

| hope| 192| 120|

| inner| 54| 37|

| jericho)| 3| 1|

| man)| 6| 1|

+------------+-----+-----+

only showing top 10 rows

//Sorted with top 30 most frequent, using orderBy desc and doing a take(30). I had to filter empty strings I found

scala> allFilteredBibleCount.orderBy(desc("count")).show(30)

+------------+-----+

|lower(words)|count|

+------------+-----+

| unto| 8997|

| lord| 7512|

| thou| 5470|

| thy| 4600|

| god| 4217|

| ye| 3973|

| thee| 3575|

| israel| 2399|

| son| 2334|

| hath| 2262|

| king| 2233|

| people| 2054|

| house| 1952|

| children| 1778|

| day| 1654|

| land| 1647|

| shalt| 1616|

| saying| 1443|

| hand| 1391|

| behold| 1323|

| saith| 1260|

| hast| 1069|

| sons| 1051|

| david| 988|

| jesus| 955|

| thine| 932|

| father| 928|

| forth| 891|

| name| 886|

| earth| 882|

+------------+-----+

only showing top 30 rows

allFiltered4300Count.where(!(allFiltered4300Count("lower(words)")==="")).orderBy(desc("count")).show(30)

+------------+-----+

|lower(words)|count|

+------------+-----+

| bloom| 1814|

| stephen| 1154|

| time| 1131|

| yes| 1050|

| eyes| 957|

| little| 870|

| bloom:| 852|

| hand| 849|

| street| 837|

| father| 828|

| day| 735|

| round| 717|

| night| 675|

| sir| 654|

| (he| 636|

| god| 636|

| name| 630|

| head| 618|

| look| 582|

| life| 580|

| john| 579|

| don't| 561|

| woman| 558|

| poor| 555|

| tell| 529|

| house| 511|

| left| 492|

| course| 492|

| dedalus| 489|

| white| 486|

+------------+-----+

only showing top 30 rows

// Next we find the most 20 frequent common words between the two files. I join the two datasets and then orderby and take 20

allFiltered4300Count.where(!(allFiltered4300Count("lower(words)")==="")).join(allFilteredBibleCount,allFiltered4300Count("lower(words)")===allFilteredBibleCount("lower(words)")).select(allFiltered4300Count("lower(words)"),allFiltered4300Count("count").alias("countBible"),allFilteredBibleCount("count").alias("count4300")).orderBy(desc("countBible"),desc("count4300")).show(20)

+------------+----------+---------+

|lower(words)|countBible|count4300|

+------------+----------+---------+

| stephen| 1154| 7|

| time| 1131| 609|

| yes| 1050| 4|

| eyes| 957| 476|

| little| 870| 239|

| hand| 849| 1391|

| street| 837| 35|

| father| 828| 928|

| day| 735| 1654|

| round| 717| 320|

| night| 675| 282|

| sir| 654| 11|

| god| 636| 4217|

| (he| 636| 7|

| name| 630| 886|

| head| 618| 346|

| look| 582| 152|

| life| 580| 413|

| john| 579| 139|

| woman| 558| 352|

+------------+----------+---------+

only showing top 20 rows

///----> check if unique needs to be done

//Sample 5% withReplacement as false, I use sample on rdd to do this

//Ref:https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html

allFiltered4300Count.where(!(allFiltered4300Count("lower(words)")==="")).sample(false,.05).show

+------------+-----+

|lower(words)|count|

+------------+-----+

| formalities| 3|

| baaaa| 3|

| tilting| 3|

| harshed| 3|

| painted| 27|

| hardy| 3|

| combed| 3|

| inwit's| 3|

| spurts| 6|

| rears| 3|

|monosyllabic| 6|

| agrarian| 3|

| birdsweet| 6|

| armpit| 15|

| retainers| 3|

| johann| 3|

| map| 9|

| flaming| 15|

| settlement| 3|

| terra| 3|

+------------+-----+

only showing top 20 rows

filtered4300StopWordsCount.join(filteredBibleStopWordsCount).sample(false,.05).collect

allFilteredBibleCount.where(!(allFilteredBibleCount("lower(words)")==="")).sample(false,.05).show

+------------+-----+

|lower(words)|count|

+------------+-----+

| pitcher| 12|

| shilonites| 1|

| eliphaz| 15|

| take:| 2|

|conversation| 20|

| launch| 1|

| watch| 59|

| dung| 27|

| decay| 1|

| shophan| 1|

| crownest| 1|

| glitter:| 1|

| waxed| 37|

| showers| 8|

| coral| 2|

| farm| 1|

| remaliah| 11|

| debate| 4|

| intend| 4|

| succour| 3|

+------------+-----+

only showing top 20 rows

**Problem 3**. Consider attached files transactions.txt and products.txt. 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. 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. 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.

**(30%)**

**Loading the files into spark with delimiter hash and with named columns**

**val products = spark.read.format("csv").option("header", "false").option("delimiter","#").load("/Users/smukherjee5/harvard\_hw/products.txt").toDF("product\_id","product\_name","unit\_price","quanitity\_available")**

**products.show**

**+----------+--------------------+----------+-------------------+**

**|product\_id| product\_name|unit\_price|quanitity\_available|**

**+----------+--------------------+----------+-------------------+**

**| 1|ROBITUSSIN PEAK C...| 9721.89| 10|**

**| 2|Mattel Little Mom...| 6060.78| 6|**

**| 3|Cute baby doll, b...| 1808.79| 2|**

**| 4| Bear doll| 51.06| 6|**

**| 5|LEGO Legends of C...| 849.36| 6|**

**| 6| LEGO Castle| 4777.51| 10|**

**| 7| LEGO Mixels| 8720.91| 1|**

**| 8| LEGO Star Wars| 7592.44| 4|**

**| 9|LEGO Lord of the ...| 851.67| 2|**

**| 10| LEGO The Hobbit| 7314.55| 9|**

**| 11| LEGO Minecraft| 5646.81| 3|**

**| 12| LEGO Hero Factory| 6911.2| 1|**

**| 13| LEGO Architecture| 604.58| 5|**

**| 14| LEGO Technic| 7423.48| 3|**

**| 15|LEGO Storage & Ac...| 3125.96| 2|**

**| 16| LEGO Classic| 9933.3| 10|**

**| 17| LEGO Galaxy Squad| 5593.16| 4|**

**| 18| LEGO Mindstorms| 6022.88| 10|**

**| 19| LEGO Minifigures| 5775.99| 1|**

**| 20| LEGO Elves| 4589.79| 4|**

**+----------+--------------------+----------+-------------------+**

**val transaction = spark.read.format("csv").option("header","false").option("delimiter","#").load("/Users/smukherjee5/harvard\_hw/transactions.txt").toDF("transaction\_date","time","customer\_id","product\_id","quantity\_bought","price\_paid")**

**transaction.show**

**+----------------+--------+-----------+----------+---------------+----------+**

**|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|**

**| 2015-03-30|10:35 AM| 63| 19| 5| 1023.57|**

**| 2015-03-30| 2:30 AM| 23| 77| 7| 5892.41|**

**| 2015-03-30| 7:41 PM| 49| 58| 4| 9298.18|**

**| 2015-03-30| 9:18 AM| 97| 86| 8| 9462.89|**

**| 2015-03-30|10:06 PM| 94| 26| 4| 4199.15|**

**| 2015-03-30|10:57 AM| 91| 18| 1| 3795.73|**

**| 2015-03-30| 7:43 AM| 20| 86| 10| 1477.35|**

**| 2015-03-30| 5:58 PM| 38| 39| 6| 1090.0|**

**| 2015-03-30| 1:08 PM| 46| 6| 10| 1014.78|**

**| 2015-03-30|12:18 AM| 56| 48| 9| 8346.42|**

**| 2015-03-30| 1:18 AM| 11| 58| 4| 364.59|**

**| 2015-03-30| 3:01 AM| 59| 9| 5| 5984.68|**

**| 2015-03-30|11:44 AM| 8| 35| 6| 1859.2|**

**| 2015-03-30|12:05 PM| 23| 8| 3| 1527.04|**

**| 2015-03-30| 4:10 AM| 85| 93| 9| 3314.71|**

**+----------------+--------+-----------+----------+---------------+----------+**

**only showing top 20 rows**

**//find 5 customers with the largest spent on the day,aggregate and find sum for the price\_paid per customer by grouping by customer\_id**

**transaction.createOrReplaceTempView("transaction")**

**products.createOrReplaceTempView("products")**

**//Checking how many dates**

**sql("select distinct transaction\_date from transaction").show**

**+----------------+**

**|transaction\_date|**

**+----------------+**

**| 2015-03-30|**

**+----------------+**

**sql("select sum(price\_paid) sum,customer\_id from transaction group by customer\_id order by sum desc").show(5)**

**+------------------+-----------+**

**| sum|customer\_id|**

**+------------------+-----------+**

**|100049.00000000001| 76|**

**| 88829.76000000001| 53|**

**| 85906.94| 56|**

**| 83312.12| 51|**

**| 83202.61| 31|**

**+------------------+-----------+**

**only showing top 5 rows**

**//Find the names of the products each of those 5 customers bought**

**custWithTopPurchases.createOrReplaceTempView("custWithTopPurchases")**

**sql("select distinct product\_name from products p,transaction t,custWithTopPurchases c where p.product\_id = t.product\_id and c.customer\_id = t.customer\_id").show**

**+--------------------+**

**| product\_name|**

**+--------------------+**

**| Gabapentin|**

**| LEGO Minifigures|**

**| CUPRUM METALLICUM|**

**| LEGO Minecraft|**

**|ATOPALM MUSCLE AN...|**

**|Far Cry 4 Limited...|**

**|healthy accents s...|**

**|Brimonidine Tartrate|**

**|Treatment Set TS3...|**

**|Star Wars Republi...|**

**|Essentials Dantes...|**

**|AMBROSIA TRIFIDA ...|**

**| Acyclovir|**

**| Grippe|**

**| Ativan|**

**| LEGO The Hobbit|**

**| LEGO Jurassic World|**

**|Santalia Clinical...|**

**|Essentials Medal ...|**

**| Alphanate|**

**+--------------------+**

**only showing top 20 rows**

**//Find the names and total number sold of 10 most popular products. I am assuming a product is considered popular if it has more transactions,**

**val mostPopularProducts = sql("select product\_name,count(t.product\_id) count from products p,transaction t where p.product\_id = t.product\_id group by t.product\_id,product\_name order by count desc").limit(10)**

**mostPopularProducts.show(false)**

**+--------------------------------------------+-----+**

**|product\_name |count|**

**+--------------------------------------------+-----+**

**|Notebook Lenovo U330p, 59-390439 |39 |**

**|SAMSUNG LED TV 39F5500, Full HD, USB |25 |**

**|Far Cry 4 Limited Edition for Xbox One |19 |**

**|Jafra |19 |**

**|Jantoven |17 |**

**|PC HP 600PD TWR, E7P49AW |16 |**

**|Procesor Intel Core i5 3470 |16 |**

**|ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU|16 |**

**|Notebook Lenovo U430p, 59-390459 |16 |**

**|Bear doll |15 |**

**+--------------------------------------------+-----+**

**mostPopularProducts.createOrReplaceTempView("mostPopularProducts")**

**//Order products once per the number sold and then by the total value (quanity\*price) sold.**

**//assuming we are doing for popular products with higher transactions. There was a little confusion around what qualifies for a popular transaction**

**sql("select product\_name,sum(quantity\_bought) totalSold from transaction t,mostPopularProducts m,products p where m.product\_id = p.product\_id and t.product\_id = m.product\_id group by p.product\_id,product\_name order by totalSold desc").show(false)**

**+--------------------------------------------+---------+**

**|product\_name |totalSold|**

**+--------------------------------------------+---------+**

**|Notebook Lenovo U330p, 59-390439 |226 |**

**|SAMSUNG LED TV 39F5500, Full HD, USB |142 |**

**|Jantoven |102 |**

**|Jafra |102 |**

**|Far Cry 4 Limited Edition for Xbox One |101 |**

**|Procesor Intel Core i5 3470 |90 |**

**|Bear doll |81 |**

**|Notebook Lenovo U430p, 59-390459 |79 |**

**|ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU|75 |**

**|PC HP 600PD TWR, E7P49AW |74 |**

**+--------------------------------------------+---------+**

**//order for the whole set not just the ones with higher transactions**

**sql("select product\_name,sum(quantity\_bought) totalSold from transaction t,products p where t.product\_id = p.product\_id group by p.product\_id,product\_name order by totalSold desc").show(false)**

**+----------------------------------------------------+---------+**

**|product\_name |totalSold|**

**+----------------------------------------------------+---------+**

**|Notebook Lenovo U330p, 59-390439 |226.0 |**

**|SAMSUNG LED TV 39F5500, Full HD, USB |142.0 |**

**|Jantoven |102.0 |**

**|Jafra |102.0 |**

**|Far Cry 4 Limited Edition for Xbox One |101.0 |**

**|Roller Derby Roller Street Series Skateboard Bruiser|91.0 |**

**|Procesor Intel Core i5 3470 |90.0 |**

**|Sony Playstation 3 |88.0 |**

**|chest congestion |84.0 |**

**|Barbie Beach Ken Doll |82.0 |**

**|Bear doll |81.0 |**

**|LG LED TV 32LN575S |80.0 |**

**|GAM X360 Need for Speed Most Wanted 2012 |80.0 |**

**|Notebook Lenovo U430p, 59-390459 |79.0 |**

**|LEGO Castle |79.0 |**

**|LEGO Mixels |79.0 |**

**|PC HP 490PD MT, D5T60EA |77.0 |**

**|PC HP 490PD MT, D5T68EA |76.0 |**

**|ZOCOR |75.0 |**

**|ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU |75.0 |**

**+----------------------------------------------------+---------+**

**val productNumberSold = sql("select p.product\_id,product\_name,sum(quantity\_bought \* unit\_price) totalValue from products p, transaction t where p.product\_id = t.product\_id group by p.product\_id,p.product\_name order by totalValue desc").show(false)**

**+----------+----------------------------------------------------+------------------+**

**|product\_id|product\_name |totalValue |**

**+----------+----------------------------------------------------+------------------+**

**|1 |ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU |729141.7499999999 |**

**|65 |Roller Derby Roller Street Series Skateboard Bruiser|708324.89 |**

**|7 |LEGO Mixels |688951.89 |**

**|50 |LG LED TV 32LN575S |670394.3999999999 |**

**|98 |Gabapentin |587159.1900000001 |**

**|22 |LEGO Speed Champion |560103.72 |**

**|16 |LEGO Classic |556264.7999999999 |**

**|69 |ibuprofen |545597.49 |**

**|37 |GAM X360 Need for Speed Most Wanted 2012 |543217.6 |**

**|89 |Glipizide |534457.0800000001 |**

**|100 |ZOCOR |528041.9999999999 |**

**|46 |SAMSUNG LED TV 32F4510, HD ready, USB |519042.28999999986|**

**|51 |Essentials Tekken 6 PS3 |497011.20000000007|**

**|62 |PC HP 490PD MT, D5T60EA |481123.72 |**

**|59 |PC HP 600PD TWR, E7P49AW |468175.80000000005|**

**|47 |SAMSUNG LED TV 55F6500, Full HD, 3D, USB |468075.57 |**

**|97 |Santalia Clinical Intenstive Spot Treatment |459447.04000000004|**

**|48 |LG LED TV 42LA6130 |456637.5 |**

**|41 |Star Wars Republic Commando PC |451027.20000000007|**

**|38 |Sony Playstation 3 |447774.8 |**

**+----------+----------------------------------------------------+------------------+**

**Problem 4**. Implement problem 3 using RDD APIs.

**(20%)**

**//RDD**

**//Bring those data in Spark and organize it with named columns.**

**val productRDD = sc.textFile("/Users/smukherjee5/harvard\_hw/products.txt")**

**productRDD.map(x => x.split("#")).take(10)**

**res1: Array[Array[String]] = Array(Array(1, ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU, 9721.89, 10), Array(2, Mattel Little Mommy Doctor Doll, 6060.78, 6), Array(3, Cute baby doll, battery, 1808.79, 2), Array(4, Bear doll, 51.06, 6), Array(5, LEGO Legends of Chima, 849.36, 6), Array(6, LEGO Castle, 4777.51, 10), Array(7, LEGO Mixels, 8720.91, 1), Array(8, LEGO Star Wars, 7592.44, 4), Array(9, LEGO Lord of the Rings, 851.67, 2), Array(10, LEGO The Hobbit, 7314.55, 9))**

**val transactionsRDD = sc.textFile("/Users/smukherjee5/harvard\_hw/transactions.txt")**

**res8: Array[Array[String]] = Array(Array(2015-03-30, 6:55 AM, 51, 68, 1, 9506.21), Array(2015-03-30, 7:39 PM, 99, 86, 5, 4107.59), Array(2015-03-30, 11:57 AM, 79, 58, 7, 2987.22), Array(2015-03-30, 12:46 AM, 51, 50, 6, 7501.89), Array(2015-03-30, 11:39 AM, 86, 24, 5, 8370.2), Array(2015-03-30, 10:35 AM, 63, 19, 5, 1023.57), Array(2015-03-30, 2:30 AM, 23, 77, 7, 5892.41), Array(2015-03-30, 7:41 PM, 49, 58, 4, 9298.18), Array(2015-03-30, 9:18 AM, 97, 86, 8, 9462.89), Array(2015-03-30, 10:06 PM, 94, 26, 4, 4199.15), Array(2015-03-30, 10:57 AM, 91, 18, 1, 3795.73), Array(2015-03-30, 7:43 AM, 20, 86, 10, 1477.35), Array(2015-03-30, 5:58 PM, 38, 39, 6, 1090.0), Array(2015-03-30, 1:08 PM, 46, 6, 10, 1014.78), Array(2015-03-30, 12:18 AM, 56, 48, 9, 8346.42))**

**case class Product(product\_id: Int,product\_name:String,unit\_price:Double,quanitity\_available:Int)**

**case class Transactions(transaction\_date:String,time:String,customer\_id:Int,product\_id:Int,quantity\_bought:Int,price\_paid:Double)**

**val transactions = transactionsRDD.map{x => x.split("#")}.map{columns => Transactions(columns(0), columns(1), columns(2).toInt,columns(3).toInt,columns(4).toInt,columns(5).toDouble)}**

**val products = productRDD.map{x => x.split("#")}.map{columns => Product(columns(0).toInt, columns(1), columns(2).toDouble,columns(3).toInt)}**

**//find 5 customers with the largest spent on the day.**

**val transactionsWithCustIdKey = transactions.map(t => (t.customer\_id,t.price\_paid)).reduceByKey(\_+\_).sortBy(\_.\_2,false).take(5)**

**transactionsWithCustIdKey: Array[(Int, Double)] = Array((76,100049.0), (53,88829.76), (56,85906.94), (51,83312.12), (31,83202.61))**

**//Find the names of the products each of those 5 customers bought.**

**val transactionsWithCustIdKeySet = transactionsWithCustIdKey.map(x => x.\_1).toSet**

**val namesOfProductsFor5Cust = transactions.filter(t => transactionsWithCustIdKeySet.contains(t.customer\_id)).map(t => (t.product\_id,t.customer\_id)).join(**

**val productLookup = products.map(p => (p.product\_id,p.product\_name))**

**productLookup).map(x=>x.\_2.\_2).collect**

**namesOfProductsFor5Cust: Array[String] = Array(chest congestion, Scrub Care Povidone Iodine Cleansing Scrub, Stomach Disorders, Far Cry 4 Limited Edition for Xbox One, Essentials Medal of Honor: Heroes 2 PSP, Procesor Intel Core i5 3470, LG LED TV 32LN575S, LEGO Technic, GAM X360 Hitman Absolution X360, AMBROSIA TRIFIDA POLLEN, Obao, Toothbrush, LEGO Mindstorms, LEGO Hero Factory, LEGO Hero Factory, Sony Playstation 3, GUNA-EGF, ATOPALM MUSCLE AND JOINT, LEGO The Hobbit, ZOCOR, ZOCOR, Notebook Lenovo Yoga Think, 20CD0001SC, Caldyphen Clear, Caldyphen Clear, LEGO Speed Champion, LG LED TV 42LA6130, PC HP 490PD MT, D5T60EA, PC HP 490PD MT, D5T60EA, Star Wars The Force Unleashed Ultimate Sith Edition PC, Might and Magic: Clash Of Heroes PC, Might and Magic: Clash Of Heroes PC, LEGO Castle,...**

**//Find the names and total number sold of 10 most popular products.**

**val mostPopularProducts = transactions.map(t=>(t.product\_id,1)).reduceByKey(\_+\_).join(productLookup).map(x => x.\_2).sortBy(\_.\_1,false).take(10)**

**val mostPopularProducts = transactions.map(t=>(t.product\_id,1)).reduceByKey(\_+\_).join(productLookup).sortBy(\_.\_2.\_1,false).take(10)**

**mostPopularProducts: Array[(Int, (Int, String))] = Array((58,(39,Notebook Lenovo U330p, 59-390439)), (44,(25,SAMSUNG LED TV 39F5500, Full HD, USB)), (28,(19,Far Cry 4 Limited Edition for Xbox One)), (93,(19,Jafra)), (86,(17,Jantoven)), (30,(16,Procesor Intel Core i5 3470)), (57,(16,Notebook Lenovo U430p, 59-390459)), (1,(16,ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU)), (59,(16,PC HP 600PD TWR, E7P49AW)), (4,(15,Bear doll)))**

**//Order products once per the number sold**

**//Assuming max transactions make a product popular**

**val mostPopularProductsSortByNumSold = transactions.map(t=>(t.product\_id,t.quantity\_bought)).filter(t => mostPopularProducts.map(\_.\_1).toSet.contains(t.\_1)).reduceByKey(\_+\_).join(productLookup).sortBy(\_.\_2.\_1,false).map(\_.\_2)**

**mostPopularProductsSortByNumSold: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[334] at map at <console>:41**

**mostPopularProductsSortByNumSold.collect**

**res39: Array[(Int, String)] = Array((226,Notebook Lenovo U330p, 59-390439), (142,SAMSUNG LED TV 39F5500, Full HD, USB), (102,Jantoven), (102,Jafra), (101,Far Cry 4 Limited Edition for Xbox One), (90,Procesor Intel Core i5 3470), (81,Bear doll), (79,Notebook Lenovo U430p, 59-390459), (75,ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU), (74,PC HP 600PD TWR, E7P49AW)**

**//Order For the whole set**

**val mostPopularProductsSortByNumSold = transactions.map(t=>(t.product\_id,t.quantity\_bought)).reduceByKey(\_+\_).join(productLookup).sortBy(\_.\_2.\_1,false).map(\_.\_2)**

**mostPopularProductsSortByNumSold.collect**

**res86: Array[(Int, String)] = Array((226,Notebook Lenovo U330p, 59-390439), (142,SAMSUNG LED TV 39F5500, Full HD, USB), (102,Jantoven), (102,Jafra), (101,Far Cry 4 Limited Edition for Xbox One), (91,Roller Derby Roller Street Series Skateboard Bruiser), (90,Procesor Intel Core i5 3470), (88,Sony Playstation 3), (84,chest congestion), (82,Barbie Beach Ken Doll), (81,Bear doll), (80,LG LED TV 32LN575S), (80,GAM X360 Need for Speed Most Wanted 2012), (79,LEGO Castle), (79,Notebook Lenovo U430p, 59-390459), (79,LEGO Mixels), (77,PC HP 490PD MT, D5T60EA), (76,PC HP 490PD MT, D5T68EA), (75,ZOCOR), (75,ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU), (74,PC HP 600PD TWR, E7P49AW), (70,GAM X360 Assassins Creed 3), (69,Scrub Care Povidone Iodine Cleansing Scrub), (69,Notebook Lenovo E545 20B2000KS...**

**//and then by the total value (quanity\*price) sold.**

**val mostPopularProductsSortByTotalValue = products.map(t=>(t.product\_id,(t.product\_name,t.unit\_price))).join( transactions.map(t=>(t.product\_id,t.quantity\_bought))).map(t=> (t.\_1,(t.\_2.\_1.\_2\*t.\_2.\_2))).reduceByKey(\_+\_).join(productLookup).map(\_.\_2).sortBy(\_.\_1,false)**

**mostPopularProductsSortByTotalValue.collect**

**res84: Array[(Double, String)] = Array((729141.7499999999,ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU), (708324.8899999998,Roller Derby Roller Street Series Skateboard Bruiser), (688951.8900000001,LEGO Mixels), (670394.4000000001,LG LED TV 32LN575S), (587159.1899999998,Gabapentin), (560103.7200000001,LEGO Speed Champion), (556264.7999999999,LEGO Classic), (545597.49,ibuprofen), (543217.6,GAM X360 Need for Speed Most Wanted 2012), (534457.0800000001,Glipizide), (528042.0,ZOCOR), (519042.29000000004,SAMSUNG LED TV 32F4510, HD ready, USB), (497011.20000000007,Essentials Tekken 6 PS3), (481123.7199999999,PC HP 490PD MT, D5T60EA), (468175.80000000005,PC HP 600PD TWR, E7P49AW), (468075.57,SAMSUNG LED TV 55F6500, Full HD, 3D, USB), (459447.0400000001,Santalia Clinical Intenstive Spot Treatmen...**

I verified that all my RDD and Dataframe values match

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used as separate files. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portion of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. Please, submit to the class drop box. For issues and comments visit the class Discussion Board. You can solve these problems using any language of your choice.