

DS503 Project 3 Report

Team3: Wenhan Ji, Hao Yin

Problem1

First, Created two datasets, one is Customer, the other is Purchase. Then, in spark shell, we load data in.

(1) In Purchase dataset, we use ‘filter’ on ‘Purchase.TransTotal’ to get the result.

TransID	CustID	TransTotal	TransNumItems	TransDesc
2	24796	991	9	rueksycisspzwahbx...
4	17148	606	2	mosadcadqpcawoejj...
5	6006	651	2	azhgvoexgkzzrefr...
9	47055	837	5	rwsjeahrldbggvmlf...
10	14692	667	8	uvrmvoasoqtviecdb...
12	26925	699	1	wnrcmzidmvxuvkgf...
13	42681	694	1	vyptvhfozvuhxraru...
14	14087	604	9	rzclljdcmonhkbzl...
15	11762	916	5	tvqxwqfgdtyrfdjwp...
17	2079	747	1	gvtfoszdzxrbszke...
18	28701	602	3	lybnlkgpfqhvvcbn...
19	36492	998	8	rbbhximkmzzyihvy...
23	32950	805	6	kffaecmrmtmkzidrki...
24	7170	810	4	wvbeoypkzzyinqae...
25	31264	956	5	xlgmhauwkmgjqxdn...
29	19617	560	7	kyjfxfnrlvnbubaor...
30	20744	987	8	xvierboaouqgbardw...
31	35958	528	7	gmhuzqwsleiykxml...
32	47840	748	2	cdfsbewxoytppikw...
33	11930	775	1	yctpkbaezgvoagkfkpy

only showing top 20 rows

(2) Using the result from (1), we group by TransNumItems and for each group calculate the average, min and max of TransTotal.

(3) By using the “.show()”, we display the result.

TransNumItems	avg(TransTotal)	min(TransTotal)	max(TransTotal)
1	750.4266732030536	501	1000
6	749.9990555255722	501	1000
3	750.6852796696647	501	1000
5	750.6498371979072	501	1000
9	750.4733771663873	501	1000
4	750.7730768159703	501	1000
8	750.5196447782643	501	1000
7	750.6331674394315	501	1000
10	750.6502201172899	501	1000
2	750.4354705138823	501	1000

(4) In this case, First, In Customer dataset, we select the “ID” and ” Age” columns with the age over 10 and less than 20 years old. Second, In Purchase1 dataset which is the result of (1), we group by “CustID” and for each group calculate sum of TransNumItems and TransTotal. Third, we join the two data and get want we want.

ID	Age	sum(TransNumItems)	sum(TransTotal)
6397	19	353	44047
45307	13	325	42875
33412	17	211	32779
31261	13	239	37761
28759	15	267	37301
17679	11	257	37253
44437	11	294	36592
16574	12	247	31537
1591	16	197	28089
30970	11	217	30610
19204	13	304	45116
13840	16	402	48324
44358	14	259	33707
3175	16	283	40155
44906	14	265	35399
1088	11	240	32406
14450	11	284	36306
16386	15	264	39213
32445	11	288	35815
3997	19	332	41858

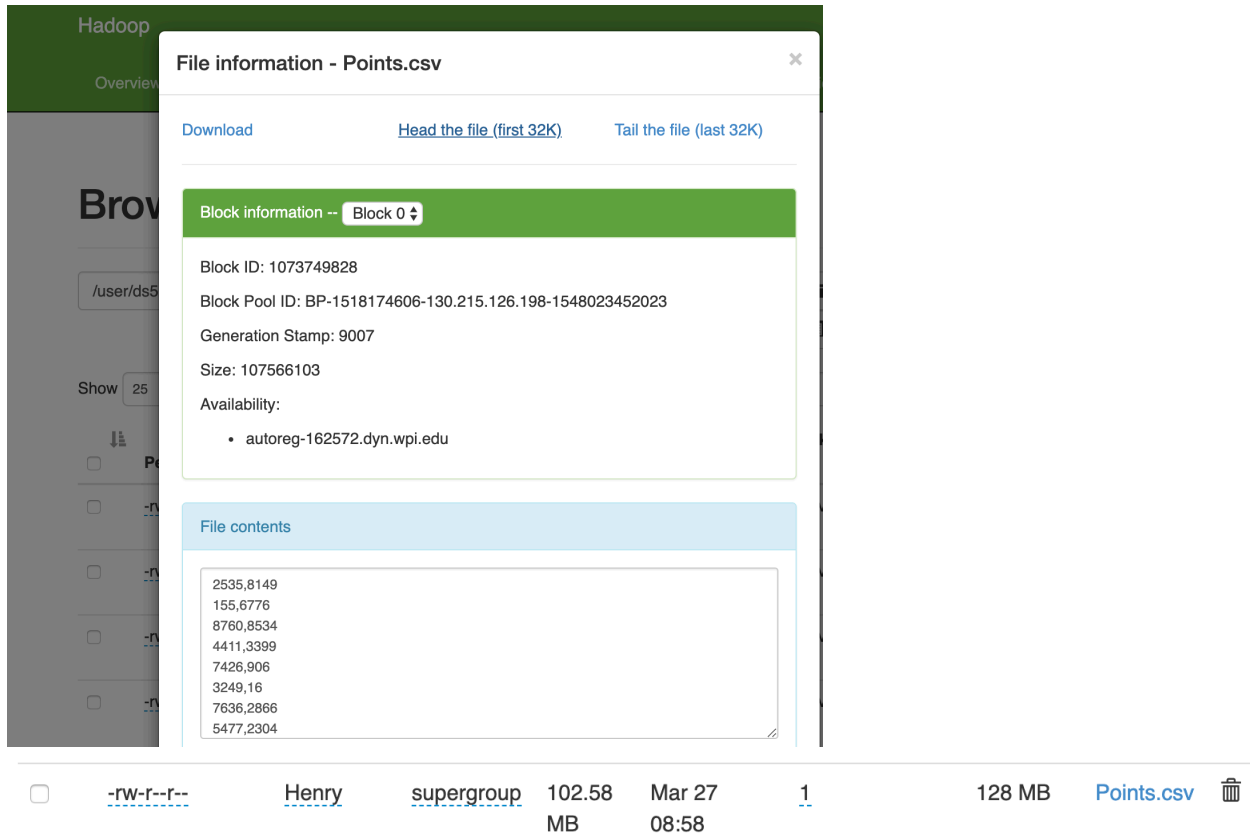
only showing top 20 rows

(5) In this problem, we build an exact same dataframe as “groupdata”, In order to distinguish them and do join operation, we rename the columns of two datasets. In Join operation, for each id in one dataset, join it with all the id in another dataset. The condition is for id1 and id2, age1 is less than age2 and total items 1 are less than total items 2 but the total amount1 is greater than total amount 2. So what we got is a big data frame with all the information fulfill the conditions we seted.

(6)select the column we need in (5), and display the result by “.show()”

C1_ID	C2_ID	Age1	Age2	TotalAmount1	TotalAmount2	TotalItemCount1	TotalItemCount2
45307	3997	13	19	42875	41858	325	332
33412	38153	17	18	32779	26488	211	213
31261	28759	13	15	37761	37301	239	267
31261	44358	13	14	37761	33707	239	259
31261	44906	13	14	37761	35399	239	265
31261	28124	13	16	37761	35407	239	268
28759	28124	15	16	37301	35407	267	268
17679	44358	11	14	37253	33707	257	259
17679	44906	11	14	37253	35399	257	265
17679	29744	11	12	37253	36011	257	278
17679	1829	11	12	37253	33875	257	277
17679	28124	11	16	37253	35407	257	268
1591	38153	16	18	28089	26488	197	213
19204	6397	13	19	45116	44047	304	353
19204	3997	13	19	45116	41858	304	332
3175	38220	16	17	40155	39550	283	301
1088	16574	11	12	32406	31537	240	247
16386	28124	15	16	39213	35407	264	268
16386	34234	15	17	39213	38327	264	274
38422	28124	15	16	38515	35407	266	268

P2.A: Load dataset into HDFS



The screenshot shows the Hadoop file browser interface. A modal window titled "File information - Points.csv" is open, displaying details for a file named "Points.csv". The modal includes links for "Download", "Head the file (first 32K)", and "Tail the file (last 32K)". Below these links is a section for "Block information" for "Block 0", showing the following details:

- Block ID: 1073749828
- Block Pool ID: BP-1518174606-130.215.126.198-1548023452023
- Generation Stamp: 9007
- Size: 107566103
- Availability:
 - autoreg-162572.dyn.wpi.edu

Below the block information is a section for "File contents" showing a list of coordinates:

```
2535,8149
155,6776
8760,8534
4411,3399
7426,906
3249,16
7636,2866
5477,2304
```

The background shows a file listing table with columns for permissions, owner, group, size, modification time, and filename. The file "Points.csv" is listed with a size of 128 MB and a modification time of Mar 27 08:58.

P2.B: Report the TOP 10 grid cells with highest Relative-Density Scores

script_scala: Time: 19.88s

script_py: Time: 46.05s

Result can be seen in ipynb 8

(1) First, we parse the point. Use map to map string “x,y” to tuple (x, y)

```
[[2535, 8149], [155, 6776]]
```

(2) Then compute cell points count.

We define a function that input point (x, y) and return it’s cell ID.

For each point, we set key as it’s cell id and value to be 1.

Then we reduce by key and sum the values. In this way, we can get how many points in each cell.

```
[(80508, 46), (36938, 49)]
```

(3) Compute cell neighbor count. (How many neighbours a cell has)

We write a function input cell id, output it’s neighbor id list. The list length is the neighbor count.

[[80508, 8], [36938, 8]]

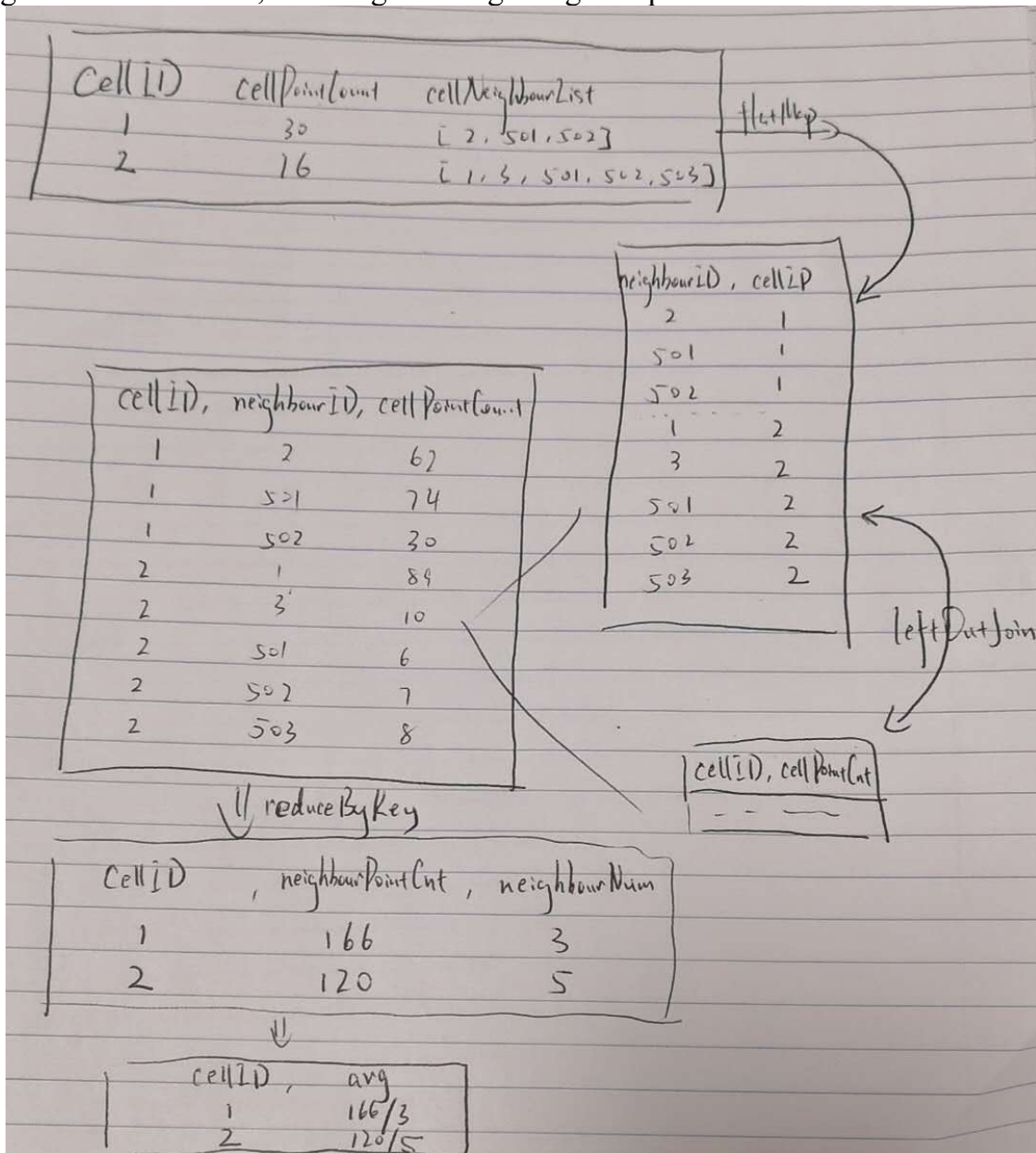
(4) Compute cell neighbours average points count

First, we compute cell ID and it's neighbor list.

Then do flatMap and set each neighbor as key.

Then join this table with cellPointCount table on neighborID, we will know how many points each neighbor has.

Then using reduceByKey, we can get sum of neighbor points. We divide it by number of neighbours. As a result, we can get average neighbor points for each cell.



```
[(80508, 45.375), (228372, 40.25)]
```

(5) Compute cell density

We join cellPointCount table with cell neighbor average points table. Then divide them and get cell density. We sort by density and get first 10.

```
[(44947, 2.032154340836013),  
(68303, 1.8313253012048192),  
(46501, 1.8206521739130437),  
(241132, 1.7969230769230768),  
(104811, 1.7655172413793103),  
(6457, 1.7610062893081762),  
(55493, 1.7476923076923077),  
(226717, 1.741046831955923),  
(84024, 1.7379310344827585),  
(142898, 1.732484076433121)]
```

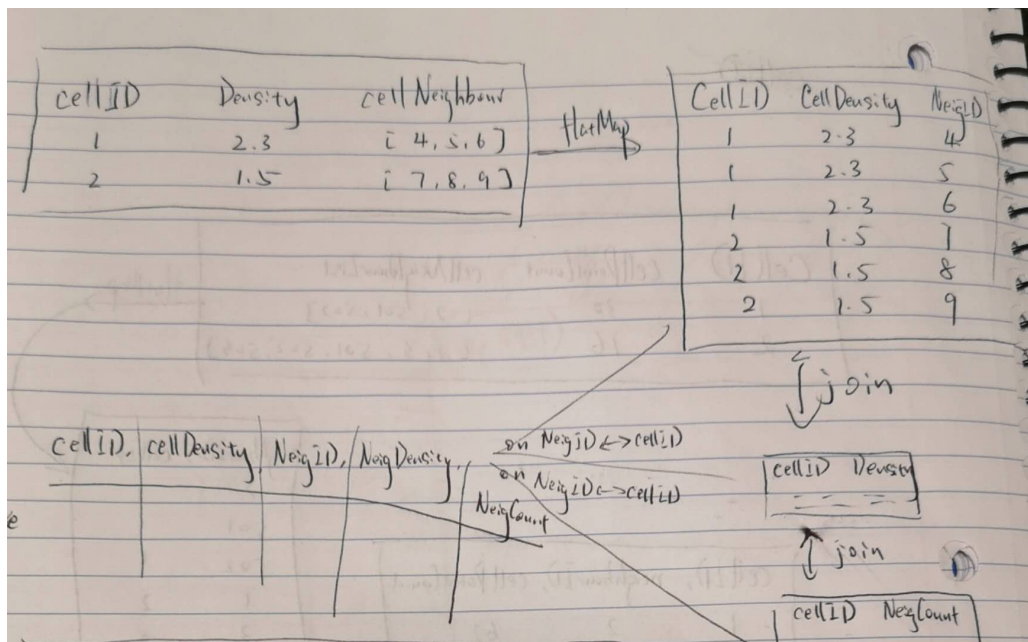
P2.C: Report the TOP k grid cells w.r.t their Relative-Density Scores

script_scala: Time: 2.22s

script_py: Time: 3.18s

Result can be seen in ipynb 9

For top10 dell density, we compute cell's neighbor. Then flatMap, so each line contains one neighbor. We join cell density and cell neighbor count on neighbor ID. Then get the result.



Following shows the result, with 4 cell and it's neighbors density information.
(cellID, cellDensity, numOfNeighbours, neighbourID, neighbourDensity)

(6457,1.7610062893081762,8,5956,0.8465608465608465)
(6457,1.7610062893081762,8,6956,0.6736842105263158)
(6457,1.7610062893081762,8,6456,0.8467966573816156)
(6457,1.7610062893081762,8,6957,1.0386740331491713)
(6457,1.7610062893081762,8,5957,0.7466666666666667)
(6457,1.7610062893081762,8,5958,0.8541666666666666)
(6457,1.7610062893081762,8,6958,1.0162162162162163)
(6457,1.7610062893081762,8,6458,0.8085106382978723)
(44947,2.032154340836013,8,44448,0.6253369272237197)
(44947,2.032154340836013,8,44948,1.0823529411764705)
(44947,2.032154340836013,8,45448,0.6772486772486772)
(44947,2.032154340836013,8,44946,0.7411444141689373)
(44947,2.032154340836013,8,45446,0.9824561403508771)
(44947,2.032154340836013,8,44446,1.1685393258426966)
(44947,2.032154340836013,8,45447,0.7091412742382271)
(44947,2.032154340836013,8,44447,0.9859943977591037)
(46501,1.8206521739130437,5,47001,0.7296137339055794)
(46501,1.8206521739130437,5,46001,1.0267857142857144)
(46501,1.8206521739130437,5,46002,0.6067415730337079)
(46501,1.8206521739130437,5,47002,0.8291316526610645)
(46501,1.8206521739130437,5,46502,1.032258064516129)
(55493,1.7476923076923077,8,54992,0.7021276595744681)
(55493,1.7476923076923077,8,55992,0.9772727272727273)
(55493,1.7476923076923077,8,55492,0.7588075880758808)
(55493,1.7476923076923077,8,54993,1.0481283422459893)
(55493,1.7476923076923077,8,55993,0.935933147632312)
(55493,1.7476923076923077,8,55994,0.7253886010362695)
(55493,1.7476923076923077,8,55494,0.9865951742627346)
(55493,1.7476923076923077,8,54994,0.8571428571428571)

Skills:

Wenhan Ji: Python, Java, R, SQL
Skills Aquired: Scala

Hao Yin: Python, Java, SQL, Scala
Skills Aquired: Scala

Contributions:

Wenhan Ji: Mainly work on problem2. Assisted peers with problem1.
Hao Yin: Mainly work on problem1. Assisted peers with problem2.

Both of us run 2 problems in our pc.