

La Trobe University

Department of Computer Science and Computer Engineering

CSE5BDC Assignment 2024

Objectives

1. Gain in depth experience playing around with big data tools (Hive, SparkRDDs, and Spark SQL).
2. Solve challenging big data processing tasks by finding highly efficient solutions.
3. Experience processing three different types of real data
 - a. Standard multi-attribute data (Bank data)
 - b. Time series data (Twitter feed data)
 - c. Bag of words data.
4. Practice using programming APIs to find the best API calls to solve your problem. Here are the API descriptions for Hive, Spark (especially spark look under RDD. There are a lot of really useful API calls).
 - a) [Hive] <https://cwiki.apache.org/confluence/display/Hive/LanguageManual>
 - b) [Spark] <http://spark.apache.org/docs/latest/api/scala/index.html#package>
 - c) [Spark SQL] <https://spark.apache.org/docs/latest/sql-programming-guide.html>
<https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Database>
<https://spark.apache.org/docs/latest/api/sql/index.html>

- If you are not sure what a spark API call does, try to write a small example and try it in the spark shell

This assignment is due 11:59 pm on Friday 24th of May, 2024.

Penalties are applied to late assignments (accepted up to 5 business days after the due date only). Five percent is deducted per business day late. A mark of zero will be assigned to assignments submitted more than 5 days late.

This is an **individual** assignment. You are not permitted to work as a part of a group when writing this assignment.

Submission checklist

- Ensure that all of your solutions read their input from the full data files (not the small example versions)
- Check that all of your solutions run without crashing in the docker containers that you used in the labs.
- Delete all output files
- Archive up everything into a single zip file and submit your assignment via LMS

Copying, Plagiarism

Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. For individual assignments, plagiarism includes the case where two or more students work collaboratively on the assignment. The Department of Computer Science and Computer Engineering treats plagiarism very seriously. When it is detected, penalties are strictly imposed.

If you are working on your assignment on the lab computers. Make sure you delete the virtual machine and empty the recycle bin before you leave. Otherwise, other students may be able to see your solutions.

ChatGPT and similar AI tools

A key purpose of this assessment task is to test your own ability to complete the assigned tasks. Therefore, the use of ChatGPT, AI tools or chatbots with similar functionality is prohibited for this assessment task. Students who are found to be in breach of this rule will be subject to normal academic misconduct measures. Additionally, students may be engaged to provide an oral validation of their understanding of their submitted work (e.g. coding).

Expected quality of solutions

- a) In general, writing more efficient code (less reading/writing from/into HDFS and less data shuffles) will be rewarded with more marks.
- b) This entire assignment can be done using the docker containers supplied in the labs and the supplied data sets **without running out of memory**. It is time to show your skills!
- c) I am not too fussed about the layout of the output. As long as it looks similar to the example outputs for each task. That will be good enough. The idea is not to spend too much time massaging the output to be the right format but instead to spend the time to solve problems.
- d) For Hive queries. We prefer answers that use less tables.

The questions in the assignment will be labelled using the following:

- [Hive]
 - Means this question needs to be done using Hive
- [Spark RDD]
 - Means this question needs to be done using Spark RDDs, you are **not allowed** to use any Spark SQL features like dataframe or datasets.
- [Spark SQL]
 - Means this question needs to be done using Spark SQL and therefore you are not allowed to use RDDs. In addition, you need to do these questions using the spark dataframe or dataset API, do not use SQL syntax.

Assignment structure:

- A script which puts all of the data files into HDFS automatically is provided for you. Whenever you start the docker container again you will need to run the following script to upload the data to HDFS again, since HDFS state is not maintained across docker runs:

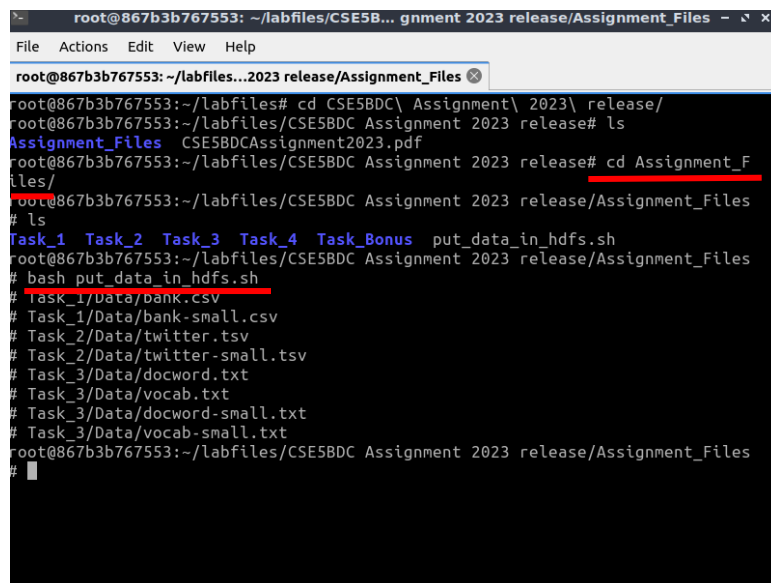
```
$ bash put_data_in_hdfs.sh
```

The script will output the names of all of the data files it copies into HDFS. If you do not run this script, solutions to the Spark questions **will not work** since they load data from HDFS.

To put the files onto HDFS do the following:

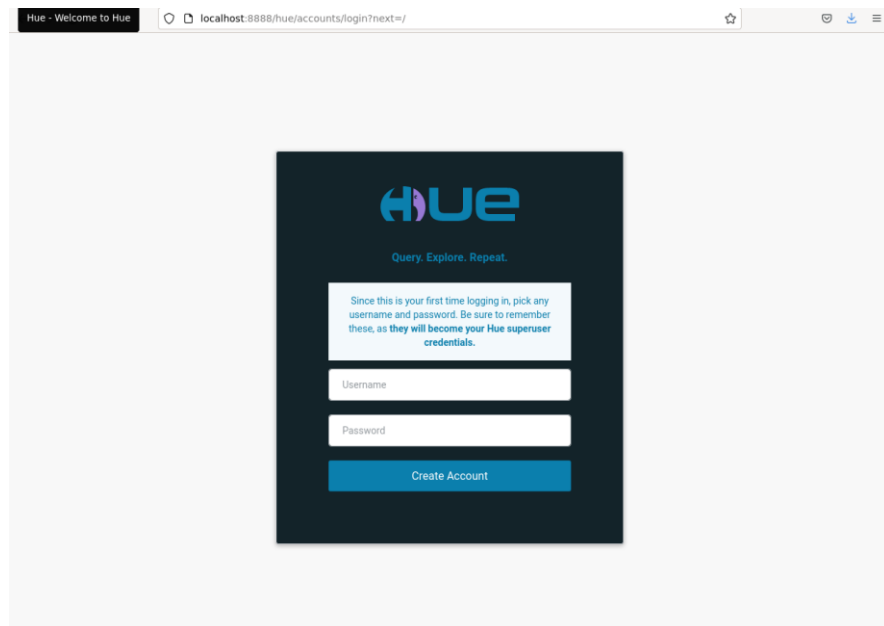
1. First start the docker container using **run.sh** like you have done for your labs.
2. Change to the directory that contains the file **put_data_in_hdfs.sh** file and then run the following command:

```
bash put_data_in_hdfs.sh
```

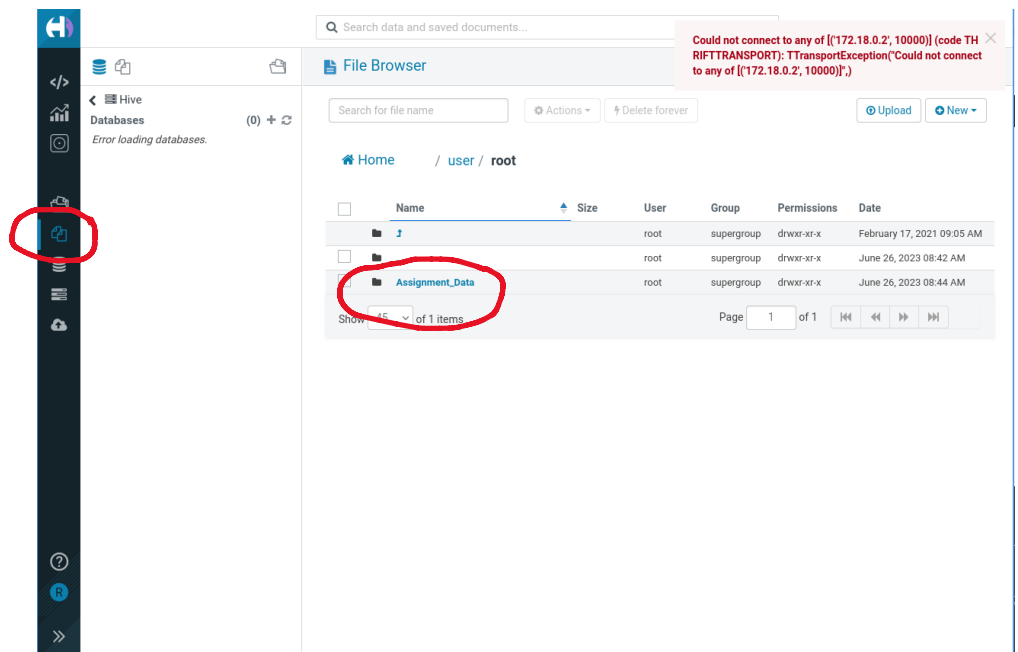
A terminal window screenshot showing the execution of the script. The prompt is root@867b3b767553: ~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files. The user runs 'ls' showing files like CSE5BDCAssignment2023.pdf. Then they run 'cd Assignment_Files/' and 'ls' again, showing files like Task_1, Task_2, Task_3, Task_4, Task_Bonus, and put_data_in_hdfs.sh. Finally, they run 'bash put_data_in_hdfs.sh' which outputs a list of files being uploaded to HDFS: Task_1/Data/bank.csv, Task_1/Data/bank-small.csv, Task_2/Data/twitter.tsv, Task_2/Data/twitter-small.tsv, Task_3/Data/docword.txt, Task_3/Data/vocab.txt, Task_3/Data/docword-small.txt, and Task_3/Data/vocab-small.txt.

```
root@867b3b767553: ~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
File Actions Edit View Help
root@867b3b767553: ~/labfiles...2023 release/Assignment_Files
root@867b3b767553:~/labfiles# cd CSE5BDC\ Assignment\ 2023\ release/
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release# ls
Assignment_Files CSE5BDCAssignment2023.pdf
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release# cd Assignment_F
iles/
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
# ls
Task_1 Task_2 Task_3 Task_4 Task_Bonus put_data_in_hdfs.sh
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
# bash put_data_in_hdfs.sh
# Task_1/Data/bank.csv
# Task_1/Data/bank-small.csv
# Task_2/Data/twitter.tsv
# Task_2/Data/twitter-small.tsv
# Task_3/Data/docword.txt
# Task_3/Data/vocab.txt
# Task_3/Data/docword-small.txt
# Task_3/Data/vocab-small.txt
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
#
```

3. The above will put all the assignment files into HDFS. You can now look at the HDFS contents in Hue like the following. Open Firefox browser and type in the following URL: **localhost:8888**



4. Type in username: **root** and password: **root**
5. Next select the files icon on the left to see the files you have uploaded to HDFS.



- For each Hive question a skeleton .hql file is provided for you to write your solution in. You can run these just like you did in labs:

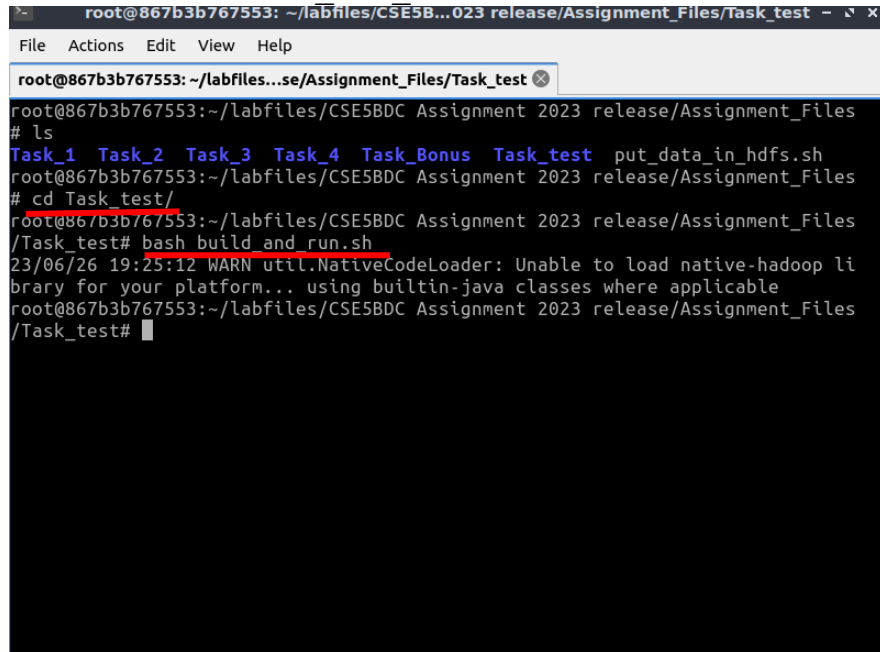
```
$ hive -f Task_XX.hql
```
- For each Spark question, a skeleton project is provided for you. Write your solution in the **.scala** file in the **src** directory. Build and run your Spark code using the provided scripts:

```
$ bash build_and_run.sh
```

Follow the instructions below to run a small test program that outputs to HDFS so you can see the output.

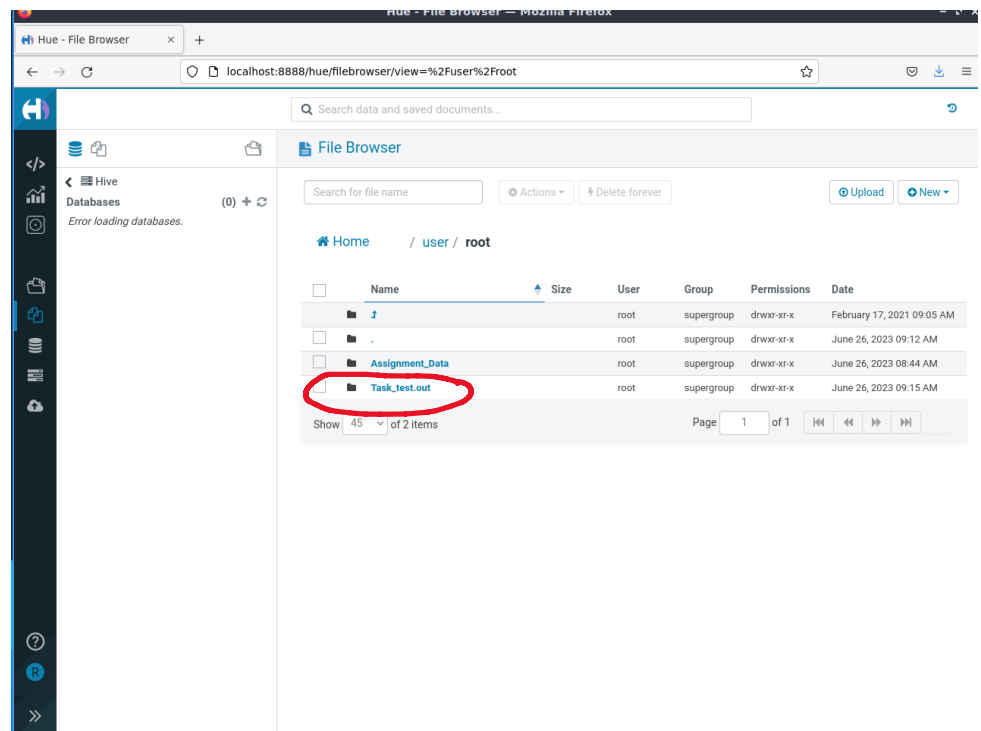
1. Change to the Task_test directory and type the following command:

```
bash build_and_run.sh
```

A terminal window screenshot with a dark background. The title bar shows the path ~/labfiles/CSE5B...023 release/Assignment_Files/Task_test. The terminal text shows the user navigating to the Task_test directory and running the script 'bash build_and_run.sh'. The output of the script is visible, including a warning message about the native-hadoop library. The prompt is currently at the Task_test directory.

```
root@867b3b767553: ~/labfiles/CSE5B...023 release/Assignment_Files/Task_test - x
File Actions Edit View Help
root@867b3b767553: ~/labfiles...se/Assignment_Files/Task_test x
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
# ls
Task_1 Task_2 Task_3 Task_4 Task_Bonus Task_test put_data_in_hdfs.sh
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
# cd Task_test/
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
/Task_test# bash build_and_run.sh
23/06/26 19:25:12 WARN util.NativeCodeLoader: Unable to load native-hadoop li
brary for your platform... using builtin-java classes where applicable
root@867b3b767553:~/labfiles/CSE5BDC Assignment 2023 release/Assignment_Files
/Task_test#
```

2. Next look at the output of the program in Hue



Tips:

1. Look at the data files **before** you begin each task. Try to understand what you are dealing with!
2. For each subtask we provide small example input and the corresponding output in the assignment specifications below. These small versions of the files are also supplied with the assignment (they have “-small” in the name). It’s a good idea to get your solution working on the small inputs first before moving on to the full files.
3. In addition to testing the correctness of your code using the very small example input. You should also use the large input files that we provide to test the scalability of your solutions.
4. It can take some time to build and run Spark applications from `.scala` files. So for the Spark questions it’s best to experiment using *spark-shell* first to figure out a working solution, and then put your code into the `.scala` files afterwards. As an example you can try to copy the following highlighted lines from the **Task_test** source file into the spark shell.

home > osboxes > CSE58DC Assignment 2023 release > Assignment_Files > Task_test > src > Task_test.scala

```
1 import org.apache.spark.sql._
2 import org.apache.spark.sql.types._
3 import org.apache.spark.SparkContext
4
5 object Main {
6   def solution(sc: SparkContext) {
7
8     // WARNING: Please DO NOT modify the line below.
9     // Please make sure you first copy the data to HDFS follow instructions
10    // on assignment specifications.
11    // The code below loads each line of the input data from HDFS.
12    val bankdataLines = sc.textFile("Assignment_Data/bank-small.csv")
13
14    // Split each line of the input data into an array of strings
15    val bankdata = bankdataLines.map(_.split(";"))
16
17    // Code that
18    val result = bankdata.map(line => (line(0), line(1), line(3), line(4)));
19    result.saveAsTextFile("Task_test.out")
20  }
21}
```

Task 1: Analysing Bank Data [32 marks total]

We will be doing some analytics on real data from a Portuguese banking institution¹. The data is stored in a semicolon (“;”) delimited format.

The data is supplied with the assignment at the following locations:

Small version	Full version
Task_1/Data/bank-small.csv	Task_1/Data/bank.csv

The data has the following attributes

Attribute index	Attribute name	Description
0	age	numeric
1	job	type of job (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
2	marital	marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
3	education	(categorical: "unknown", "secondary", "primary", "tertiary")
4	default	has credit in default? (binary: "yes", "no")
5	balance	average yearly balance, in euros (numeric)
6	housing	has housing loan? (binary: "yes", "no")
7	loan	has personal loan? (binary: "yes", "no")
8	contact	contact communication type (categorical: "unknown", "telephone", "cellular")
9	day	last contact day of the month (numeric)
10	month	last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
11	duration	last contact duration, in seconds (numeric)
12	campaign	number of contacts performed during this campaign and for this client (numeric, includes last contact)
13	pdays	number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
14	previous	number of contacts performed before this campaign and for this client (numeric)
15	poutcome	outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success")
16	termdeposit	has the client subscribed a term deposit? (binary: "yes", "no")

1 : Banking data source: <http://archive.ics.uci.edu/ml/datasets/Bank+Marketing>

Here is a small example of the bank data that we will use to illustrate the subtasks below (we only list a subset of the attributes in this example, see the above table for the description of the attributes):

<u>job</u>	<u>marital</u>	<u>education</u>	<u>balance</u>	<u>loan</u>
management	Married	tertiary	2143	Yes
technician	Divorced	secondary	29	Yes
entrepreneur	Single	secondary	2	No
blue-collar	Married	unknown	1506	No
services	Divorced	secondary	829	Yes
technician	Married	tertiary	929	Yes
Management	Divorced	tertiary	22	No
technician	Married	primary	10	No

Please note we specify whether you should use [Hive] or [Spark RDD] for each subtask at the beginning of each subtask.

- a) [Hive] Report the number of clients of each job category. Write the results to "Task_1a-out". For the above small example data set you would report the following (output order is not important for this question):

```
"blue-collar" 1
"entrepreneur" 1
"management" 2
"services" 1
"technician" 3
```

[6 marks]

- b) [Hive] Report the average yearly balance for all people in each education category. Write the results to "Task_1b-out". For the small example data set you would report the following (output order is not important for this question):

```
"primary" 10.0
"secondary" 286.6666666666667
"tertiary" 1031.3333333333333
"unknown" 1506.0
```

[6 marks]

- c) [Spark RDD] Group balance into the following three categories:
- Low: -infinity to 500
 - Medium: 501 to 1500 =>
 - High: 1501 to +infinity

Report the number of people in each of the above categories. Write the results to “Task_1c-out” in text file format. For the small example data set you should get the following results (output order is not important in this question):

```
(High,2)
(Medium,2)
(Low,4)
```

[8 marks]

- d) [Spark RDD] Sort all people in ascending order of *education*. For people with the same *education*, sort them in descending order by *balance*. This means that all people with the same *education* should appear grouped together in the output. For each person report the following attribute values: *education*, *balance*, *job*, *marital*, *loan*. Write the results to “Task_1d-out” in text file format (multiple parts are allowed). For the small example data set you would report the following:

```
("primary",10,"technician","married","no")
("secondary",829,"services","divorced","yes")
("secondary",29,"technician","divorced","yes")
("secondary",2,"entrepreneur","single","no")
("tertiary",2143,"management","married","yes")
("tertiary",929,"technician","married","yes")
("tertiary",22,"management","divorced","no")
("unknown",1506,"blue-collar","married","no")
```

[12 marks]

Task 2: Analysing Twitter Time Series Data [28 marks]

In this task we will be doing some analytics on real Twitter data². The data is stored in a tab (“t”) delimited format.

The data is supplied with the assignment at the following locations:

Small version	Full version
Task_2/Data/twitter-small.tsv	Task_2/Data/twitter.tsv

The data has the following attributes

Attribute index	Attribute name	Description
0	tokenType	In our data set all rows have Token type of hashtag. So this attribute is useless for this assignment.
1	month	The year and month specified like the following: YYYYMM. So 4 digits for year followed by 2 digits for month. So like the following 200905, meaning the year 2009 and month of May
2	count	An integer representing the number tweets of this hash tag for the given year and month
3	hashtagName	The #tag name, e.g. babylove, mydate, etc.

Here is a small example of the Twitter data that we will use to illustrate the subtasks below:

Token type	Month	count	Hash Tag Name
hashtag	200910	2	babylove
hashtag	200911	2	babylove
hashtag	200912	90	babylove
hashtag	200812	100	mycoolwife
hashtag	200901	201	mycoolwife
hashtag	200910	1	mycoolwife
hashtag	200912	500	mycoolwife
hashtag	200905	23	abc
hashtag	200907	1000	abc

2 : Twitter data source: <http://www.infochimps.com/datasets/twitter-census-conversation-metrics-one-year-of-urls-hashtags-sm>

- a) [Spark RDD] Find the single row that has the highest count and for that row report the month, count and hashtag name. Print the result to the terminal output using `println`. So, for the above small example data set the result would be:

month: 200907, count: 1000, tagName: abc
--

[6 marks]

- b) [Do twice, once using Hive and once using Spark RDD] Find the hash tag name that was tweeted the most in the entire data set across all months. Report the total number of tweets for that hash tag name. You can either print the result to the terminal or output the result to a text file. So, for the above small example data set the output would be:

abc 1023

[12 marks total: 6 marks for Hive and 6 marks for Spark RDD]

- c) [Spark RDD] Given two months x and y, where $y > x$, find the hashtag name that has increased the number of tweets the most from month x to month y. Ignore the tweets in the months between x and y, so just compare the number of tweets at month x and at month y. Report the hashtag name, the number of tweets in months x and y. Ignore any hashtag names that had no tweets in either month x or y. You can assume that the combination of hashtag and month is unique. Therefore, the same hashtag and month combination cannot occur more than once. Print the result to the terminal output using `println`. For the above small example data set:

Input	x = 200910, y = 200912
Output	tagName: mycoolwife, countX: 1, countY: 500

For this subtask you can specify the months x and y as arguments to the script. This is required to test on the full-sized data. For example:

```
$ bash build_and_run.sh 200901 200902
```

[10 marks]

Task 3: Indexing Bag of Words data [20 marks]

In this task you are asked to create a partitioned index of words to documents that contain the words. Using this index you can search for all the documents that contain a particular word efficiently.

The data is supplied with the assignment at the following locations³:

Small version	Full version
Task_3/Data/docword-small.txt	Task_3/Data/docword.txt
Task_3/Data/vocab-small.txt	Task_3/Data/vocab.txt

The first file is called *docword.txt*, which contains the contents of all the documents stored in the following format:

Attribute index	Attribute name	Description
0	docId	The ID of the document that contains the word
1	vocabId	Instead of storing the word itself, we store an ID from the vocabulary file.
2	count	An integer representing the number of times this word occurred in this document.

The second file called *vocab.txt* contains each word in the vocabulary, which is indexed by vocabIndex from the *docword.txt* file.

Here is a small example content of the *docword.txt* file.

docId	vocabId	count
3	3	600
2	3	702
1	2	120
2	5	200
2	2	500
3	1	100
3	5	2000
3	4	122
1	3	1200
1	1	1000

Here is an example of the *vocab.txt* file

vocabId	word
1	plane
2	car
3	motorbike
4	truck
5	boat

Complete the following subtasks using Spark:

- a) [spark SQL] Calculate the total count of each word across all documents. List the words in ascending alphabetical order. Write the results to “Task_3a-out” in CSV format (multiple output parts are allowed). So, for the above small example input the output would be the following:

```
boat,2200
car,620
motorbike,2502
plane,1100
truck,122
```

Note: spark SQL will give the output in multiple files. You should ensure that the data is sorted globally across all the files (parts). So, all words in part 0, will be alphabetically before the words in part 1.

[5 marks]

- b) [spark SQL] Create a dataframe containing rows with four fields: (*word*, *docId*, *count*, *firstLetter*). You should add the *firstLetter* column by using a UDF which extracts the first letter of *word* as a String. Save the results in parquet format **partitioned by firstLetter** to docwordIndexFilename. Use show() to print the first 10 rows of the dataframe that you saved.

So, for the above example input, you should see the following output (the exact ordering is not important):

```
+-----+-----+-----+-----+
|      word|docId|count|firstLetter|
+-----+-----+-----+-----+
|    plane|    1| 1000|         p|
|    plane|    3|   100|         p|
|     car|    2|   500|         c|
|     car|    1|   120|         c|
|motorbike|    1| 1200|         m|
|motorbike|    2|   702|         m|
|motorbike|    3|   600|         m|
|   truck|    3|   122|         t|
|    boat|    3| 2000|         b|
|    boat|    2|   200|         b|
+-----+-----+-----+-----+
```

[9 marks]

- c) [spark SQL] Load the previously created dataframe stored in parquet format from subtask b). For each document ID in the `docIds` list (which is provided as a function argument for you), use `println` to display the following: the document ID, the word with the most occurrences in that document (you can break ties arbitrarily), and the number of occurrences of that word in the document. Skip any document IDs that aren't found in the dataset. Use an optimisation to prevent loading the parquet file into memory multiple times.

If `docIds` contains "2" and "3", then the output for the example dataset would be:

```
[2, motorbike, 702]
[3, boat, 2000]
```

For this subtask specify the document ids as arguments to the script. For example:

```
$ bash build_and_run.sh 2 3
```

[3 marks]

- d) [spark SQL] Load the previously created dataframe stored in parquet format from subtask b). For each word in the `queryWords` list (which is provided as a function argument for you), use `println` to display the `docId` with the most occurrences of that word (you can break ties arbitrarily). Use an optimisation based on how the data is partitioned.

If `queryWords` contains "car" and "truck", then the output for the example dataset would be:

```
[car,2]
[truck,3]
```

For this subtask you can specify the query words as arguments to the script. For example:

```
$ bash build_and_run.sh computer environment power
```

[3 marks]

Task 4: Creating co-occurring words from Bag of Words data [20 marks]

Using the same data as that for task 3 perform the following subtasks:

- a) [spark SQL] Remove all rows which reference infrequently occurring words from docwords. Store the resulting dataframe in Parquet format at “../frequent_docwords.parquet” **and** in CSV format at “Task_4a-out”. An infrequently occurring word is any word that appears less than 1000 times in the *entire corpus of documents*. For the small example input file the expected output is:

```
3,1,1200
3,2,702
3,3,600
5,3,2000
5,2,200
1,1,1000
1,3,100
```

[5 marks]

- b) [spark SQL] Load up the Parquet file from “../frequent_docwords.parquet” which you created in the previous subtask. Find all pairs of frequent words that occur in the same document and report the number of documents the pair occurs in. Report the pairs in decreasing order of frequency. The solution may take a few minutes to run.
- Note there should not be any replicated entries like:
 - o (truck, boat) (truck, boat)
 - Note you should not have the same pair occurring twice in opposite order. Only one of the following should occur:
 - o (truck, boat) (boat, truck)

Save the results in CSV format at “Task_4b-out”. For the example above, the output should be as follows (it is OK if the text file output format differs from that below sample, but the data contents should be the same):

```
boat, motorbike, 2
motorbike, plane, 2
boat, plane, 1
```

For example the following format and content for the text file will also be acceptable (note the order is slightly different, that is also OK since we break frequency ties arbitrarily):

```
(2,(plane, motorbike))
(2,(motorbike, boat))
(1,(plane, boat))
```

[15 marks]

Bonus Marks:

1. Using spark, perform the following task using the data set of task 2.

[Spark RDD or Spark SQL] Find the hash tag name that has increased the number of tweets the most from among any two consecutive months of any hash tag name. Consecutive month means for example, 200801 to 200802, or 200902 to 200903, etc. Report the hash tag name, the 1st month count, and the 2nd month count using `println`.

For the small example data set of task 2, the output would be:

```
Hash tag name: mycoolwife  
count of month 200812: 100  
count of month 200901: 201
```

[10 marks]

Total Marks:

Please note that the total mark for this assignment is capped at 100. If your marks add to more than 100 then your final mark will be capped at 100.

Return of Assignments

Departmental Policy requires that assignments be returned within three weeks of the submission date. We will endeavour to have your assignment returned before the BDC exam.