The data has the following attributes

|  |  |  |
| --- | --- | --- |
| Attribute index | Attribute name | Description |
| 0 | Name | The game name |
| 1 | Platform | Platform of the games release (Categorical: "Wii","NES","GB","DS","X360","PS3","PS2","SNES","GBA",  "3DS","PS4","N64","PS","XB","PC","2600","PSP","XOne",  "GC","WiiU","GEN","DC","PSV","SAT","SCD","WS","NG",  "TG16","3DO","GG","PCFX") |
| 2 | Year | Year of the game's release |
| 3 | Genre | Genre of the game (categorical: "Sports","Platform","Racing",  "Role-Playing","Puzzle","Misc","Shooter","Simulation",  "Action","Fighting","Adventure","Strategy") |
| 4 | Publisher | Publisher of the game (categorical: "Nintendo","Microsoft Game Studios","Take-Two Interactive","Sony Computer Entertainment","Activision","Ubisoft","Bethesda Softworks",  "Electronic Arts","Sega","SquareSoft","Atari","505 Games", …, and 567 other unique records) |
| 5 | NA\_Sales | Sales in North America (in millions) |
| 6 | EU\_Sales | Sales in Europe (in millions) |
| 7 | JP\_Sales | Sales in Japan (in millions) |
| 8 | Other\_Sales | Sales in the rest of the world (in millions) |

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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Platform** | **Year** | **Genre** | **Publisher** | **NA\_Sales** | **EU\_Sales** | **JP\_Sales** | **Other\_Sales** | |
| Gran Turismo 3: A-Spec | PS2 | 2001 | Racing | Sony Computer Entertainment | 6.85 | 5.09 | 1.87 | | 1.16 |
| Call of Duty: Modern Warfare 3 | X360 | 2011 | Shooter | Activision | 9.03 | 4.28 | 0.13 | | 1.32 |
| Pokemon Yellow: Special Pikachu Edition | GB | 1998 | Role-Playing | Nintendo | 5.89 | 5.04 | 3.12 | | 0.59 |
| Call of Duty: Black Ops | X360 | 2010 | Shooter | Activision | 9.67 | 3.73 | 0.11 | | 1.13 |
| Pokemon HeartGold/Pokemon SoulSilver | DS | 2009 | Action | Nintendo | 4.4 | 2.77 | 3.96 | | 0.77 |
| High Heat Major League Baseball 2003 | PS2 | 2002 | Sports | 3DO | 0.18 | 0.14 | 0 | | 0.05 |
| Panzer Dragoon | SAT | 1995 | Shooter | Sega | 0 | 0 | 0.37 | | 0 |
| Corvette | GBA | 2003 | Racing | TDK Mediactive | 0.2 | 0.07 | 0 | | 0.01 |
| Resident Evil: Revelations | PS3 | 2013 | Action | Capcom | 0.14 | 0.32 | 0.22 | | 0.12 |
| Mission: Impossible - Operation Surma | PS2 | 2003 | Platform | Atari | 0.14 | 0.11 | 0 | | 0.04 |
| Suikoden Tierkreis | DS | 2008 | Role-Playing | Konami Digital Entertainment | 0.09 | 0.01 | 0.15 | | 0.01 |
| Dragon Ball GT: Final Bout | PS | 1997 | Fighting | Namco Bandai Games | 0.02 | 0.02 | 0.22 | | 0.02 |
| Harry Potter and the Sorcerer's Stone | PS2 | 2003 | Action | Electronic Arts | 0.14 | 0.11 | 0 | | 0.04 |
| Tropico 4 | PC | 2011 | Strategy | Kalypso Media | 0.1 | 0.13 | 0 | | 0.04 |
| Call of Juarez: The Cartel | X360 | 2011 | Shooter | Ubisoft | 0.14 | 0.11 | 0 | | 0.03 |
| Prince of Persia: The Two Thrones | GC | 2005 | Action | Ubisoft | 0.11 | 0.03 | 0 | | 0 |
| NHL 07 | PSP | 2006 | Sports | Electronic Arts | 0.13 | 0.02 | 0 | | 0.02 |
| Disney's Winnie the Pooh's Rumbly Tumbly Adventure | GBA | 2005 | Platform | Ubisoft | 0.09 | 0.03 | 0 | | 0 |
| Batman & Robin | PS | 1998 | Action | Acclaim Entertainment | 0.06 | 0.04 | 0 | | 0.01 |
| Spider-Man: Battle for New York | DS | 2006 | Platform | Activision | 0.12 | 0 | 0 | | 0.01 |

1. [Hive] Report the top 3 games for the 'X360' console in North America based on their sales in descending order. Write the results to “Task\_1a-out”. For the above small example data set you would report the following (Name, Genre, NA\_Sales):

Call of Duty: Black Ops Shooter 9.67

Call of Duty: Modern Warfare 3 Shooter 9.03

Call of Juarez: The Cartel Shooter 0.14

[6 marks]

1. [Hive] Report the number of games for each pair of platform and genre, in descending order of count. Write the results to “Task\_1b-out”. Hint you can group data by more than one column. For the small example data set you would report the following (Platform, Genre, count):

X360 Shooter 3

PS Action 1

SAT Shooter 1

PSP Sports 1

PS3 Action 1

PS2 Sports 1

PS2 Racing 1

PS2 Platform 1

PS2 Action 1

PS Fighting 1

PC Strategy 1

GC Action 1

GBA Racing 1

GBA Platform 1

GB Role-Playing 1

DS Role-Playing 1

DS Platform 1

DS Action 1

1. [Spark RDD] Find the highest and lowest selling genre based on global sale. Print the result to the terminal output using *println*. For the small example data set you should get the following results:

(**Formula**:) [Global Sales = NA\_Sales + EU\_Sales + JP\_Sales]

Highest selling Genre: Shooter Global Sale (in millions): 27.57

Lowest selling Genre: Strategy Global Sale (in millions): 0.23

1. [Spark RDD] Report the top 50 publishers by market share (percentage of games made by the publisher among all games), along with the total number of games they made. Output should be in descending order of market share. Print the result to the terminal output using *println*. Hint: The Spark RDD method *take* might be useful. For the small example data set you would report the following (publisher name, number of games made by the published, percentage of games made by the publisher among all games):

(Ubisoft,3,15.0)

(Activision,3,15.0)

(Electronic Arts,2,10.0)

(Nintendo,2,10.0)

(Acclaim Entertainment,1,5.0)

(Sega,1,5.0)

(3DO,1,5.0)

(Namco Bandai Games,1,5.0)

(TDK Mediactive,1,5.0)

(Sony Computer Entertainment,1,5.0)

(Konami Digital Entertainment,1,5.0)

(Kalypso Media,1,5.0)

(Capcom,1,5.0)

(Atari,1,5.0)

Task 2: Analysing Twitter Time Series Data [24 marks]

In this task we will be doing some analytics on real Twitter data[[1]](#footnote-1). The data is stored in a tab (“\t”) delimited format.

The data is supplied with the assignment in the zip file and on HDFS at the below locations:

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 |

1. [Hive] Find the top 5 months with the highest number of cumulative tweets and sort it according to the number of tweets of each month. Write the results to “Task\_2a-out”. The month with the highest number of cumulative tweets should come first. So, for the above small example dataset the result would be:

month numtweets

200907 1000

200912 590

200901 201

200812 100

200905 23

1. [Spark RDD] Find the year with the maximum number of cumulative tweets in the entire dataset across all months. Report the total number of tweets for that year. Print the result to the terminal output using println. For the above small example dataset the output would be:

2009 1819

1. [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. We have already written code in your code template that reads the x and y values from the keyboard. 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. Print the result to the terminal output using println. For the above small example data set the output should be the following:

**Input** x = 200910, y = 200912

**Output** hashtagName: mycoolwife, countX: 1, countY: 500

Task 3: Indexing Bag of Words data [25 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 in the zip file and on HDFS at the below locations:

**Local filesystem:**

|  |
| --- |
| **Small version** |
| t3/Input\_data/docword-small.txt |
| t3/Input\_data/vocab-small.txt |
| **Full version** |
| t3/Input\_data/docword.txt |
| t3/Input\_data/vocab.txt |

**HDFS:**

|  |
| --- |
| **Small version** |
| hdfs:///user/ashhall1616/bdc\_data/assignment/t3/docword-small.txt |
| hdfs:///user/ashhall1616/bdc\_data/assignment/t3/vocab-small.txt |
| **Full version** |
| hdfs:///user/ashhall1616/bdc\_data/assignment/t3/docword.txt |
| hdfs:///user/ashhall1616/bdc\_data/assignment/t3/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:

1. [spark SQL] Calculate the total count of each word across all documents. List the words ordered by count in descending order. Write the results in csv format at “file:///home/**USERNAME**/Task\_3a-out”, replacing **USERNAME** with your **CloudxLab** username. Use *show()* to print the first 10 rows of the dataframe that you saved. So, for the above small example input the output would be the following:

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

| word|sum(count)|

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

|motorbike| 2502|

| boat| 2200|

| plane| 1100|

| car| 620|

| 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.

1. [spark SQL]
   1. 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. Write the results in parquet format **partitioned by firstLetter** at “file:///home/**USERNAME**/t3\_docword\_index\_part.parquet”, replacing **USERNAME** with your **CloudxLab** username. Use *show()* to print the first 10 rows of the partitioned 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|

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

1. [spark SQL] Load the dataframe stored in partitioned parquet format from subtask b). The task template includes code to obtain a list of query words from the user. For each word in the queryWords list use println to display the following: the word and; the docId with the largest count for that word (you can break ties arbitrarily). Skip any query words that aren’t found in the dataset. To iterate over the query words, use a normal Scala for loop, like this:

for(queryWord <- queryWords) {

// ...put Spark query code here...

}

For this subtask there is a big optimisation that can be made because of how the data is partitioned. So, think carefully about how you filter the data.

If queryWords contains “car”, “dog”, and “truck”, then the output for the example dataset would be:

[car,2]

[truck,3]

1. [spark SQL] Again load the dataframe stored in partitioned parquet format from subtask b). The task template includes code to obtain a list of document IDs from the user. For each document ID in the docIds list use println to display the following: the document ID and; the word with the most occurrences in that document, along with the number of occurrences of that word in the document (you can break ties arbitrarily). Skip any document IDs that aren’t found in the dataset. Note in this task we are searching via document ID instead of query word so we cannot take advantage of the same optimization you used for part c). So, you need to find some other kind of optimization. Hint: think about the fact you are repeatedly reusing the same dataframe in the scala for loop as you process each docId.

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

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

Task 4: Finding associations between news publishers [22 marks]

*Note: Due to the size of the full dataset, you may need to run tasks using this command (the command below allocates 1GB of RAM to the spark executor rather than the default 512MB):*

spark-shell -i Task\_XX.scala –-driver-memory 1g

The data has the following attributes

|  |  |  |
| --- | --- | --- |
| Attribute index | Attribute name | Description |
| 0 | articleId | Numeric identifier for article |
| 1 | title | Title of article |
| 2 | url | Url of article |
| 3 | publisher | The publisher of the article |
| 4 | category | News category (b = business, t = science and technology, e = entertainment, m = health) |
| 5 | storyId | Alphanumeric id of “story”. All articles are first grouped via a clustering algorithm. Then each cluster is assigned a story id. The story id identifies the topic of the story, not the specific article. Therefore, there are multiple articles (rows of data) per story. |
| 6 | hostname | URL hostname |
| 7 | timestamp | Approximate time the news was published in unix time |

Here is a small example of the article 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):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| articleId | publisher | Category | storyId | hostname |
| 1 | Los Angeles Times | B | ddUyU0VZz0BRneMioxUPQVP6sIxvM | www.latimes.com |
| 2 | Livemint | B | ddUyU0VZz0BRneMioxUPQVP6sIxvM | www.livemint.com |
| 3 | IFA Magazine | B | ddUyU0VZz0BRneMioxUPQVP6sIxvM | www.ifamagazine.com |
| 4 | IFA Magazine | B | ddUyU0VZz0BRneMioxUPQVP6sIxvM | www.ifamagazine.com |
| 5 | Moneynews | B | ddUyU0VZz0BRneMioxUPQVP6sIxvM | www.moneynews.com |
| 6 | NASDAQ | B | ddUyU0VZz0BRneMioxUPQVP6sIxvM | www.nasdaq.com |
| 16 | NASDAQ | B | dPhGU51DcrolUIMxbRm0InaHGA2XM | www.nasdaq.com |
| 19 | IFA Magazine | B | dPhGU51DcrolUIMxbRm0InaHGA2XM | www.ifamagazine.com |

Complete the following subtasks using Spark:

1. [Spark SQL]
   1. Create a list of each story paired with each publisher that wrote at least one article for that story. Store the resulting dataframe in Parquet format at “file:///home/**USERNAME**/t4\_story\_publishers.parquet”, replacing **USERNAME** with your **CloudxLab** username. For the small example input file the expected output is:

[ddUyU0VZz0BRneMioxUPQVP6sIxvM, Livemint]

[ddUyU0VZz0BRneMioxUPQVP6sIxvM, IFA Magazine]

[ddUyU0VZz0BRneMioxUPQVP6sIxvM, Moneynews]

[ddUyU0VZz0BRneMioxUPQVP6sIxvM, NASDAQ]

[dPhGU51DcrolUIMxbRm0InaHGA2XM, IFA Magazine]

[ddUyU0VZz0BRneMioxUPQVP6sIxvM, Los Angeles Times]

[dPhGU51DcrolUIMxbRm0InaHGA2XM, NASDAQ]

* 1. From this dataframe, find the top 10 most popular stories (measured by number of articles) and the number of publishers that wrote an article about them, report the results using *println*. A story is considered popular if at least 5 publishers published at least one article on it. For the small example input file the expected output is:

[ddUyU0VZz0BRneMioxUPQVP6sIxvM,5]

1. [Spark SQL] Load up the Parquet file which you created in the previous subtask. Find all pairs of publishers that publish articles about the same stories. For each publisher pair report the number of co-published stories. Where a co-published story in a story published by both publishers. Report the pairs in decreasing order of frequency. The solution may take a few minutes to run. Note the solution must conform to the following rules:
   1. There should not be any replicated entries like:

NASDAQ, NASDAQ, 1000

* 1. You should not have the same pair occurring twice in opposite order. Only one of the following should occur:

NASDAQ, Reuters, 1000  
Reuters, NASDAQ, 1000

(i.e. it is **incorrect** to have **both** of the above two lines in your result)

Save the results in CSV format at “file:///home/**USERNAME**/t4\_paired\_publishers.csv”, again replacing **USERNAME** with your **CloudxLab** username. For the example above, the output should be as follows (it is OK if the csv is split over multiple files, but the combined data contents of the multiple files should be the same, with the order preserved):

[NASDAQ,IFA Magazine,2]

[Moneynews,Livemint,1]

[Moneynews,IFA Magazine,1]

[NASDAQ,Livemint,1]

[NASDAQ,Los Angeles Times,1]

[Moneynews,Los Angeles Times,1]

[Los Angeles Times,IFA Magazine,1]

[Livemint,IFA Magazine,1]

[NASDAQ,Moneynews,1]

[Los Angeles Times,Livemint,1]

Bonus:

1. Using spark, perform the following task using the Twitter dataset from 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:

Hashtag name: mycoolwife

count of month 200812: 100

count of month 200901: 201

1. : Twitter data source: <http://www.infochimps.com/datasets/twitter-census-conversation-metrics-one-year-of-urls-hashtags-sm> [↑](#footnote-ref-1)