DALHOUSIE UNIVERSITY

Data Management, Warehousing and Analytics

Jaskaran Singh

MACS

B00948857

js356337@dal.ca

Assignment 3

Gitlab Repository link:

https://git.cs.dal.ca/singh16/csci5408_s23_b00948857_jaskaran_singh.git

PROBLEM 1

PROBLEM 1A:

- "ReutRead.java," is designed to read the contents of the given news files, extract the relevant information between the specified <REUTERS></REUTERS>, <TEXT></TEXT>, and <TITLE></TITLE> tags, and then store this information as documents in a MongoDB database named "ReuterDb." Each document in the database contains the news article's title, body (text), and dateline.
- The code uses regular expressions (regex) to extract the required information from the news files. The Pattern and Matcher classes from the java.util.regex package are used to perform the matching and extraction.
- The code for this part is present in git folder in the “Problemone” folder in “ReutRead.java” file. Maven project has been created for problem 1.
- Here are the steps explaining all the code:

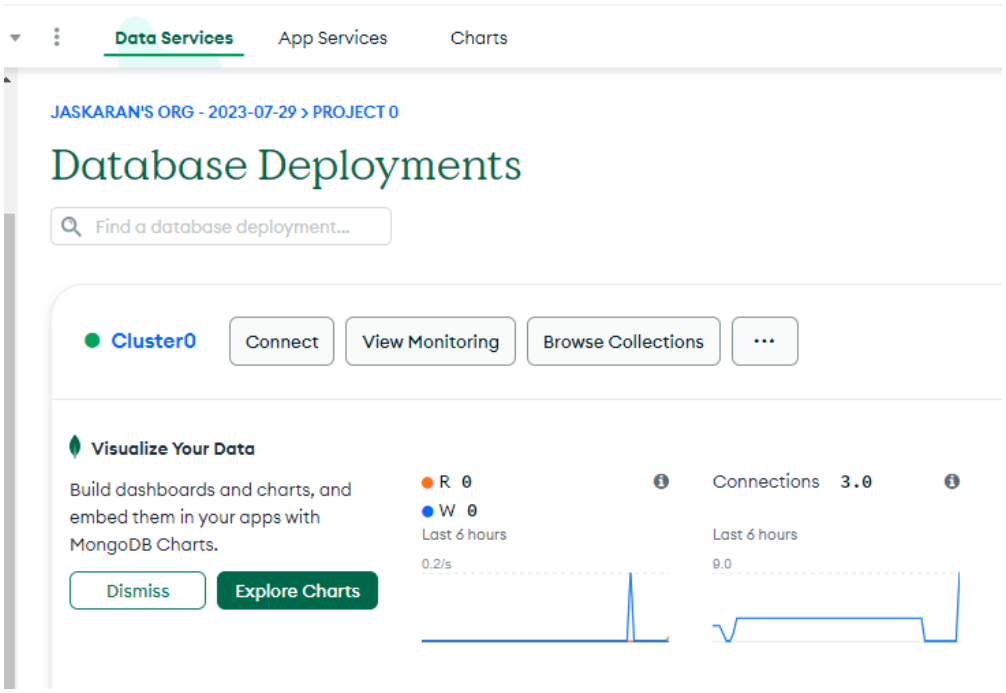


Figure 1: Create a cluster on Mongo Atlas

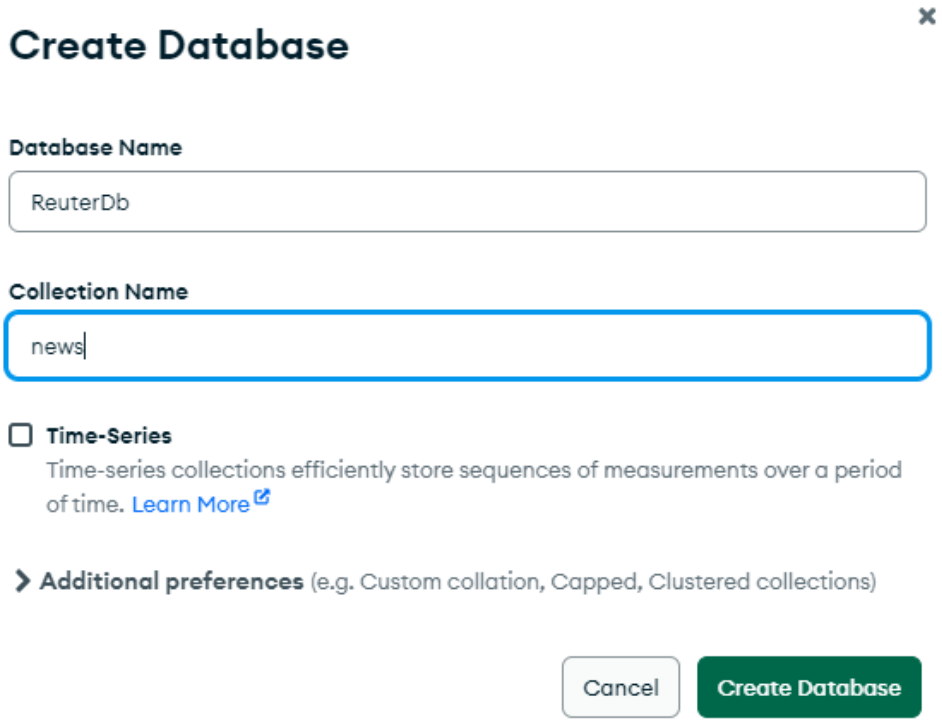


Figure 2: Database and Collection creation in MongoDB after building connection using connection string.

Algorithm:

1. The MongoDB client (Compass) is set up using the provided connection information, such as username, password, cluster name, and database name.

2. The method buildConnection() establishes a connection to the MongoDB database and returns the collection named "news," where the news articles will be stored.

3. The createDocumentinCollection() method takes the title, body, and dateline as input and creates a MongoDB document containing this information. It then inserts the document into the "news" collection in the MongoDB database.

4. The main method reads the contents of the news files specified by file1 and file2. It then uses regular expressions to find each <REUTERS> tag and extracts the title, body, and dateline information using separate regex patterns.

5. For each <REUTERS> tag found, a document is created using the extracted information, and it is inserted into the MongoDB collection using the createDocumentinCollection() method.

6. After processing all the files, the MongoDB connection is closed.

The code efficiently extracts the desired information from the news files and stores it in the MongoDB database.

Code:

Methods:

1. In ‘static’ block, connection is built with Mongo Atlas.
2. ‘buildConnection()’ method returns ‘news’collection.
3. ‘closeConnection()’ method closes connection with database.
4. ‘createDocumentinCollection()’ creates a document in ‘news’ collection.
5. ‘main’ method fetches data from input files using regex expressions and then calls method to successfully save data in collection.

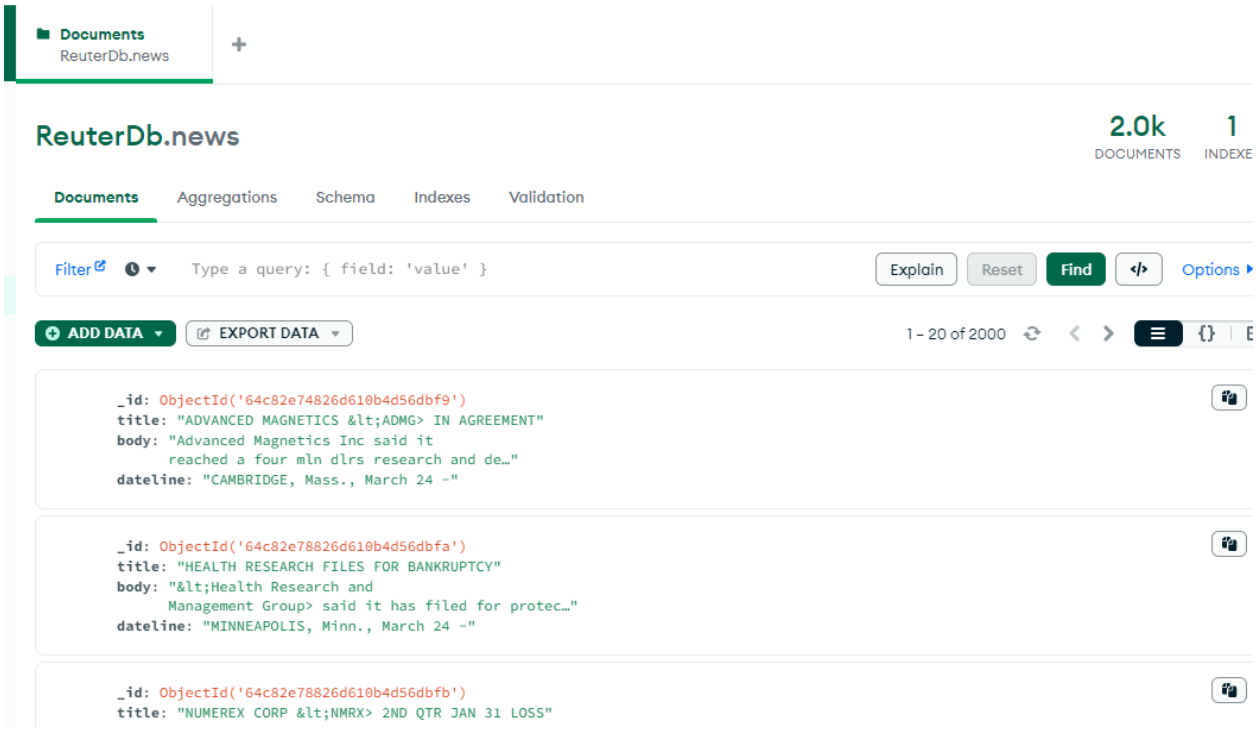


Figure 3: Documents inserted in MongoDB database

FlowChart:

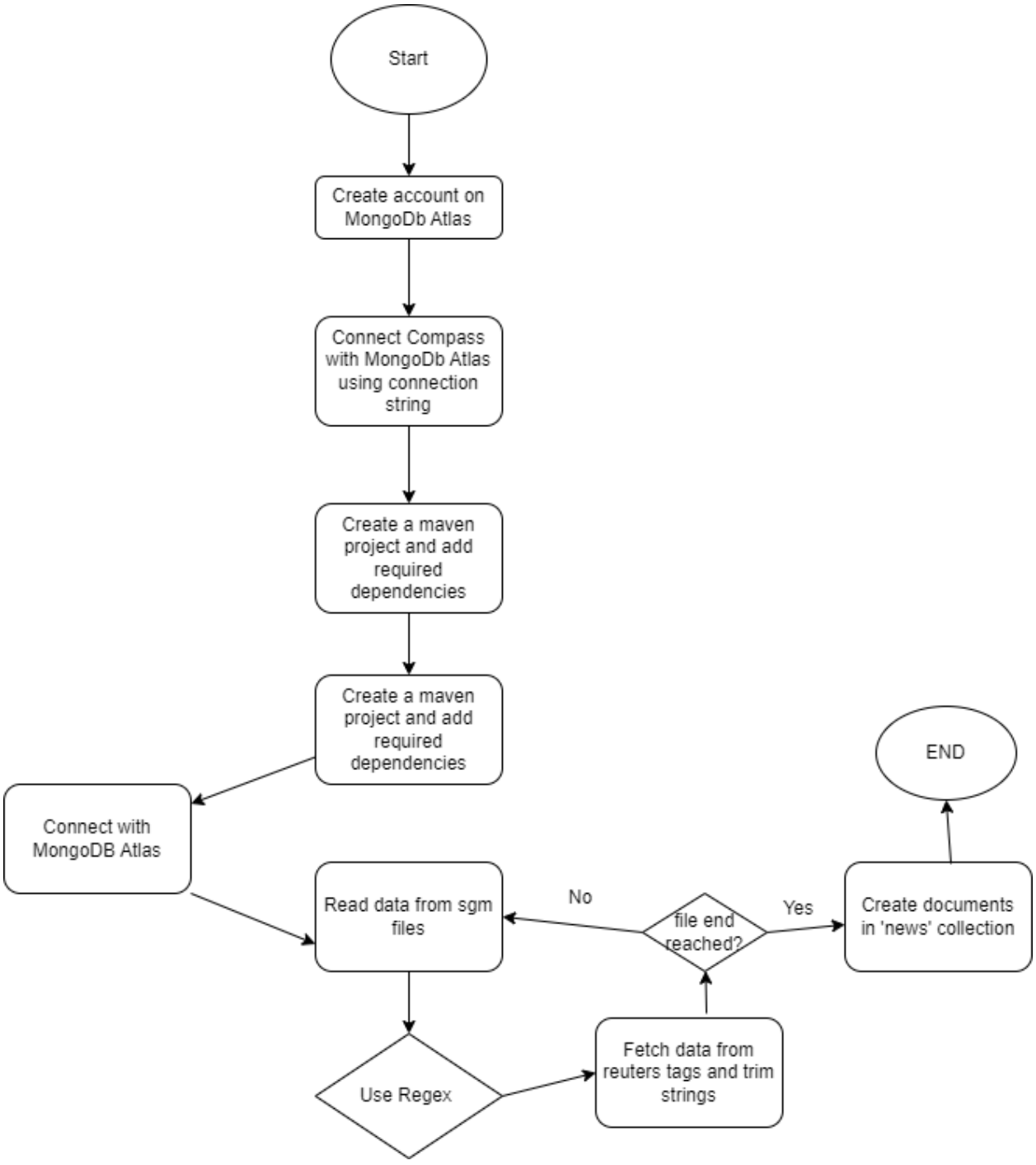


Figure 4: FlowChart for Reuters data transformation

PROBLEM 1B:

- Create a cluster using Dataproc on GCP using following steps:
    1. Sign in into your GCP account.
    2. Create a project in GCP.
    3. Now search for DataProc in search bar.

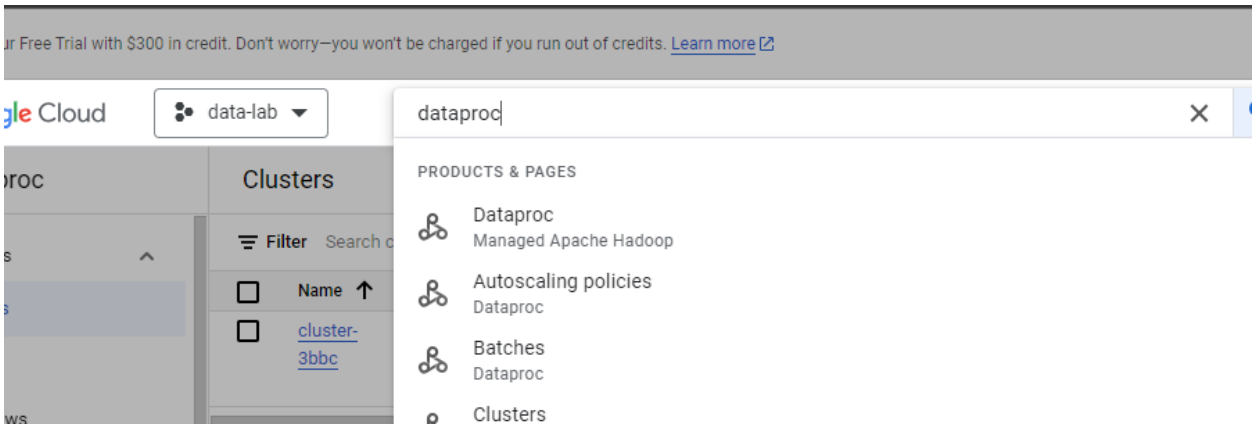


Figure 5: Steps for creating cluster on GCP.

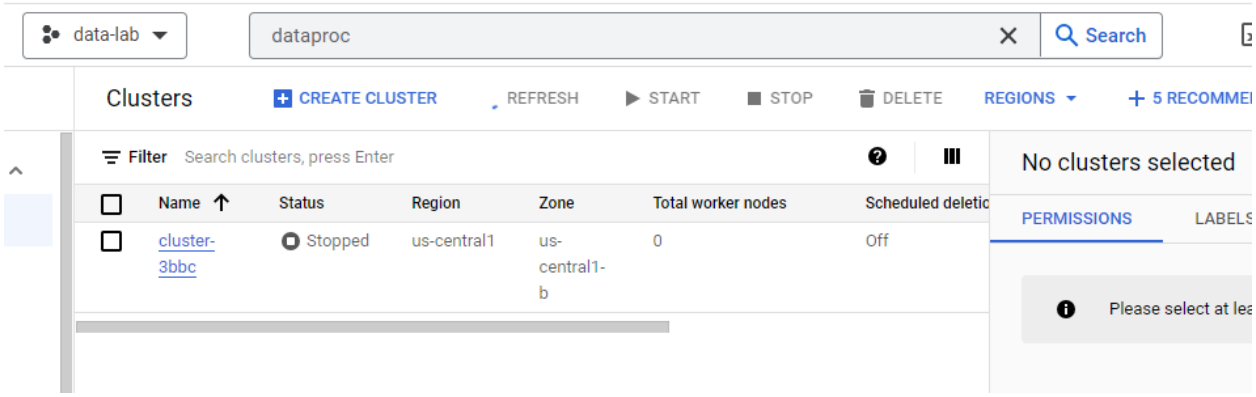


Figure 6: Cluster details

4. Now configure the cluster and start the instance.
5. After instance is loaded, go to VM instances and click SSH to create connection with cluster.

- Now, in the maven project created for problem 1A, create a new Java class 'WordCounter' to create a MapReduce program.
- Add dependencies 'spark-core_2.12' and 'spark-sql_2.12' in pom.xml of your maven project.

- The "WordCounter.java" program reads a text file ("reut2-009.sgm") using Apache Spark, counts the occurrences of each word, and then prints the most and least occurred words along with their respective occurrence counts. Additionally, it saves the sorted word counts to an output file named "output.txt."
- Code is present on GIT in Projectone folder in WordCounter.java file.

- **Explanation:**
    1. Setting up Spark Session and JavaSparkContext:
    2. The code starts by setting up a SparkSession using the SparkSession.builder(). The SparkSession is the entry point to any Spark functionality, and it allows you to interact with Spark and create DataFrames or RDDs. A JavaSparkContext is also created, which is a Java-friendly interface to the Spark cluster.
    3. **Reading the Text File:** The code reads the contents of the "reut2-009.sgm" file using the textFile() method, which returns an RDD (Resilient Distributed Dataset) of strings, where each string represents a line from the file.
    4. **Removing XML Tags:** The textContents RDD is created by mapping each line and removing any XML tags present in the text. This is done using a regular expression (replaceAll("<.*?>", "")) to match and replace all occurrences of XML tags with empty strings, effectively removing them.
    5. **Word Tokenization and Cleaning:** The RDD "words" is created by flat-mapping each line and splitting it into individual words using the regular expression line.split("\\W+"), which splits the line based on non-word characters. The words are then cleaned by converting them to lowercase and removing any non-alphanumeric characters using a regular expression (replaceAll("[^a-zA-Z0-9]", "")). Words with a length greater than 0 are filtered to remove any empty strings.
    6. **Counting Word Occurrences:** The RDD "cleanedWords" contains cleaned words, and countByValue() is used to count the occurrences of each unique word. The result is stored in the wordCounts map, where the word is the key, and the occurrence count is the value.
    7. **Sorting the Word Counts:** The wordCounts map is converted to a List of Map entries, sorted based on the occurrence count in descending order using the Stream API and sorted() method.
    8. **Printing and Saving the Results:** The most and least occurred words along with their respective occurrence counts are printed to the console. If there are words found, the sorted word counts are saved to the "output.txt" file using a BufferedWriter.
    9. **Stopping Spark:**
    10. Finally, the SparkSession is stopped to release the resources.

- **Algorithm for MapReduce method:**
    1. Create a SparkSession and JavaSparkContext.
    2. Read the contents of the "reut2-009.sgm" file into an RDD called textLines.

3. Remove XML tags from each line using a regular expression and create an RDD called textContents.
4. Tokenize the lines into individual words, clean the words, and create an RDD called cleanedWords.
5. Count the occurrences of each word in the cleanedWords RDD and store the result in a map called wordCounts.
6. Convert the wordCounts map to a List of Map entries and sort it based on the occurrence count in descending order, resulting in a list called sortedWords.
7. Print the size of the sortedWords list.
8. If sortedWords is not empty, print the most and least occurred words along with their occurrence counts. Save the sorted word counts to the "output.txt" file.
9. Stop the SparkSession to release resources.
10. The code performs word counting and sorting using Apache Spark's distributed computing capabilities, which makes it scalable for handling large datasets. The output file "output.txt" will contain the words and their respective occurrence counts, sorted in descending order based on occurrence.

- Now after creating the MapReduce program, package this program into jar file using the IntelliJ project life cycle option in Maven.
- Now go to the GCP account, open the shell in VM instances by clicking ssh.
- After that, upload your jar files and input file ‘`reut2-009.sgm`’ on shell.

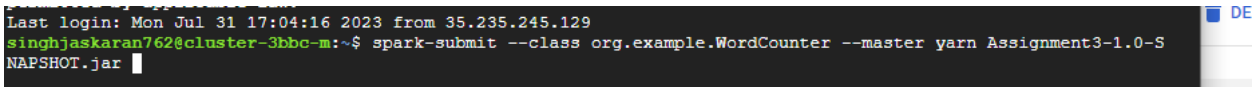


Figure 7: Spark submit command

- Now run command ‘spark submit’ on shell as shown in the figure.
- There will be an ‘output.txt’ file created after successful completion of the program.

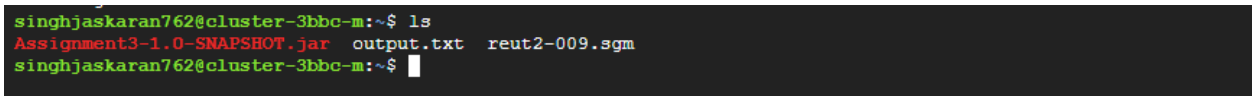


Figure 7: ls command after successful run of MapReduce program

- This file contains occurrences of all the words and their counts in the descending order.
- Download the file. Also, The file is uploaded to Gitlab folder.

Figure 8: Head of output.txt file showing words with maximum occurrence.

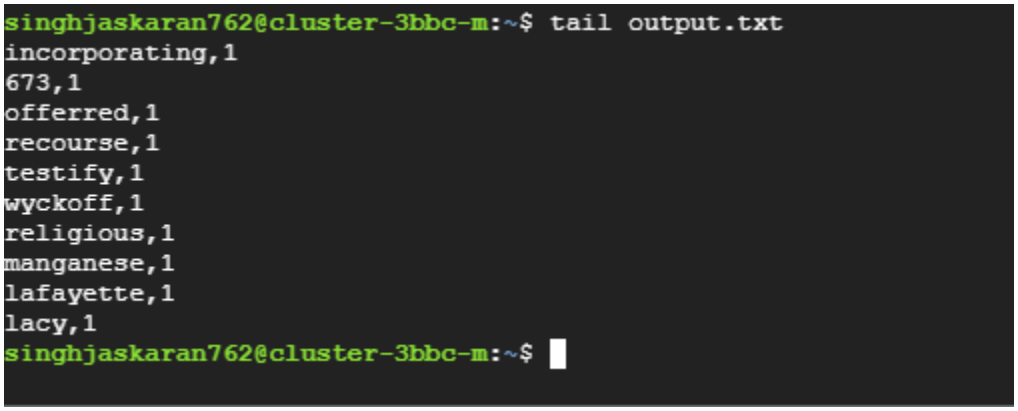


Figure 9: Tail of output.txt file showing words with minimum occurrence.

PROBLEM 2

- Code is present in the BOW folder.
- Project is managed using maven.
- Three java files are created for this problem.
- The provided code consists of three classes:
    1. ReutRead
    2. Main
    3. BOWModel.
- It aims to analyze news titles and classify them as positive, negative, or neutral based on the presence of positive and negative words. The code uses Bag-of-Words (BOW) modeling to represent each news title and calculates the polarity score based on the occurrence of positive and negative words in the titles.

- **Explanation:**

  <u>ReutRead Class:</u>

- This class is responsible for reading news titles from the files "reut2-009.sgm" and "reut2-014.sgm."
- The method readTitleInFiles() reads the contents of the specified files, extracts the news titles using regular expressions, and stores them in a list called titleList.
- The methods getPositiveWordList() and getNegativeWordList() read positive and negative words from the files "positive-words.txt" and "negative-words.txt," respectively. These lists are filtered to remove any empty strings and returned.
- The method getBOWModels() performs BOW modeling on each title in the titleList. It creates a list of BOWModel objects, each representing a news title with its word count map.

  <u>Main Class:</u>

- This class contains the main method and is responsible for the execution of the sentiment analysis process.
- It first calls the getBOWModels() method from the ReutRead class to obtain a list of BOWModel objects representing the news titles and their word count maps.
- Then, it retrieves the positive and negative word lists using the methods getPositiveWordList() and getNegativeWordList() from the ReutRead class.
- For each BOWModel, it calculates the polarity score using the calculatePolarity() method from the BOWModel class. The polarity score is determined based on the occurrence of positive and negative words in the title. A positive polarity score indicates positive sentiment, a negative score indicates negative sentiment, and a score of zero indicates neutral sentiment.
- Finally, the code prints the title, classification (positive, negative, or neutral), and matched words (if any) for each news title.

BOWModel Class:

- This class represents the Bag-of-Words (BOW) model for a news title.
- It has fields to store the title itself, a list of names (words) in the title, and a map (cnt) that holds the word counts for each word in the title.
- The method calculatePolarity() calculates the polarity score for the news title based on the occurrence of positive and negative words. It takes the positive and negative word lists as input and iterates through the word counts to determine the polarity score. It also prints the matched words for each news title.
- There are getter and setter methods for the class fields.
- Overall, this code performs sentiment analysis on news titles using a BOW modeling approach. It uses positive and negative word lists to classify the news titles as positive, negative, or neutral based on the presence and frequency of positive and negative words in the titles. The matched words are also printed for each title to provide insights into the classification.

2A

- `ReutRead.getBOWModels()`
- This method is created to create List of BOWModels objects which contain title, words in title of each news and occurrence of each word.

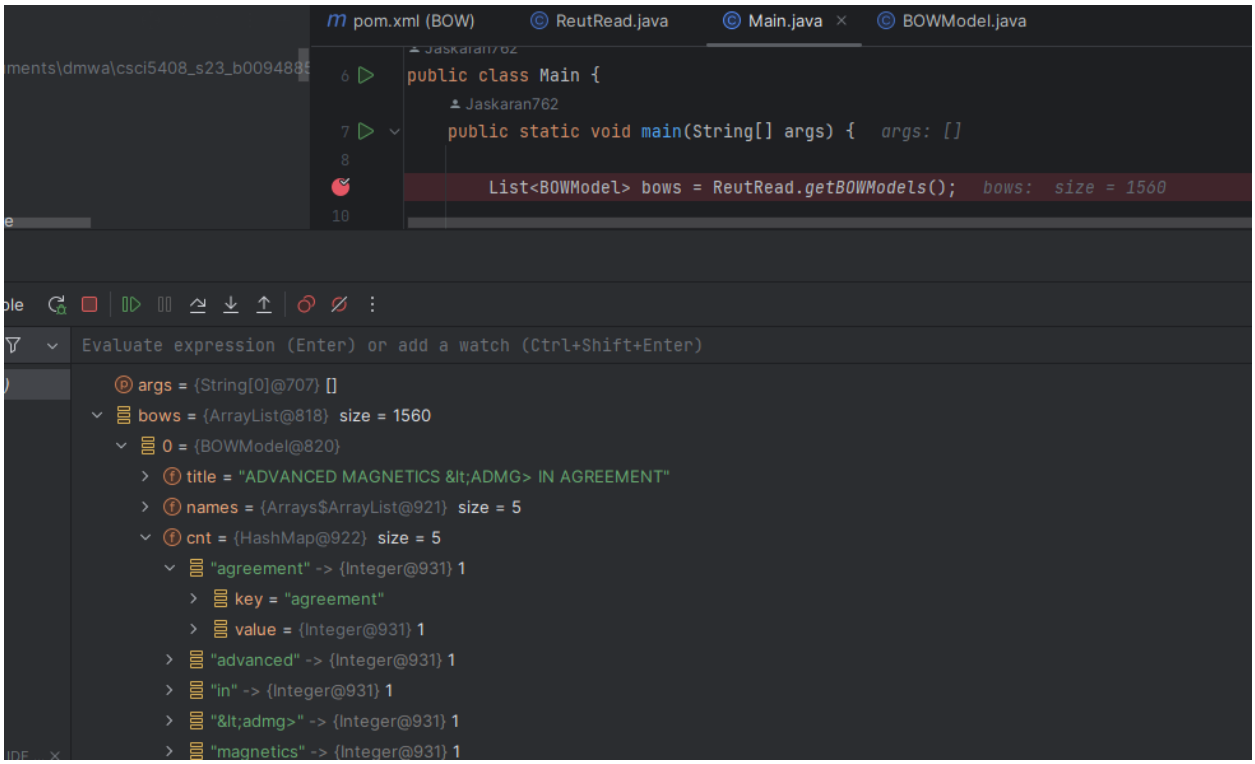


Figure 10: Bows containing list of BOW showing details of each news

- In the above figure, each object in ‘bows’, contains details like title, words in title (in names) and cnt variable showing count of each word of title of news.

2B and 2C:

```
List<String> positiveWordListWithNull = ReutRead.getPositiveWordList();
List<String> positiveWordList = positiveWordListWithNull.stream()
        .filter(s -> !s.isEmpty())
        .toList();
List<String> negativeWordListWithNull = ReutRead.getNegativeWordList();
List<String> negativeWordList = negativeWordListWithNull.stream()
        .filter(s -> !s.isEmpty())
        .toList();
```

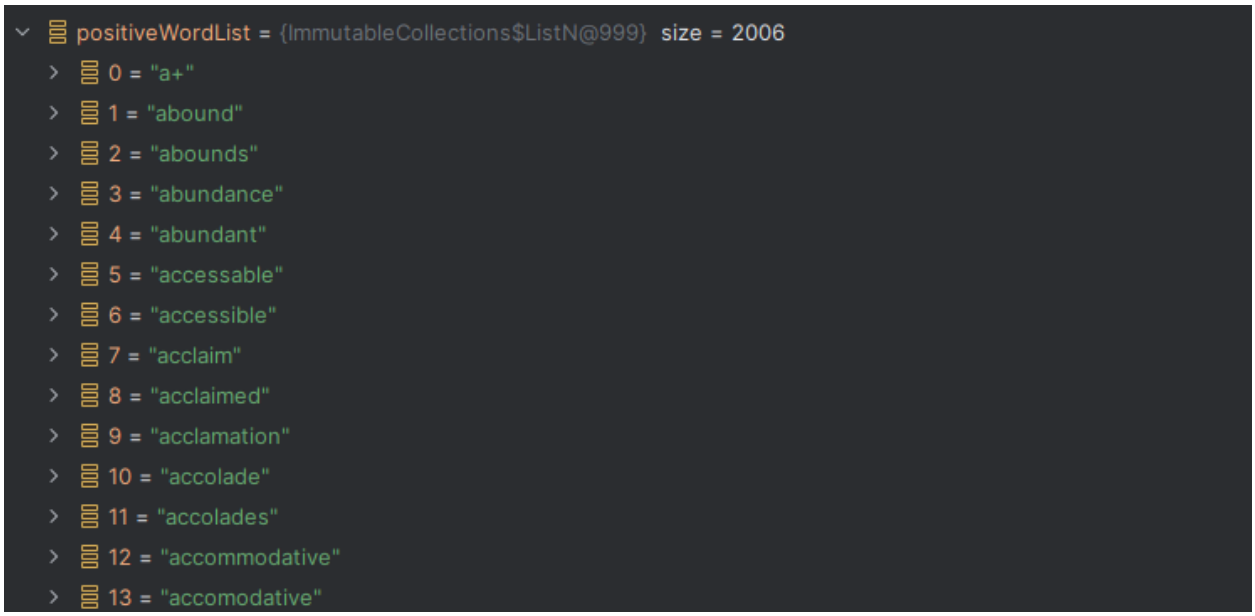
- ‘positiveWordList’ and ‘negativeWordList’ contains all the words fetched from the github repo text files.
- 

Figure 11: positiveWordList containing all the positive words fetched from text file.

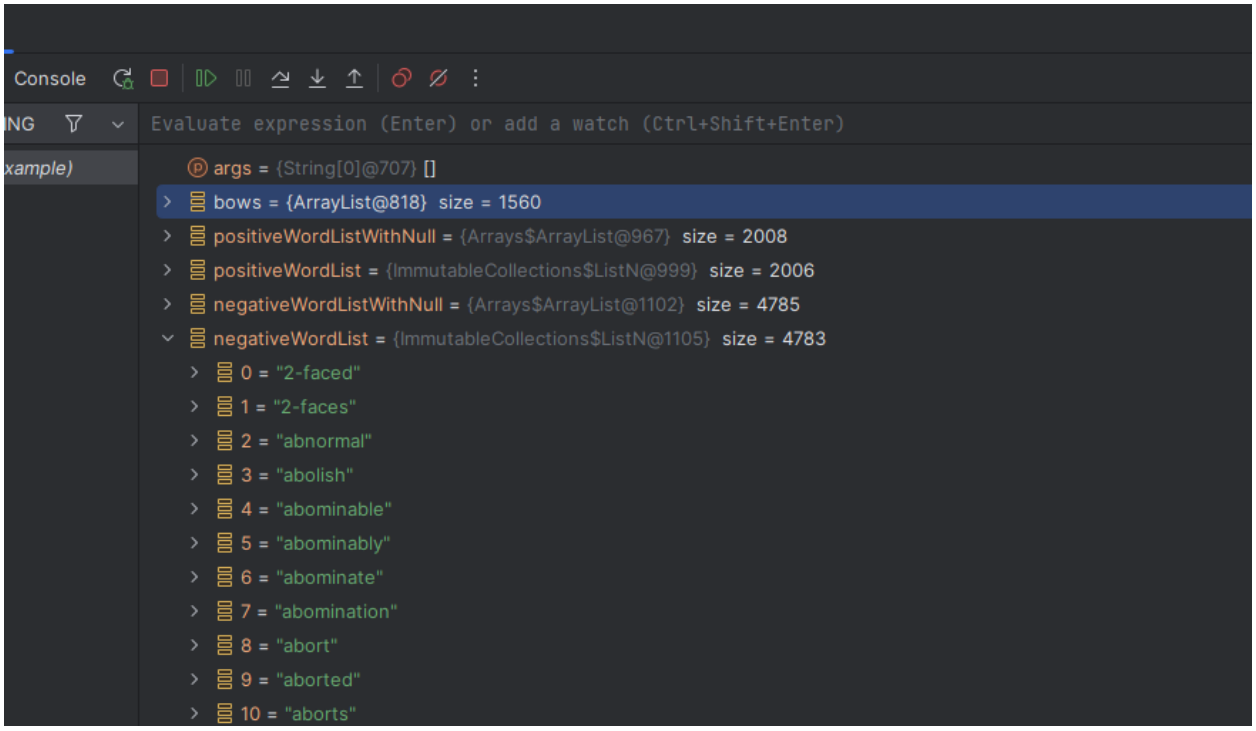


Figure 12: negativeWordList containing all the negative words fetched from text file.

Figure 13: calculatePolarity method calculating polarity pf each title.

```
Matched Words: advanced
1 ||  ADVANCED MAGNETICS &lt;ADMG> IN AGREEMENT  || Positive News  || 1

No matched Words
2 ||  HEALTH RESEARCH FILES FOR BANKRUPTCY  || Neutral News  || 0

Matched Words: loss
3 ||  NUMEREX CORP &lt;NMRX> 2ND QTR JAN 31 LOSS  || Negative News  || -1

No matched Words
4 ||  COMMODORE &lt;CBU>, ATARI IN SETTLEMENT  || Neutral News  || 0

Matched Words: supports
5 ||  BALDRIGE SUPPORTS NIC TALKS ON CURRENCIES  || Positive News  || 1

No matched Words
6 ||  TRIANGLE &lt;TRI> BEGINS EXCHANGE OFFER  || Neutral News  || 0
```

```
No matched Words
1555 ||  AMERTEK INC &lt;ATEKF> 1ST QTR NET  || Neutral News  || 0

No matched Words
1556 ||  COMSTOCK GROUP &lt;CSTK> SELLS PREFERRED STOCK  || Neutral News  || 0

No matched Words
1557 ||  QVC NETWORK &lt;QVCN> CLARIFIES AGREEMENT  || Neutral News  || 0

No matched Words
1558 ||  ALEX BROWN INC &lt;ABSB> 1ST QTR MARCH 27 NET  || Neutral News  || 0

Matched Words: equitable
1559 ||  EQUITABLE RESOURCES &lt;EQT> FILES UNIT OFFERING  || Positive News  || 1

No matched Words
1560 ||  TOWN AND COUNTRY JEWELRY MANUFACTURING &lt;TCJC>  || Neutral News  || 0
```

Figure 14 & 15: Table showing matched words, s.no, title of each news, tag of each news (positive, negative or neutral) and score of each news

- **References:**

1. FlowChart, Draw.io

   Link: https://app.diagrams.net/

2. Docs, MongoDB

   Link: https://www.mongodb.com/docs/

3. Guide to MongoDb with Java, Baeldung

   Link: https://www.baeldung.com/java-mongodb

4. Dataproc, Google Cloud Platform

   Link:https://console.cloud.google.com/dataproc/clusters?project=data-lab-389815

5. Pattern Class, Oracle Docs

   Link: https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

6. Spark Java, JavaTPoint

   Link: https://www.javatpoint.com/spark-java#:~:text=What%20is%20Spark%20Java%3F,was%20the%20inspiration%20for%20it.