DSE 230: Programming Assignment 4.2 - Word Count on Amazon EMR

Tasks:

- Work with **BookReviews_1M** dataset to set up the word count exercise as you did in PA2.
  - This is only to get you started on AWS and you won't need to report anything on this dataset except the run-time of your code. The word counts that you got from your previous assignment would (and should) be the same as the result you get from running the same experiment on AWS.
- Find top 100 words and their counts based on word count for the **BookReviews_5M** dataset
- Calculate average and standard deviation of execution times over 3 runs for these three settings:

  1. BookReviews_1M - 1 master + 1 worker node
  2. BoookReviews_5M - 1 master + 1 worker node
  3. BookReviews_5M - 1 master + 3 worker nodes

  Note that worker nodes are also called core nodes when initializing them on AWS.

- Submission on Gradescope (4 files) under **PA4.2**

  - Completed PySpark notebook (.ipynb) with results for 5M reviews dataset
    - Make sure that all expected outputs are present
  - A PDF of this Jupyter Notebook
    - If exporting fails, download as HTML and print to PDF
  - CSV file of first 100 rows of results for 5M reviews dataset
  - A screenshot (.png) of your AWS EMR Cluster page showing that all clusters have been terminated.
  - NOTE - You do NOT have to submit an exported executable (.py) unlike other PAs

Due date: Friday 5/28/2021 at 11:59 PM PST

Remember: when in doubt, read the documentation first. It's always helpful to search for the class that you're trying to work with, e.g. pyspark.sql.DataFrame.

PySpark API Documentation: https://spark.apache.org/docs/latest/api/python/index.html

1. Upload the 1M dataset to S3

To make the datasets available to the EMR cluster, we need to upload the data files to Amazon S3. Follow these steps to do so:

1. In the Amazon console, open the **Services** menu on the top left and select **S3**
2. Create a bucket if you don't have one yet. Use the default settings, but your bucket name must be unique.
3. Create a folder in your bucket, e.g. `data`, using the default settings. (Don't upload the data file to the root of the bucket; we'll also use this bucket for later assignments, so it's good to keep everything organized.)
4. Enter the folder and upload the **.txt** file. Do NOT upload the zip, as Spark won't know what to do with it.

You can use this dataset now. The next steps are for setting up an EMR cluster. After setting up the cluster, create a **PySpark** notebook and read the file you uploaded by copying the S3 URI, convert it to a dataframe and do anything else you want.

This exercise is only to help you understand how you can create your own S3 buckets and read data from it. The actual task for you is to read data (BookReviews_5M.txt) from a different S3 bucket and work on that dataset.

2. Setting up the EMR cluster and creating a PySpark notebook

We have already uploaded the 5M reviews data to the s3 bucket `s3://dse230-emr`. Follow the steps below to create an EMR cluster:

1. In AWS, go to Services -> EMR.
2. Click on 'Create cluster'.
3. Click on 'Go to advanced options'.
4. Select the EMR version 6.2.0, add required software packages as shown in class.
5. Specify the instance count for master and core nodes
6. Give your cluster a name, select an EC2 keypair that you should have created earlier. If you have not created an EC2 keypair, stop here. Go back and create a keypair first, then come back to this step.
7. Proceed to create a cluster and wait for completion. This will take a few minutes (typically ~5-8 minutes).
8. Go to `Notebooks` section in the sidebar of the EMR dashboard page, create a new notebook and attach it to the cluster you created earlier
9. Open JupyterLab from this page and create a **PySpark** notebook
10. The data is at `s3://dse230-emr/BookReviews_5M.txt`. In the following sections, use this URI for data file path

3. Start Spark Session

Note that yo don't need to manually start the spark session. AWS does it for you in the background, so that the spark session is started as soon as you import pyspark. The spark session is automatically available in the global variable `spark`

Remember that the kernel for running this Notebook is **PySpark** and not Python 3.

In [1]:
```
# Initialize Spark

import pyspark

print (spark.version, pyspark.version.__version__)
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
3	application_1621892030844_0004	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.

3.0.1-amzn-0 3.0.1+amzn.0
```

In [2]:
```
# Record the starting time of execution for timing this notebook

import time
start_time = time.time()
```

In [3]:
```
# Read data from HDFS or S3 - For the purposes of this assignment, you should re
# Although you can read directly from S3 theoretically.

# Provide the HDFS file path of the 5M dataset.
dataFileName = 's3://dse230-emr/BookReviews_5M.txt'
#dataFileName = 's3://dse230-notebooks-bucket/BookReviews_1M.txt'

# Read data from the above file path and convert it to a dataframe.
textDF = spark.read.text(dataFileName)
```

4. Examine the data

Your task:

1. Examine the contents of the dataframe that you've just read from file.

Expected output:

1. Print the schema of the raw dataframe, as well as its first 25 rows.

In [4]:
```
# YOUR CODE HERE to print the schema
print(textDF.schema)

# YOUR CODE HERE to print the first 25 rows of the dataframe
print(textDF.show(25, truncate = True))
```

```
StructType(List(StructField(value,StringType,true)))
+--------------------+
|               value|
+--------------------+
|This was the firs...|
|Also after going ...|
|As with all of Ms...|
|I've not read any...|
|This romance nove...|
|Carolina Garcia A...|
|Not only can she ...|
|Once again Garcia...|
|The timing is jus...|
|Engaging. Dark. R...|
|Set amid the back...|
|This novel is a d...|
|If readers are ad...|
| Reviewed by Phyllis|
|      APOOO BookClub|
|A guilty pleasure...|
|In the tradition ...|
|Beryl Unger, top ...|
|What follows is a...|
|The book flap say...|
|I'd never before ...|
|The novel's narra...|
|It is centered on...|
|If you like moder...|
|Beryl Unger is a ...|
+--------------------+
only showing top 25 rows

None
```

5. Clean the data

Your task:

1. Remove all punctuations and convert all characters to lower case.

Expected output:

1. The first 25 rows of a dataframe, with a column containing the cleaned sentences.

In [5]:

```python
# Do not change this cell.

# NOTE: Counterintuitively, column objects do NOT store any data; instead they s
#       The below function takes in a column object, and adds more expressions t
#       Once we have a column object representing the expressions we want, use D

from pyspark.sql.functions import regexp_replace, trim, col, lower
def removePunctuation(column):
    """Removes punctuation, changes to lower case, and strips leading and traili
    return trim(lower(regexp_replace(column, "[^A-Za-z0-9 ]", ""))).alias("sente
```

In [6]:

```python
# Recommended: take a look at the contents of a column object returned from remo
# No answers or outputs required for this cell.
print(removePunctuation(textDF.value))
```

```
Column<b'trim(lower(regexp_replace(value, [^A-Za-z0-9 ], ))) AS `sentence`'>
```

In [7]:

```
# execute the column expressions generated by removePunctuation() to clean the s
# After that, use the show() function to print the first 25 rows of the datafram
# Hint: you'll need the Column object returned by removePunctuations().

# YOUR CODE HERE for printing the expected output.
textDF_new = (textDF.select(removePunctuation(textDF.value)))
print(textDF_new.show(25, truncate = True))
```

```
+--------------------+
|            sentence|
+--------------------+
|this was the firs...|
|also after going ...|
|as with all of ms...|
|ive not read any ...|
|this romance nove...|
|carolina garcia a...|
|not only can she ...|
|once again garcia...|
|the timing is jus...|
|engaging dark rea...|
|set amid the back...|
|this novel is a d...|
|if readers are ad...|
| reviewed by phyllis|
|      apooo bookclub|
|a guilty pleasure...|
|in the tradition ...|
|beryl unger top e...|
|what follows is a...|
|the book flap say...|
|id never before r...|
|the novels narrat...|
|it is centered on...|
|if you like moder...|
|beryl unger is a ...|
+--------------------+
only showing top 25 rows

None
```

6. Get dataframe containing unique words and their counts

Your task:

1. Split each sentence into words based on the delimiter space (' ').
2. Put each word in each sentence row into their own rows. Put your results into a new dataframe.
3. Print out the first 5 rows of the dataframe.

1. First 5 rows of the output dataframe.

In [8]:

```
# We assemble the 'split' and 'explode' column expressions, then apply them to t

# YOUR CODE HERE for printing the first 5 rows of the dataframe after the requir
import pyspark.sql.functions as f
words_df = (textDF_new.select(f.split(textDF_new.sentence, ' ').alias('words')))
```

```
word_df = (words_df.select(f.explode(words_df.words).alias('word')))
print(word_df.show(5, truncate = True))
```

```
+-----+
| word|
+-----+
| this|
|  was|
|  the|
|first|
| time|
+-----+
only showing top 5 rows

None
```

Filter out all empty rows in the dataframe.

In [9]:
```
#YOUR CODE HERE
clean_word_df = (word_df.where(word_df.word != ''))
```

Group the dataframe by unique words, then count each group

In [10]:
```
# YOUR CODE HERE
word_count = clean_word_df.groupBy("word").count()
```

7. Sort the word count dataframe in a descending manner.

Your task:

1. Sort the previous dataframe by the counts column in a descending manner. Put your results into a new dataframe.

Expected output:

1. First 25 rows of the sorted word count dataframe. The first row would have the maximum count.

In [11]:
```
# Sort the dataframe by the 'count' column
sorted_word_count = (word_count.orderBy('count', ascending = 0))
print(sorted_word_count.show(25, truncate = True))
```

```
+-----+--------+
| word|   count|
+-----+--------+
|  the|10642903|
|    i| 6326216|
|   to| 5607568|
|  and| 5537690|
|    a| 5166838|
|   it| 4654902|
|   is| 3242588|
|  for| 2860227|
| this| 2845219|
|   of| 2782166|
|   my| 2319813|
|   in| 2147373|
| with| 2046990|
| that| 1983044|
|   on| 1758801|
|  you| 1754054|
| have| 1632887|
|  but| 1508591|
|  not| 1460730|
|  was| 1434985|
|   as| 1185866|
|  are| 1007811|
|   so|  994529|
|great|  988223|
| very|  893737|
+-----+--------+
only showing top 25 rows

None
```

8. Record the execution time

Your task:

1. Print the execution time.

Expected output: The execution time. No particular value is expected.

In [12]:
```
# Print the time since execution start - You will need this value later.
print(time.time() - start_time)
```

```
39.888976097106934
```

9. Save the sorted word counts directly to S3 as a CSV file

NOTE: Spark uses a distributed memory system, and stores working data in fragments known as "partitions". This is advantageous when a Spark cluster spans multiple machines, as each machine will only require part of the working data to do its own job. By default, Spark will save each of these data partitions into a individual file to avoid I/O collisions. We want only one output file, so we'll need to fuse all the data into a single partition first.

Your task:

1. Coalesce the previous dataframe to one partition. This makes sure that all our results will end up in the same CSV file.
2. Save the 1-partition dataframe to S3 using the DataFrame.write.csv() method. Take note to store the file inside S3, at a place that you can remember. The save path should look something like `s3://<your-bucket>/<your-folder>/<your-result-file>.csv`. Change these parameters to point to your bucket and folder.

3. Remember to save the csv file along with the header

Note:

You only need to run the section 9 and section 10 once for the 5M dataset.

Section 11 requires you to run multiple iterations of this Notebook, and for that you can comment out the code in section 9 so that it's easier for you to run.

In [13]:

```
# Save results to S3

sorted_word_count.coalesce(1).write.csv("s3://dse230-notebooks-bucket/wordCounts
```

In [14]:

```
# Stop Spark session

spark.stop()
```

10. Download the CSV file from S3 to your local machine and create the expected CSV output file

1. Navigate to the S3 folder where you stored your output
2. Note the name of this file, it should look something like `part-00000-xx.....xx.csv`.
3. Click on this file, it should open the file properties.
4. Beside 'Copy S3 URI', click on 'Object actions' and then click on 'Download'.
5. After downloading the file, you can rename it to anthing, say `results.csv`.
6. We want you to submit a CSV containing the first 101 rows of the results file. To do this, use the command `head -n 101 results.csv > 101_rows.csv` on a terminal. You can also do so manually, since CSV files are in plain text. Remember that we want the first 101 lines which would include the header as well - so basically it is header + 100 rows.

11. Execution times on different dataset and settings.

You need to experiment with using different number of master and worker nodes for running this whole Jupyter Notebook. You will have to report the execution time of this Notebook as you noted in an earlier section.

1. Create a cluster with the required number of master and worker nodes.
2. Then go to the Kernel tab in JupyterLab, and do 'Restart and run all cells.'
3. You should note the time in the cell just before section 9 - this is the time that it took for all the code to run.
4. Then, start a new cluster with a different configuration of master and worker nodes and dataset as expected. Run the Notebook again, and note the execution times.

Fill in the times in the table below.

Dataset	#Master Nodes	#Core Nodes	Runtime_1	Runtime_2	Runtime_3	Mean	Std
1M	1	1	32.64	32.61	33.13	32.79	0.238
5M	1	1	73.27	73.29	73.17	73.24	0.052
5M	1	3	39.49	39.96	39.89	39.78	0.207

12. Screenshots of terminated EMR clusters

You need to attach a screenshot of your Amazon EMR 'Clusters' page which shows that all of your clusters have been terminated after you are done with your assignment.

In [ ]:

```

```