

PD1: Building a word count application

MTI850 - Big Data Analytics

Fall 2022

Equipe 1

```
Entrée [ ]: ! sudo apt-get update
! sudo mkdir -p /usr/share/man/man1
! sudo apt-get install -y openjdk-11-jdk
! pip install pyspark
```



The volume of unstructured text in existence is growing dramatically, and Spark is an excellent tool for analyzing this type of data. In this lab, we will write code that calculates the most common words in the [Complete Works of William Shakespeare \(http://www.gutenberg.org/ebooks/100\)](http://www.gutenberg.org/ebooks/100) retrieved from [Project Gutenberg \(http://www.gutenberg.org\)](http://www.gutenberg.org). This could also be scaled to larger applications, such as finding the most common words in Wikipedia.

In this PD we will cover:

- *Part 1:* Creating a base DataFrame and performing operations
- *Part 2:* Counting with Spark SQL and DataFrames
- *Part 3:* Finding unique words and a mean value
- *Part 4:* Apply word count to a file

Note that for reference, you can look up the details of the relevant methods in [Spark's Python API \(https://spark.apache.org/docs/latest/api/python/reference/index.html\)](https://spark.apache.org/docs/latest/api/python/reference/index.html).

Spark Context Initialization and Imports

```
Entrée [2]: import findspark
findspark.init()

# Test module for MTI850
import testmti850

# Util module for MTI850
import utilmti850

import pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local") \
    .appName("Web Server Log Analysis") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Part 1: Creating a base DataFrame and performing operations

In this part of the lab, we will explore creating a base DataFrame with `spark.createDataFrame` and using DataFrame operations to count words.

(1a) Create a DataFrame

We'll start by generating a base DataFrame by using a Python list of tuples and the `spark.createDataFrame` method. Then we'll print out the type and schema of the DataFrame. The Python API has several examples for using the `createDataFrame` [method](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.SQLContext.createDataFrame) (<http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.SQLContext.createDataFrame>).

```
Entrée [3]: wordsDF = spark.createDataFrame([('cat',), ('elephant',), ('rat',), ('r
wordsDF.show()

print (type(wordsDF))

wordsDF.printSchema()
```

```
+-----+
|      word|
+-----+
|      cat|
|elephant|
|      rat|
|      rat|
|      cat|
+-----+
```

```
<class 'pyspark.sql.dataframe.DataFrame'>
root
 |-- word: string (nullable = true)
```

DataFrame Visualization

For a better visualization of a DataFrame you can use 'DataFrame.limit().toPandas().head()'

```
Entrée [4]: wordsDF.limit(5).toPandas().head()
```

Out[4]:

	word
0	cat
1	elephant
2	rat
3	rat
4	cat

(1b) Using DataFrame functions to add an 's'

Let's create a new DataFrame from `wordsDF` by performing an operation that adds an 's' to each word. To do this, we'll call the `select` [DataFrame function \(http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.select\)](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.select) and pass in a column that has the recipe for adding an 's' to our existing column. To generate this `Column` object you should use the `concat` [function \(http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.concat\)](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.concat) found in the `pyspark.sql.functions` [module \(http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#module-pyspark.sql.functions\)](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#module-pyspark.sql.functions). Note that `concat` takes in two or more string columns and returns a single string column. In order to pass in a constant or literal value like 's', you'll need to wrap

that value with the `lit` `column function` (<http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.lit>).

Please replace `<FILL IN>` with your solution.

After you have created `pluralDF` you can run the next cell which contains two tests. If your implementation is correct it will print `1 test passed` for each test.

This is the general form that exercises will take. Exercises will include an explanation of what is expected, followed by code cells where one cell will have one or more `<FILL IN>` sections.

The cell that needs to be modified will have:

```
# TODO: Replace <FILL IN> with appropriate code
```

on its first line.

Once the `<FILL IN>` sections are updated and the code is run, the test cell can then be run to verify the correctness of your solution. The last code cell before the next markdown section will contain the tests.

Note: Make sure that the resulting DataFrame has one column which is named 'word'.

Entrée [5]:

```
# TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import lit, concat

pluralDF = wordsDF.select(concat(wordsDF.word, lit("s")).alias("word"))

pluralDF.show()
pluralDF.limit(5).toPandas().head()
```

```
+-----+
|      word|
+-----+
|      cats|
|elephants|
|       rats|
|       rats|
|       cats|
+-----+
```

Out[5]:

	word
0	cats
1	elephants
2	rats
3	rats

Test (1b)

Entrée [6]: `# TEST Using DataFrame functions to add an 's' (1b)`
`testmti850.Test.assertEquals(pluralDF.first()[0], 'cats', 'incorrect re`
`testmti850.Test.assertEquals(pluralDF.columns, ['word'], "there should`

1 test passed.
 1 test passed.

(1c) Length of each word

Now use the SQL `length` function to find the number of characters in each word. The `length` function (<http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.length>) is found in the `pyspark.sql.functions` module.

Entrée [7]: `# TODO: Replace <FILL IN> with appropriate code`
`from pyspark.sql.functions import length`
`pluralLengthsDF = pluralDF.select(length(pluralDF.word).alias("Length w`
`pluralLengthsDF.show()`
`pluralLengthsDF.limit(5).toPandas().head()`

```
+-----+
|Length word|
+-----+
|          4|
|          9|
|          4|
|          4|
|          4|
+-----+
```

Out[7]:

	Length word
0	4
1	9
2	4
3	4
4	4

Test (1c)

Entrée [8]:

```
# TEST Length of each word (1c)
from collections.abc import Iterable
asSelf = lambda v: map(lambda r: r[0] if isinstance(r, Iterable) and le
testmti850.Test.assertEqual(set(asSelf(pluralLengthsDF.collect())), {4
'incorrect values for pluralLengths')
```

1 test passed.

Part 2: Counting with Spark SQL and DataFrames

Now, let's count the number of times a particular word appears in the 'word' column. There are multiple ways to perform the counting, but some are much less efficient than others.

A naive approach would be to call `collect` on all of the elements and count them in the driver program. While this approach could work for small datasets, we want an approach that will work for any size dataset including terabyte- or petabyte-sized datasets. In addition, performing all of the work in the driver program is slower than performing it in parallel in the workers. For these reasons, we will use data parallel operations.

Entrée [9]:

```
# TODO: Replace <FILL IN> with appropriate code

wordCountsDF = wordsDF.groupBy("word").count()

wordCountsDF.show()
```

```
+-----+-----+
|   word|count|
+-----+-----+
|    rat|    2|
|    cat|    2|
|elephant|    1|
+-----+-----+
```

Test (2a)

Entrée [10]:

```
# TEST groupBy and count (2a)
testmti850.Test.assertEqual(set(wordCountsDF.collect()), {('rat', 2),
'incorrect counts for wordCountsDF')
```

1 test passed.

Part 3: Finding unique words and a mean value

(3a) Unique words

Calculate the number of unique words in `wordsDF` . You can use other DataFrames that you

```
Entrée [11]: #This function returns all the DataFrames in the notebook and their co  
utilmti850.printDataFrames(True)
```

```
wordsDF: ['word']  
pluralDF: ['word']  
pluralLengthsDF: ['Length word']  
wordCountsDF: ['word', 'count']
```

```
Entrée [12]: # TODO: Replace <FILL IN> with appropriate code
```

```
uniqueWordsCount = wordCountsDF.count()  
  
print (uniqueWordsCount)
```

3

Test (3a)

```
Entrée [13]: # TEST Unique words (3a)  
testmti850.Test.assertEqual(uniqueWordsCount, 3, 'incorrect count of
```

1 test passed.

(3b) Means of groups using DataFrames

Find the mean number of occurrences of words in `wordCountsDF` .

You should use the `mean` [GrouppedData method \(http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.GrouppedData.mean\)](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.GrouppedData.mean) to accomplish this. Note that when you use `groupBy` you don't need to pass in any columns. A call without columns just prepares the DataFrame so that aggregation functions like `mean` can be applied.

```
Entrée [14]: # TODO: Replace <FILL IN> with appropriate code
```

```
averageCount = (wordCountsDF.groupBy()).mean('count').take(1)[0][0]  
  
print (averageCount)
```

1.6666666666666667

Test (3b)

```
Entrée [15]: # TEST Means of groups using DataFrames (3b)  
testmti850.Test.assertEqual(round(averageCount, 2), 1.67, 'incorrect
```

1 test passed.

Part 4: Apply word count to a file

In this section we will finish developing our word count application. We'll have to build the `wordCount` function, deal with real world problems like capitalization and punctuation, load in our data source, and compute the word count on the new data.

(4a) The `wordCount` function

First, define a function for word counting. You should reuse the techniques that have been covered in earlier parts of this lab. This function should take in a `DataFrame` that is a list of words like `wordsDF` and return a `DataFrame` that has all of the words and their associated counts.

```
Entrée [16]: # TODO: Replace <FILL IN> with appropriate code

def wordCount(wordListDF):
    """Creates a DataFrame with word counts.

    Args:
        wordListDF (DataFrame of str): A DataFrame consisting of one s

    Returns:
        DataFrame of (str, int): A DataFrame containing 'word' and 'co
    """
    return wordListDF.groupBy("word").count()

wordCount(wordsDF).show()
```

```
+-----+-----+
|   word|count|
+-----+-----+
|    rat|    2|
|    cat|    2|
|elephant|    1|
+-----+-----+
```

Test (4a)

```
Entrée [17]: # TEST wordCount function (4a)
res = [(row[0], row[1]) for row in wordCount(wordsDF).collect()]
testmti850.Test.assertEquals(sorted(res),
                              [('cat', 2), ('elephant', 1), ('rat', 2)],
                              'incorrect definition for wordCountDF function')
```

1 test passed.

(4b) Capitalization and punctuation

Real world files are more complicated than the data we have been using in this lab. Some of

the issues we have to address are:

- Words should be counted independent of their capitalization (e.g., Spark and spark should be counted as the same word).
- All punctuation should be removed.
- Any leading or trailing spaces on a line should be removed.

Define the function `removePunctuation` that converts all text to lower case, removes any punctuation, and removes leading and trailing spaces. Use the Python [`regexp_replace`](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.regexp_replace) (http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.regexp_replace) module to remove any text that is not a letter, number, or space. If you are unfamiliar with regular expressions, you may want to review [this tutorial](https://developers.google.com/edu/python/regular-expressions) (<https://developers.google.com/edu/python/regular-expressions>) from Google. Also, [this website](https://regex101.com/#python) (<https://regex101.com/#python>) is a great resource for debugging your regular expression.

You should also use the `trim` and `lower` functions found in [pyspark.sql.functions](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions) (<http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions>).

Note that you shouldn't use any RDD operations or need to create custom user defined functions (udfs) to accomplish this task

```

Entrée [18]: # TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import regexp_replace, trim, col, lower
import re

def removePunctuation(column):
    """Removes punctuation, changes to lower case, and strips leading

    Note:
        Only spaces, letters, and numbers should be retained. Other c
        eliminated (e.g. it's becomes its). Leading and trailing spac
        punctuation is removed.

    Args:
        column (Column): A Column containing a sentence.

    Returns:
        Column: A Column named 'sentence' with clean-up operations app
    """

    return lower(trim(regexp_replace(column, '[^A-Za-z0-9 ]+', ''))).ali

sentenceDF = spark.createDataFrame([('Hi, you!',),
                                     (' No under_score!',),
                                     (' *      Remove punctuation
sentenceDF.show(truncate=False)

(sentenceDF
 .select(removePunctuation(col('sentence'))))
 .show(truncate=False))

```

```

+-----+
|sentence|
+-----+
|Hi, you!|
| No under_score!|
| *      Remove punctuation then spaces * |
+-----+

```

```

+-----+
|sentence|
+-----+
|hi you|
|no underscore|
|remove punctuation then spaces|
+-----+

```

Test (4b)

Entrée [19]: `# TEST Capitalization and punctuation (4b)`
`testPunctDF = spark.createDataFrame([(" The Elephant's 4 cats. ",)])`
`testmti850.Test.assertEquals(testPunctDF.select(removePunctuation(col(`
 `'the elephants 4 cats',`
 `'incorrect definition for removePunctuation function`

1 test passed.

(4c) Load a text file

For the next part of this lab, we will use the [Complete Works of William Shakespeare](http://www.gutenberg.org/ebooks/100) (<http://www.gutenberg.org/ebooks/100>) from [Project Gutenberg](http://www.gutenberg.org/wiki/Main_Page) (http://www.gutenberg.org/wiki/Main_Page). To convert a text file into a DataFrame, we use the

`sqlContext.read.text()` method. We also apply the recently defined `removePunctuation()` function using a `select()` transformation to strip out the punctuation and change all text to lower case. Since the file is large we use `show(15)`, so that we only print 15 lines.

Entrée [20]: `#fileName = "hdfs://localhost:9000/shakespeare.txt"`
`fileName = "shakespeare.txt"`

`shakespeareDF = spark.read.text(fileName).select(removePunctuation(col`
`shakespeareDF.show(15, truncate=False)`

```
+-----+
+-----+
|sentence|
|
+-----+
+-----+
|project gutenbergs the complete works of william shakespeare by willi|
|am shakespeare|
|
|
|this ebook is for the use of anyone anywhere in the united states and|
|
|most other parts of the world at no cost and with almost no restricti|
|ons|
|whatsoever you may copy it give it away or reuse it under the terms|
|
|of the project gutenberg license included with this ebook or online a|
|t|
|wwwgutenbergorg if you are not located in the united states youll|
|
```

```
Entrée [21]: shakespeareDF.limit(10).toPandas().head()
```

```
Out[21]:
```

	sentence
0	project gutenbergs the complete works of willi...
1	
2	this ebook is for the use of anyone anywhere i...
3	most other parts of the world at no cost and w...
4	whatsoever you may copy it give it away or re...

(4d) Words from lines

Before we can use the `wordcount()` function, we have to address two issues with the format of the DataFrame:

- The first issue is that that we need to split each line by its spaces.
- The second issue is we need to filter out empty lines or words.

Apply a transformation that will split each 'sentence' in the DataFrame by its spaces, and then transform from a DataFrame that contains lists of words into a DataFrame with each word in its own row. To accomplish these two tasks you can use the `split` and `explode` functions found in [pyspark.sql.functions](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions) (<http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions>).

Once you have a DataFrame with one word per row you can apply the [DataFrame operation](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.where) where [_](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.where) (<http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.where>) to remove the rows that contain "".

Note that `shakeWordsDf` should be a DataFrame with one column named `word`.

```
Entrée [22]: # TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import split, explode

shakeWordsDF = (shakespeareDF.select(explode(split(shakespeareDF.sente

shakeWordsDF = shakeWordsDF.filter(shakeWordsDF.col != "")
shakeWordsDF = shakeWordsDF.withColumnRenamed("col", "word")

shakeWordsDF.show()

shakeWordsDFCount = shakeWordsDF.count()

print (shakeWordsDFCount)
```

```
+-----+
|      word|
+-----+
|    project|
| gutenbergs|
|      the|
| complete|
|      works|
|      of|
|    william|
|shakespeare|
|      by|
|    william|
|shakespeare|
|      this|
|      ebook|
|      is|
|      for|
|      the|
| .....
```

Test (4d)

```
Entrée [23]: # TEST Remove empty elements (4d)
testmti850.Test.assertEquals(shakeWordsDF.count(), 961306, 'incorrect
testmti850.Test.assertEquals(shakeWordsDF.columns, ['word'], "shakeWor

1 test passed.
1 test passed.
```

(4e) Count the words

We now have a DataFrame that is only words. Next, let's apply the `wordCount()` function to produce a list of word counts. We can view the first 20 words by using the `show()` action; however, we'd like to see the words in descending order of count, so we'll need to apply the `orderBy` [DataFrame method \(http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.orderBy\)](http://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame.orderBy) to first sort the DataFrame that is returned from `wordCount()`.

You'll notice that many of the words are common English words. These are called stopwords. In a later lab, we will see how to eliminate them from the results.

Entrée [24]: `# TODO: Replace <FILL IN> with appropriate code`
`from pyspark.sql.functions import desc`
`topWordsAndCountsDF = wordCount(shakeWordsDF).orderBy(col("count").desc)`
`topWordsAndCountsDF.show()`

```
+-----+-----+
|word|count|
+-----+-----+
| the|30205|
| and|28386|
|  i|21949|
| to|20923|
| of|18822|
|  a|16182|
| you|14437|
| my|13180|
| in|12232|
| that|11776|
|  is| 9713|
| not| 9066|
| with| 8528|
|  me| 8263|
| for| 8195|
|  it| 8180|
|  ...| 7500|
```

Test (4e)

Entrée [25]: `# TEST Count the words (4e)`
`testmti850.Test.assertEquals(topWordsAndCountsDF.take(15),`
`[(u'the', 30205), (u'and', 28386), (u'i', 21949), (u'`
`(u'a', 16182), (u'you', 14437), (u'my', 13180), (u'`
`(u'is', 9713), (u'not', 9066), (u'with', 8528), (u'`
`'incorrect value for top15WordsAndCountsDF')`

1 test passed.



(<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

Adapted from Databricks notebooks. This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](http://creativecommons.org/licenses/by-nc-nd/4.0/)

(<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

 Created **Deepnote**(https://deepnote.com?utm_source=created-in-deepnote-cell&projectId=f3e36fb3-ac28-4ed6-af6f-c82db9d8acf7) in

