PD1-Word_Count-MTI850-A24

September 24, 2025

1 PD1: Building a word count application

1.1 MTI850 - Big Data Analytics

V2.0 - Fall 2022V2.1 - Fall 2024

Equipe 9

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 retrieved from Project Gutenberg. 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.

1.2 Spark Context Initialization and Imports

[1]:
```
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()
```

```
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
25/09/24 12:11:42 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
```

1.3 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.

1.3.1 (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.

[2]:
```
wordsDF = spark.createDataFrame([('cat',), ('elephant',), ('rat',), ('rat',),␣
↪('cat', )], ['word'])

wordsDF.show()

print (type(wordsDF))

wordsDF.printSchema()
```

```
[Stage 0:>                                                          (0 + 1) / 1]
```

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

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

1.3.2 DataFrame Visualization

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

[3]:
```
wordsDF.limit(5).toPandas().head()
```

[3]:

	word
0	cat
1	elephant
2	rat
3	rat
4	cat

1.3.3 (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 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 found in the `pyspark.sql.functions` module. 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.

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

After you have created `pluralDF` you can run the next cell which contains two tests. If you 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'.

[4]:
```
# 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|
+---------+
```

[4]:

	word
0	cats
1	elephants
2	rats
3	rats
4	cats

1.3.4 Test (1b)

[5]:
```
# TEST Using DataFrame functions to add an 's' (1b)
testmti850.Test.assertEquals(pluralDF.first()[0], 'cats', 'incorrect result: you need to add an s')
testmti850.Test.assertEquals(pluralDF.columns, ['word'], "there should be one column named 'word'")
```

```
1 test passed.
1 test passed.
```

1.3.5 (1c) Length of each word

Now use the SQL `length` function to find the number of characters in each word. The `length` function is found in the `pyspark.sql.functions` module.

[6]:
```
# TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import length
```

```
pluralLengthsDF = pluralDF.select(length(pluralDF.word))

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

```
+------------+
|length(word)|
+------------+
|           4|
|           9|
|           4|
|           4|
|           4|
+------------+
```

[6]:

	length(word)
0	4
1	9
2	4
3	4
4	4

1.3.6 Test (1c)

[7]:
```
# TEST Length of each word (1c)
from collections.abc import Iterable
asSelf = lambda v: map(lambda r: r[0] if isinstance(r, Iterable) and len(r) ==␣
 ↪1 else r, v)

testmti850.Test.assertEquals(set(asSelf(pluralLengthsDF.collect())), {4, 9, 4,␣
 ↪4, 4},
                  'incorrect values for pluralLengths')
```

```
1 test passed.
```

1.4 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.

[14]:
```
# TODO: Replace <FILL IN> with appropriate code
wordCountsDF = wordsDF.groupBy(wordsDF.word).count()
wordCountsDF.show()
```

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

1.4.1 Test (2a)

[15]:
```
# TEST groupBy and count (2a)
testmti850.Test.assertEquals(set(wordCountsDF.collect()), {('rat', 2),␣
 ↪('elephant', 1), ('cat', 2)},
                 'incorrect counts for wordCountsDF')
```

```
1 test passed.
```

1.5 Part 3: Finding unique words and a mean value

1.5.1 (3a) Unique words

Calculate the number of unique words in `wordsDF`. You can use other DataFrames that you have already created to make this easier.

[10]:
```
#This function returns all the DataFrames in the notebook and their␣
 ↪corresponding column names.
utilmti850.printDataFrames(True)
```

```
wordsDF: ['word']
pluralDF: ['word']
pluralLengthsDF: ['length(word)']
wordCountsDF: ['word', 'count']
```

[11]:
```
# TODO: Replace <FILL IN> with appropriate code
uniqueWordsCount = wordCountsDF.count()
print(uniqueWordsCount)
```

```
3
```

1.5.2 Test (3a)

[12]:
```
# TEST Unique words (3a)
testmti850.Test.assertEquals(uniqueWordsCount, 3, 'incorrect count of unique␣
 ↪words')
```

```
1 test passed.
```

1.5.3 (3b) Means of groups using DataFrames

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

You should use the `mean` GroupedData method 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.

[17]:
```
# TODO: Replace <FILL IN> with appropriate code

averageCountDF = wordCountsDF.groupBy().mean("count")
averageCountDF.show()
averageCount = averageCountDF.collect()[0][0]

print (averageCount)
```

```
+------------------+
|        avg(count)|
+------------------+
|1.6666666666666667|
+------------------+

1.6666666666666667
```

1.5.4 Test (3b)

[ ]:
```
# TEST Means of groups using DataFrames (3b)
testmti850.Test.assertEquals(round(averageCount, 2), 1.67, 'incorrect value of␣
 ↪averageCount')
```

1.6 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.

1.6.1 (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.

[19]:
```
# 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 string␣
↪column called 'word'.

    Returns:
        DataFrame of (str, int): A DataFrame containing 'word' and 'count'␣
↪columns.
    """

    return wordListDF.groupBy(wordsDF.word).count()

wordCount(wordsDF).show()
```

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

1.6.2 Test (4a)

[20]:
```
# 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.
```

1.6.3 (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 capitialization (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 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 from Google. Also, this website is a great resource for debugging your regular expression.

You should also use the `trim` and `lower` functions found in 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

[24]:
```
# TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import regexp_replace, trim, col, lower

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

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

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

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

    return trim(lower(regexp_replace(column, "[^a-zA-Z0-9 ]", "")))



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

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

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


[Stage 57:>                                                         (0 + 1) / 1]

+-------------------------------------------------------+
|trim(lower(regexp_replace(sentence, [^a-zA-Z0-9 ], , 1)))|
```

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

1.6.4 Test (4b)

[25]:
```
# TEST Capitalization and punctuation (4b)
testPunctDF = spark.createDataFrame([(" The Elephant's 4 cats. ",)])
testmti850.Test.assertEquals(testPunctDF.select(removePunctuation(col('_1'))).
 ↪first()[0],
                  'the elephants 4 cats',
                  'incorrect definition for removePunctuation function')
```

```
1 test passed.
```

1.7 (4c) Load a text file

For the next part of this lab, we will use the Complete Works of William Shakespeare from Project Gutenberg. 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.

[38]:
```
fileName = "hdfs://localhost:9000/user/vboxuser/shakespeare.txt"

shakespeareDF = spark.read.text(fileName).
 ↪select(removePunctuation(col('value')).alias("text"))

shakespeareDF.show(15, truncate=False)
```

```
+-------------------------------------------------------------------------------
----+
|text
|
+-------------------------------------------------------------------------------
----+
|project gutenbergs the complete works of william shakespeare by william
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 restrictions
|
```

```
|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 at
|
|wwwgutenbergorg  if you are not located in the united states youll
|
|have to check the laws of the country where you are located before using
|
|this ebook
|
|
|
|
|
|title the complete works of william shakespeare
|
|
|
|author william shakespeare
|
|
|
+--------------------------------------------------------------------------------
----+
only showing top 15 rows
```

[39]:
```
shakespeareDF.limit(10).toPandas().head()
```

[39]:

	text
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…

1.7.1 (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.

Once you have a DataFrame with one word per row you can apply the DataFrame operation `where` to remove the rows that contain ''.

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

[42]:
```
# TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import split, explode


shakeWordsDF = shakespeareDF.select(explode(split(shakespeareDF["text"], " ")).
 ↪alias("word")).where(col("word") != "")
# split separe tous les mots dans une liste en utilisant l'espace comme␣
 ↪separateur
# explode prend la liste en entree, retourne une ligne par element
# where nous permet de garder uniquement les lignes ou le mot est non vide


shakeWordsDF.show()

shakeWordsDFCount = shakeWordsDF.count()

print (shakeWordsDFCount)
```

```
+-----------+
|       word|
+-----------+
|    project|
| gutenbergs|
|        the|
|   complete|
|      works|
|         of|
|    william|
|shakespeare|
|         by|
|    william|
|shakespeare|
|       this|
|      ebook|
|         is|
|        for|
|        the|
|        use|
|         of|
|     anyone|
|   anywhere|
+-----------+
only showing top 20 rows
961306
```

1.7.2 Test (4d)

[43]:
```
# TEST Remove empty elements (4d)
testmti850.Test.assertEquals(shakeWordsDF.count(), 961306, 'incorrect value for␣
 ↪shakeWordCount')
testmti850.Test.assertEquals(shakeWordsDF.columns, ['word'], "shakeWordsDF␣
 ↪should only contain the Column 'word'")
```

```
1 test passed.
1 test passed.
```

1.7.3 (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 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.

[46]:
```
# TODO: Replace <FILL IN> with appropriate code
from pyspark.sql.functions import desc

topWordsAndCountsDF = shakeWordsDF.groupBy("word").count().
 ↪orderBy(desc("count"))

topWordsAndCountsDF.show(20)
```

```
[Stage 78:>                                                         (0 + 1) / 1]

+----+-----+
|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|
| his| 7581|
|  be| 7370|
|this| 7178|
|your| 7076|
+----+-----+
only showing top 20 rows
```

1.7.4 Test (4e)

[47]:
```
# TEST Count the words (4e)
testmti850.Test.assertEquals(topWordsAndCountsDF.take(15),
                  [(u'the', 30205), (u'and', 28386), (u'i', 21949), (u'to',␣
 ↪20923), (u'of', 18822),
                   (u'a', 16182), (u'you', 14437), (u'my', 13180), (u'in',␣
 ↪12232), (u'that', 11776),
                   (u'is', 9713), (u'not', 9066), (u'with', 8528), (u'me',␣
 ↪8263), (u'for', 8195)],
                  'incorrect value for top15WordsAndCountsDF')
```

```
[Stage 81:>                                                         (0 + 1) / 1]
```

```
1 test passed.
```