Word Count Lab: Building a word count application

This lab will build on the techniques covered in the Spark tutorial to develop a simple word count application. 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) retrieved from Project Gutenberg (http://www.gutenberg.org/wiki/Main_Page). This could also be scaled to larger applications, such as finding the most common words in Wikipedia.

During this lab 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/pyspark.html#pyspark.sql).

Part 1: Creating a base DataFrame and performing operations

In this part of the lab, we will explore creating a base DataFrame with `sqlContext.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 `sqlContext.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/latest/api/python/pyspark.sql.html#pyspark.sql.SQLConte

```
# Write the code here:
wordsDF=sqlContext.createDataFrame([('anything',1),('car',2),('data',3),
('epilepsy',4)],['word','number'])
wordsDF.printSchema()
```

```
root
 |-- word: string (nullable = true)
 |-- number: long (nullable = true)
```

(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/latest/api/python/pyspark.sql.html#pyspark.sql.DataFram 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/latest/api/python/pyspark.sql.html#pyspark.sql.functions.(found in the `pyspark.sql.functions` module (http://spark.apache.org/docs/latest/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/latest/api/python/pyspark.sql.html#pyspark.sql.functions.

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

```
# Write the code here:
from pyspark.sql.functions import lit, concat
wordsDF=sqlContext.createDataFrame([('cat',),('elephant',),('rat',),('rat',),
('cat',)],['word'])
pluralDF=wordsDF.select(concat(wordsDF.word,lit('s')).alias('word'))
#Actions:
#pluralDF.take(1)
#pluralDF.collect()
pluralDF.show()
# pluralDF = ?
```

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

```
wordsDF.show()
```

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

```
# Load in the testing code and check to see if your answer is correct
# If incorrect it will report back '1 test failed' for each failed test
# Make sure to rerun any cell you change before trying the test again
from databricks_test_helper import Test
# TEST Using DataFrame functions to add an 's' (1b)
Test.assertEquals(pluralDF.first()[0], 'cats', 'incorrect result: you need to
add an s')
Test.assertEquals(pluralDF.columns, ['word'], "there should be one column named
'word'")
```

```
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/latest/api/python/pyspark.sql.html#pyspark.sql.functions. is found in the `pyspark.sql.functions` module.

```
# Write the code here:
from pyspark.sql.functions import length

pluralLengthsDF=pluralDF.select('word',length('word').alias('length'))

display(pluralLengthsDF)


# pluralLengthsDF = ?
```

word
cats
elephants
rats
rats
cats

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.

(2a) Using `groupBy` and `count`

Using DataFrames, we can preform aggregations by grouping the data using the `groupBy` function (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFram on the DataFrame. Using `groupBy` returns a `GroupedData` object (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.GroupedD and we can use the functions available for `GroupedData` to aggregate the groups. For example, we can call `avg` or `count` on a `GroupedData` object to obtain the average of the values in the groups or the number of occurrences in the groups, respectively.

To find the counts of words, group by the words and then use the `count` function (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.GroupedD to find the number of times that words occur.

```
# Write the code here:
from pyspark.sql.functions import count
wordCountsDF = wordsDF.groupBy('word').count()


display(wordCountsDF)


# wordCountsDF = ?
```

word
rat
cat
elephant

```
# TEST groupBy and count (2a)
Test.assertEquals(set(wordCountsDF.collect()), {('rat', 2), ('elephant', 1),
('cat', 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 have already created to make this easier.

```
from spark_notebook_helpers import printDataFrames


#This function returns all the DataFrames in the notebook and their
corresponding column names.
printDataFrames(True)
```

```
pluralLengthsDF: ['word', 'length']
wordCountsDF: ['word', 'count']
```

```
pluralDF: ['word']
wordsDF: ['word']

# Write the code here:

uniqueWordsCount=wordsDF.distinct().groupBy().count().head()[0]
print(dir(wordsDF))
wordsDF.createOrReplaceTempView('WordsTable')
wordsTable=spark.sql("SELECT*FROM WordsTable WHERE length(word)>3")

#print(type(wordsTable),type(wordsDF))
display(wordsTable)
#print(uniqueWordsCount)
# uniqueWordsCount = ?
```

word
elephant

```
%sql
SELECT*FROM WordsTable;
```

word
cat
elephant
rat
rat
cat

```
# TEST Unique words (3a)
Test.assertEquals(uniqueWordsCount, 3, 'incorrect count of unique words')

1 test passed.
```

(3b) Means of groups using DataFrames

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

You should use the `mean` GroupedData method (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.GroupedD 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.

```
# Write the code here:

averageCount = wordCountsDF.groupBy().mean('count')
display(averageCount)
# averageCount = ?
```

avg(count)
1.6666666666666667

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.

```
# Write the code here:

def wordCount(wordListDF):
  """
  Returns:
  DataFrame of (str, int):containing'word' and 'count'
  """
  return wordListDF.groupBy('word').count()
display(wordCount(wordsDF))
# wordCount = ?
```

word
rat
cat
elephant

```
# TEST wordCount function (4a)
res = [(row[0], row[1]) for row in wordCount(wordsDF).collect()]
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 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 (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.functions. 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) from Google. Also, this website (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/latest/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

```
# Write the code here:
from pyspark.sql.functions import regexp_replace,trim,col,lower

def removePunctuation(column):
 """
 Returns:
   Columns:A Column named'sentence' with clean-up operations applied
 """
 return trim(lower(regexp_replace(column,'[^\sa-zA-Z0-
9]','').alias('sentence')

sentenceDF=sqlContext.createDataFrame(
   [
    ('Hi,you!',),
   (' No under/-score!',),
   (' *      Remove punctuation then spaces *',)
   ],
   ['sentence']
)
display(sentenceDF.select(removePunctuation(col('sentence'))))
# sentenceDF = ?
```

sentence
hiyou
no underscore
remove punctuation then spaces

```
# TEST Capitalization and punctuation (4b)
testPunctDF = sqlContext.createDataFrame([(" The Elephant's 4 cats. ",)])
Test.assertEquals(testPunctDF.select(removePunctuation(col('_1'))).first()[0],
                  '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) from Project Gutenberg (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.

```
# This is a demo file hosted by Databricks
fileName = "dbfs:/databricks-datasets/cs100/lab1/data-001/shakespeare.txt"

# Write the code here:


shakespeare = spark.read.text(fileName)
shakespeare.select(removePunctuation('value'))
shakespeare.take(10)
shakespeare.show(15,truncate=False)
# shakespeareDF = ?
```

```
+--------------------------------------------------------+
|value                                                   |
+--------------------------------------------------------+
|1609                                                    |
|                                                        |
|THE SONNETS                                             |
|                                                        |
|by William Shakespeare                                  |
|                                                        |
|                                                        |
|                                                        |
|                     1                                  |
|  From fairest creatures we desire increase,            |
```

```
|  That thereby beauty's rose might never die,          |
|  But as the riper should by time decease,             |
|  His tender heir might bear his memory:               |
|  But thou contracted to thine own bright eyes,        |
|  Feed'st thy light's flame with self-substantial fuel,|
+-------------------------------------------------------+
only showing top 15 rows
```

(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/latest/api/python/pyspark.sql.html#pyspark.sql.functions)

Once you have a DataFrame with one word per row you can apply the DataFrame operation `where` (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFram to remove the rows that contain ''.

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

```
from pyspark.sql.functions import split, explode
shakeWordsDF = (shakespeare
                .select(explode(split(shakespeare.value, ' ')).alias('word'))
                .where(col('word') != ''))

shakeWordsDF.show()
```

```
+-----------+
|       word|
```

```
+-----------+
|       1609|
|        THE|
|    SONNETS|
|         by|
|    William|
|Shakespeare|
|          1|
|       From|
|    fairest|
|  creatures|
|         we|
|     desire|
|  increase,|
|       That|
|    thereby|
|   beauty's|
|       rose|
|      might|
|      never|
|       die,|
+-----------+
only showing top 20 rows


# Write the code here:
shakeWordsDFCount = shakeWordsDF.count()

print shakeWordsDFCount
# shakeWordsDFCount = ?

883320

# TEST Remove empty elements (4d)
Test.assertEquals(shakeWordsDF.count(), 883320, 'incorrect value for
shakeWordCount')
Test.assertEquals(shakeWordsDF.columns, ['word'], "shakeWordsDF should only
contain the Column 'word'")

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/latest/api/python/pyspark.sql.html#pyspark.sql.DataFram 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.

```
from pyspark.sql.functions import desc
topWordsAndCountsDF = wordCount(shakeWordsDF).orderBy("count", ascending=False)
topWordsAndCountsDF.show()
```

```
+----+-----+
|word|count|
+----+-----+
| the|23197|
|   I|19540|
| and|18263|
|  to|15592|
|  of|15507|
|   a|12516|
|  my|10825|
|  in| 9565|
| you| 9059|
|  is| 7831|
|that| 7521|
| And| 7068|
| not| 6946|
|with| 6718|
| his| 6218|
|your| 6003|
|  be| 5991|
| for| 5600|
|have| 5231|
|  it| 4903|
+----+-----+
only showing top 20 rows
```

```
# TEST Count the words (4e)
Test.assertEquals(topWordsAndCountsDF.take(2),
                  [(u'the', 23197), (u'I', 19540)],
                  'incorrect value for top15WordsAndCountsDF')
```

```
1 test passed.
```

Assignment #3

Using DBFS on your Databricks community edition notebook, access the files under `dbfs:/databricks-datasets/power-plant/`

Schema for Data

AT = Atmospheric Temperature in C
V = Exhaust Vaccum Speed
AP = Atmospheric Pressure
RH = Relative Humidity
PE = Power Output

Using All files(Tab Separated Values) under `dbfs:/databricks-datasets/power-plant/data/` dataset, answer the following questions:

What is the number of unique Atmoshperic Pressure Values (AP)?

What is the standard deviation of Temperature (AT) and print result with word 'Celsius'?

```
dbutils.fs.ls("dbfs:/databricks-datasets/power-plant/data")
```

```
Out[22]:
[FileInfo(path=u'dbfs:/databricks-datasets/power-plant/data/Sheet1.tsv', name=
u'Sheet1.tsv', size=308693L),
 FileInfo(path=u'dbfs:/databricks-datasets/power-plant/data/Sheet2.tsv', name=
u'Sheet2.tsv', size=308693L),
 FileInfo(path=u'dbfs:/databricks-datasets/power-plant/data/Sheet3.tsv', name=
u'Sheet3.tsv', size=308693L),
 FileInfo(path=u'dbfs:/databricks-datasets/power-plant/data/Sheet4.tsv', name=
u'Sheet4.tsv', size=308693L),
```

```
 FileInfo(path=u'dbfs:/databricks-datasets/power-plant/data/Sheet5.tsv', name=
u'Sheet5.tsv', size=308693L)]

#write code for Question 1 (number of unique Atmoshperic Pressure Values (AP))

# take a quick peek at one of the files
dbutils.fs.head('dbfs:/databricks-datasets/power-plant/data/Sheet1.tsv')
```

```
[Truncated to first 65536 bytes]
Out[23]: u'AT\tV\tAP\tRH\tPE\n14.96\t41.76\t1024.07\t73.17\t463.26\n25.18\t62.
96\t1020.04\t59.08\t444.37\n5.11\t39.4\t1012.16\t92.14\t488.56\n20.86\t57.32\t
1010.24\t76.64\t446.48\n10.82\t37.5\t1009.23\t96.62\t473.9\n26.27\t59.44\t101
2.23\t58.77\t443.67\n15.89\t43.96\t1014.02\t75.24\t467.35\n9.48\t44.71\t1019.1
2\t66.43\t478.42\n14.64\t45\t1021.78\t41.25\t475.98\n11.74\t43.56\t1015.14\t7
0.72\t477.5\n17.99\t43.72\t1008.64\t75.04\t453.02\n20.14\t46.93\t1014.66\t64.2
2\t453.99\n24.34\t73.5\t1011.31\t84.15\t440.29\n25.71\t58.59\t1012.77\t61.83\t
451.28\n26.19\t69.34\t1009.48\t87.59\t433.99\n21.42\t43.79\t1015.76\t43.08\t46
2.19\n18.21\t45\t1022.86\t48.84\t467.54\n11.04\t41.74\t1022.6\t77.51\t477.2\n1
4.45\t52.75\t1023.97\t63.59\t459.85\n13.97\t38.47\t1015.15\t55.28\t464.3\n17.7
6\t42.42\t1009.09\t66.26\t468.27\n5.41\t40.07\t1019.16\t64.77\t495.24\n7.76\t4
2.28\t1008.52\t83.31\t483.8\n27.23\t63.9\t1014.3\t47.19\t443.61\n27.36\t48.6\t
1003.18\t54.93\t436.06\n27.47\t70.72\t1009.97\t74.62\t443.25\n14.6\t39.31\t101
1.11\t72.52\t464.16\n7.91\t39.96\t1023.57\t88.44\t475.52\n5.81\t35.79\t1012.14
\t92.28\t484.41\n30.53\t65.18\t1012.69\t41.85\t437.89\n23.87\t63.94\t1019.02\t
44.28\t445.11\n26.09\t58.41\t1013.64\t64.58\t438.86\n29.27\t66.85\t1011.11\t6
3.25\t440.98\n27.38\t74.16\t1010.08\t78.61\t436.65\n24.81\t63.94\t1018.76\t44.
51\t444.26\n12.75\t44.03\t1007.29\t89.46\t465.86\n24.66\t63.73\t1011.4\t74.52
\t444.37\n16.38\t47.45\t1010.08\t88.86\t450.69\n13.91\t39.35\t1014.69\t75.51\t
469.02\n23.18\t51.3\t1012.04\t78.64\t448.86\n22.47\t47.45\t1007.62\t76.65\t44
```

```
# We can see from the output of head() that the values are tab-delimited (\t)
and the records are delimited by the newline character (\n)
# We can also see that column headers are included in the file (AT, V, AP, RH,
PE)
# Load all 5 files into a dataframe. A wildcard character is used in the
filename to ensure all 5 files are loaded.
# Pass the sep parameter the tab character, otherwise the csv() function
assumes a comma separator
# Pass 'FAILFAST' as the mode parameter to ensure we are notified if a record
is corrupt.
# Pass the header parameter so the csv() function will expect column names in
the header record
# Pass the inferSchema parameter so the csv() function will determine that the
column values are all numeric
df = spark.read.csv('dbfs:/databricks-datasets/power-plant/data/Sheet*.tsv',
sep='\t', mode='FAILFAST', header='true', inferSchema='true')
```

```
# The output tells us that all of the columns we were expecting, were created
in the dataframe and each column contains numeric values

# Group the dataframe by the atmospheric pressure values, and provide
# a count for the number of occurrences of each distinct value
gdf = df.groupBy("AP").count()

# Count the number of groups
gdf.count()

Out[25]: 2517

# Therefore, there are 2517 unique values for atmospheric pressure in the
dataset.


#standard deviation of Temperature (AT)
from pyspark.sql.functions import stddev, col
aggdf = df.agg(stddev(col('AT')))
display(aggdf)
```

stddev_samp(AT)
7.452161658340004

```
# Just as a gut-check, let's use the describe() function to compare against the
result we calculated:
df.describe().show()

+-------+------------------+------------------+------------------+------------
------+------------------+
|summary|                AT|                 V|                AP|
RH|                PE|
+-------+------------------+------------------+------------------+------------
------+------------------+
|  count|             47840|             47840|             47840|
47840|             47840|
|   mean|19.651231187290996| 54.30580372073594|1013.2590781772572| 73.30897784
280918|454.36500940635506|
| stddev| 7.452161658340004|12.707361709685806| 5.938535418520816|14.599658352
081477| 17.06628146683769|
|    min|              1.81|             25.36|            992.89|
25.56|            420.26|
|    max|             37.11|             81.56|            1033.3|
100.16|            495.76|
```

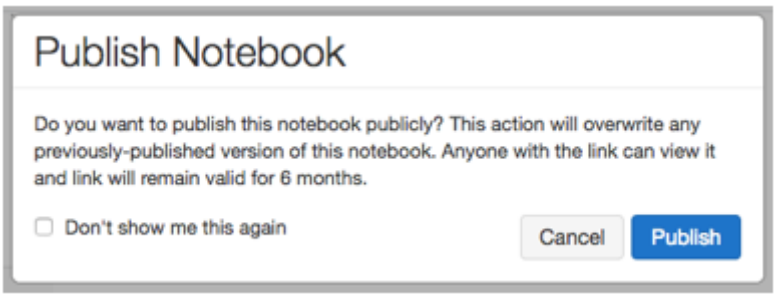
```
+-------+-----------------+-----------------+-----------------+-----------
------+------------------+

# The results match. Output the calculated value per the requirements.
ret = aggdf.select('stddev_samp(AT)').collect()[0][0]
print "{0:.15f} Celsius".format(ret)

7.452161658340004 Celsius
```

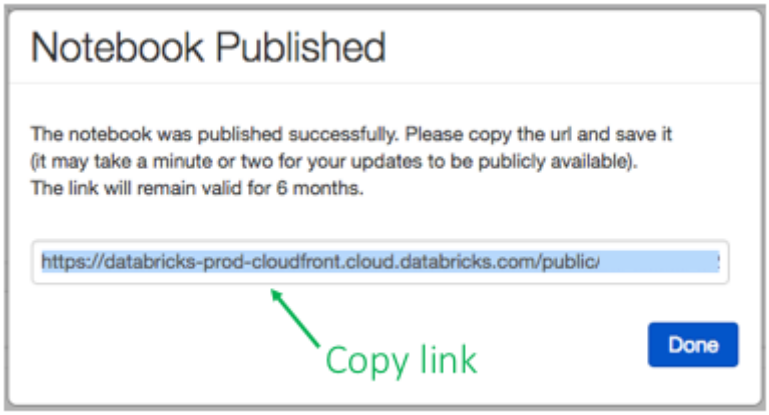
Publish your LAB notebook(this notebook) by clicking on the "Publish" button at the top of your LAB notebook.

When you click on the button, you will see the following popup.



When you click on "Publish", you will see a popup with your notebook's public link.

Copy the link and submit it using this google form with your full name: (https://goo.gl/forms/oz43FIXw5ubCilTW2)https://goo.gl/forms/oz43FIXw5ubCilTW (https://goo.gl/forms/oz43FIXw5ubCilTW2)


Notebook Published
The notebook was published successfully. Please copy the url and save it
(it may take a minute or two for your updates to be publicly available).
The link will remain valid for 6 months.
https://databricks-prod-cloudfront.cloud.databricks.com/public/
Copy link
Done