CMPSC/DS 410: EVERYTHING HOMEWORK, DUE DEC 14 (NO EXTENSIONS AFTER END OF CLASSES)

1. Instructions

1.1. What to look at.

The only references you may use for this lab are:

- The scala documentation: `https://docs.scala-lang.org/`
- Spark documentation at databricks.com
- Any material on canvas, including the spark textbook, scala cheetsheat and scala book that are linked to on the main canvas page.

In particular, do not use stackoverflow or chegg for two reasons: (1) it will result in an academic integrity violation and (2) you are likely to get a hilariously incorrect answer from them (don't be an anecdote).

1.2. How to collaborate.

- Write down (in your code) the names of people you collaborate with.
- You *cannot* look at code/pseudocode
- You *cannot* copy/email/transmit code. Your code must be typed by you without looking at other people's code.
- You can discuss algorithmic strategies (pictures are fine, pseudocode is not)

1.3. Getting the code and submitting the code.

Run **updatestarter**, submit to gradescope.
Do not use **var** or mutable state (except if you do string.split(), since then you have no choice).

2. Instructions

This assignment uses the Cities dataset in HDFS **/datasets/cities**. **It was also used in debuglab, so to save time you can look at the lab to see how to process this dataset**. It is a tab-separated dataset that has information about cities (name, state, county, population, zip codes in the city, and an ID) and has some bad lines. Each line is a city.

Some cities have 0 zip codes, some have 1 zip code, some have 2, etc. We want to know the zip code distribution in the data: how many cities have 0 zip codes, how many have 1 zip code, etc. In the output of the code you generate, the number of zip codes comes first and the number of cities comes second. For example, if the output looks like:

```
0       10
2       20
1       99
5       87
```

it means that 10 cities have 0 zip codes, 20 cities have 2 zip codes, 99 cities have 1 zip code, etc. (I made these numbers up). **For best results:** don't code right away; instead spend some time looking at the data again and thinking about what kind of processing you need to get the information being asked for. Once you have a plan, then start coding.

Question 1. Fill in the **everything/mr/cities.py** and **everything/mr/run** with **mapreduce** code to solve this problem (don't forget test data). The output should go into **everything_mr** in HDFS. The code must use a combiner (properly). In the output, the key should be the number of zip codes (e.g., 1 from the example above) and the value should be the number of cities (e.g., 99 from the example above).

Question 2. Fill in the **everything/rdd/cities.scala** file with **spark rdd** code to solve this problem (any use of SparkSession or DataFrames will get 0 points, autograder will try to check this for you). Also fill in the files needed for compilation. In the output key-value rdd, the key should be the number of zip codes (e.g., 1 from the example above) and the value should be the number of cities (e.g., 99 from the example above). These are the functions you need to fill in (don't change the rest, autograder is counting on them):

- **getSC()** to return the spark context
- **getRDD()** to return the cities dataset as an RDD.
- **doCity()**. Its input is a test rdd or the cities rdd and its output should be the RDD containing the correct answer.
- **getTestRDD()**: for your testing data
- **expectedOutput()**: the RDD you should get from the test data

Question 3. Fill in the **everything/df/cities.scala** file with **spark dataframes** code (also fill in the files needed for compilation) to solve this problem (any use of SparkContext or RDDs will get 0 points, autograder will try to check this for you). In the output dataframe, the first column should be the number of zip codes (e.g., 1 from the example above) and the second should be the number of cities (e.g., 99 from the example above). Do not save the headers in the output (the provided code makes sure the headers are not saved). **Make sure to consult the slides before starting**. These are the functions you need to fill in (don't change the rest, autograder is counting on them):

- **getDF()**: returns the spark dataframe corresponding to the city. Make sure to use a schema.
- **doCity()**: does the computation to get the answer
- **getTestDF()**: self-explanatory
- **expectedOutput()**: returns the output (as a dataframe) that you should get from the test dataframe.

It is super important to read the following:

- Working with the zipcode list in DataFrames is very awkward. The simplest way is to use user-defined-functions (udf). The starting code already defines a udf called zipCounter that you can use directly in your dataframes manipulations (e.g., "zipCounter(zipcolumn)") as if it were a built-in SQL function. zipCounter take the space-separted list of zips and returns the number of zip codes in the list.
- If you are using the "spark" variable from the spark shell, you will need to call Q3.registerZipCounter(spark) so that it knows about this udf.
- You can specify column names (when reading the file) that differ from the ones in the csv header (this way, you don't have column names with spaces in them).
- Reading errors: when spark reads csv files, if it has any problems with bad lines, it will just set appropriate fields to nulls. You will need to use the slide information about nulls to examine where the nulls occur and how to filter them.
- Nulls: make sure to properly handle nulls, but only do this for columns that are needed for the answer.