DATAFRAMES LAB, DUE: DEC 14, 11:59PM EST

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. Make sure to only commit scala code, build.sbt, build.properties, but no other files into your repository.

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

Question 1. This question uses the Cities dataset in HDFS **/datasets/cities**. 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. In the file **dataframes/q1** fill in the file **city.scala** so that it computes, for every **state abbreviation** (in the dataset, Washington D.C. counts as a state), the number of cities it has, the total population in the state, and the maximum number of zip codes that appear in a city (in the state). Make sure the fields appear in that exact order. **Make sure to consult the slides often**.

- 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 Q1.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: only handle nulls for columns that are strictly necessary to get the answer.
- The bulk of the work needs to be done by the function **doCity()**. Other functions to fill in are getDF() to read in the cities data (make sure to use a schema), expectedOutput(), getTestDF(). Make sure that expectedOutput() returns a dataframe.
- You may only use DataFrames (no RDDs).
- You should not be using any joins.
- Pay careful attention to how you create test dataframes. You should not be making 1 big string for each line, instead each line should be a record.

Question 2. This question uses the various tables in the **/datasets/orders** dataset. Fill in **dataframes/q2/orders.scala** to compute, for every country, the *amount of money spent* from that country. Make sure the fields appear in that order (and no extra fields in the output).

- Your code should be runnable with spark-submit.
- You must include all the necessary files for compiling the code.
- The bulk of the work needs to be done by the function **doOrders()**. You should also fill in the import statements. You need to fill in getDF() – it should return tuple of 3 dataframes (not 1). The dataframes should be for customers, orders, and items (in that order). **Always use a schema**. getTestDF() should return test versions of those dataframes. expectedOutput() should return the dataframe you should get if the input was the test dataframe.
- Nulls: remember that one customer has a missing customer id. You should fill it in with the customer id "blank" using the fill function (`mydf.na.fill(value-to-replace-null, Seq(list-of-columns-where-you-want-to-fill-nulls))`)
- No rdds at all. Use only dataframes.