Lab 3: Hive and TFIDF
Note, this is a two-week lab

Part 1: Hive for On-Time Airline Tracking [5 pts]

Do some airlines have better average on-time performance than others?
You and Hive will find out!

In this exercise you will produce a text file, `worst-average-arrival-delay.txt` which will have two fields, one the airline name, and the other the average arrival delay for that airline. The records will be sorted by average arrival delay in descending order.

Your script will be named `create-worst-average-arrival-delay-file`, and here is sample output:

```
# create-worst-average-arrival-delay-file
# head -n 2 worst-average-arrival-delay.txt

Spirit Air Lines,18.634635101127785
JetBlue Airways,17.283021432779012
```

The airline database has been pre-loaded into the Docker image (use image label `lab3`). There are two data tables in the airlines database you will need:

- `On_Time_On_Time_Performance_2016_1` – has the airline code (`AirlineID`), and the number of minutes delay in arrival (`ArrDelayMinutes`). You do not need any other columns from this table.
- `L_AIRLINE_ID`, which has the airline code (`Code`) and the airline name (`Description`). Note the `Description` column contains the airline name, then a colon, then the airline's code. For example,
  `Cochise Airlines Inc.: COC.`
  In your final table you want the airline name only, not the code.

Take the following steps:

- Import those two tables from the 'airlines' database into Hive, using a shell script `import-airline-tables` which will have calls to Sqoop to import the tables.
- Use a Hive command to create a new Hive table `AirlineAvgArrDelayMinutes`. You create the table using a Hive query that joins those two tables, groups by airline, computes the average delay in minutes, fixes the airline name, and sorts the rows by `AvgArrDelayMinutes` in descending order. Your table must be named `AirlineAvgArrDelayMinutes` and must have columns `AirlineName` and

`AvgArrDelayMinutes`. This Hive command must be stored in a file `create-joined-table.hive`.

- Use a Hive command to export this table in CSV format to your local filesystem. You will export the table to a directory `average-delay-table` in your current directory. After the export you will see the `average-delay-table` directory, and it will contain one or more "part files" like those generated by Hadoop in HDFS, except they will have different name(s). This Hive command must be stored in a file name `export-table.hive`.
- Use a shell command to move the "Hive part file(s)" to the file ./worst-arrival-delay.txt. One last shell command deletes the directory `average-delay-table`

Putting everything together, your script file `create-worst-average-arrival-delay-file` will look like this:

```
#!/bin/bash
import-airline-tables
hive < create-joined-table.hive
hive < export-table.hive
cat average-delay-table/0* > ./worst-arrival-delay.txt
rm -rf average-delay-table
```

For this part of the lab, put your solution files in a directory `airline-arrival`. The directory should contain:

- Script `create-worst-average-arrival-delay-file`
- Script `import-airline-tables`
- Hive command files `create-joined-table.hive and export-table.hive`

Part 2: Text Processing / TF-IDF [15 pts]

Overview

Over the course of the quarter, you will build a simple search engine, where you will use Hadoop/Spark systems to index the documents and a NoSQL data store to store the indexed documents.

In this lab, we will do the first part of document processing: computing a `TF-IDF` measure for the terms in the document corpus (already in HDFS). Notes from Week 3 introduced you to the concept of TF-IDF, and also outlined how we will approach the calculation.

This lab will implement a (Streaming) Hadoop pipeline to compute TFIDF.

- Input – a directory in HDFS containing a document corpus

- Output – triples of the form `(term, document_id, tfidf_number)`, where the tfidf_number is a measurement of how "important" the term is in the specified document.

It is very common to solve problems using a sequence of MapReduce steps. Rarely can you solve a problem using just one MapReduce program. In each sequence, one step produces (key, value) tuples that are used as the input to the next step. For computing TFIDF we will use these steps

- Step 1: doc/term counts
  This phase is essentially a word count, but remember you need to convert a word to a term. To convert to a term, you must use the code used in the Week 2 Demo example for avg-word-length-by-letter. Second complication is that for simple word count the MapReduce key is just the term, but in this case the key is a pair (document_id, term). So we need to get a document_id, and also a "composite key" for our mapper. For the document_id we will use the *basename* of the file containing the document. For example, the file containing the text for Shakespere's Hamlet might be in an HDFS file named /data/textcorpora/shakespere-hamlet.txt, and we will use "shakespere-hamlet" as the ID for that document. You can get the file path basename for the document in the MapReduce code with the following line:

  ```
  import os
  docid = os.path.splitext(
              os.path.basename(
                 os.getenv('map_input_file')))[0]
  ```

  In this step, for your MapReduce key, use the string "+" to send to the reducer, so if your mapper is processing the document with ID "shakespeare-hamlet" and is processing the term "father", it would emit the key "shakespere-hamlet+father". Sample output for this and the other steps is below.

- Step 2: term count per document
  This step reads the document corpus again and just calculates the number of terms per document. It is essentially the word count pattern, except (a) the doc_id is passed to the reducer, and (b) words in the input are converted to terms before being sent to the reducer. Output for this step is a stream of tuples of the form (doc_id, term_count) and examples are below.

- Step 3: split key from Step 1
  In Step 1, we needed a composite key with both `doc_id` and `term`. However, to compute Term Frequency we need to join the doc/term count in Step 1 with the document count in Step 2. So this Step 3 will be a map-only job that will simply take the tuples from Step 1 of the form (<doc_id>+<term>, count) and produce three-tuples of the form (doc_id, term, count). By map-only job I mean that the reducer just writes out the tuples it gets from the mapper without modifying them.

- Step 4: compute Document Frequency (DF)
  This MapReduce job will go back to the documents, but this time the mapper will construct tuples of the form (term, doc_id) and the reducer will emit tuples of the form (term, unique_doc_id_count).
  Hint: the mapper will look the same as mappers you have already seen, but the reducer's job is different. Rather than summing a quantity or taking max from a stream of numbers, it needs to "remember" the set of unique values it has seen for each key. The code for doing so is very short and simple, but it will require some thought.

- Step 5: compute TF-IDF
  You have three streams (HDFS directories) available:

  - (doc_id, term_count) from Step 2
  - (doc_id, term, count) from Step 3
  - (term, unique_doc_id_count) from Step 4

  The goal is a stream (doc_id, term, tf-idf). You can compute `TF-IDF` from these three inputs. You will use Hive to first build three table abstractions over these three inputs, then you can compute your desired `TF-IDF` tuples using a single select statement. Since TF-IDF values tend to be very small, multiply the computed TF-IDF value by 1000000.

  You can then have Hive create a table based on this select, and that will make the table appear in HDFS.

Your full solution is a shell script `run-tfidf` which will

- Move the documents into HDFS
- Run each of the steps of the TF-IDF process
- Move the final output to HDFS in a directory /output/5-tfidf

The components of your solution will be Hadoop Streaming jobs that have these names

- 1-term-count, 2-term-count-document, 3-split-doc-term, 4-df
- The last step will run Hive on a hive script that must be named 5-tfidf.hive

Example Output / Hint

Here is some sample output from the first four MapReduce jobs. Your output might be in a different order.

```
# hdfs dfs -cat /output/1-term-count/* | head -n 5
austen-emma+a 3073
austen-emma+abbey 23
austen-emma+abbeymill 7
```

```
austen-emma+abbeyoh 1
austen-emma+abbots 1

# hdfs dfs -cat /output/2-term-count-document/* | head -n 5
austen-emma 158128
austen-persuasion 83259
austen-sense 118620
bible-kjv 790029
blake-poems 6817

# hdfs dfs -cat /output/3-split-doc-term/* | head -n 5
austen-emma freak 1
austen-emma free 5
austen-emma freed 2
austen-emma freedom 3
austen-emma freeze 1

# hdfs dfs -cat /output/4-df/* | head -n 5
a 18
aaron 2
aaronites 1
aarons 2
ab 1

# hdfs dfs -cat /output/5-tfidf/* | head -n 5
austen-emma     abbeymill       44.26793483759992
austen-emma     abbeyoh 6.323990691085703
austen-emma     abbots  6.323990691085703
austen-emma     abdys   6.323990691085703
austen-emma     abhor   1.5809976727714257
```

For this part, submit:

- a folder `tfidf` that contains the following subfolders and files:
    - folder `1-term-count`: contains your MapReduce code for step 1
    - folder `2-term-count-document`: contains your MapReduce code for step 2
    - folder `3-split-key`: contains your MapReduce code for step 3
    - folder `4-df`: contains your MapReduce code for step 4
    - file `5-tfidf.hive`: contains the Hive program that computes TF-IDF
    - script file `run-tfidf`

Submission

A `zip` file with:

1. The folder airline-arrival
2. The folder tf-idf
3. A retrospective report in a file `retrospective.pdf` – a reflection on the assignment, with the following components
    a. Your name
    b. How much time you spent on the assignment
    c. If parts of the assignments are not fully working, which parts and what the problem(s) are
    d. Were there aspects of the assignment that were particularly challenging? Particularly confusing?
    e. What were the main learning take-aways from this lab – that is, did it introduce particular concepts or techniques that might help you as an analyst or engineer in the future?