LAB DISTRIBUTED DATA ANALYTICS I
TUTORIAL V

Juan Fernando Espinosa Reinoso
303158
Group 1: Monday Tutorial

Exercise 1: Preparing your Hadoop infrastructure (Warm up exercise)

1.2 Basic Hadoop operations

- Check hadoop version

```
[maria_dev@sandbox ~]$hadoop version
Hadoop 2.7.3.2.5.0.0-1245
Subversion git@github.com:hortonworks/hadoop.git -r cb6e514b14fb60e9995e5ad9543315cd404b4e59
Compiled by jenkins on 2016-08-26T00:55Z
Compiled with protoc 2.5.0
From source with checksum eba8ae32a1d8bb736a829d9dc18dddc2
This command was run using /usr/hdp/2.5.0.0-1245/hadoop/hadoop-common-2.7.3.2.5.0.0-1245.jar
```

- List files in HDFS: hadoop fs -ls /

```
[maria_dev@sandbox ~]$ hadoop fs -ls /
Found 13 items
drwxr-xr-x   - raj_ops hdfs          0 2020-06-12 14:30 /DDA
drwxrwxrwx   - yarn    hadoop        0 2016-10-25 08:10 /app-logs
drwxr-xr-x   - hdfs    hdfs          0 2016-10-25 07:54 /apps
drwxr-xr-x   - yarn    hadoop        0 2016-10-25 07:48 /ats
drwxr-xr-x   - hdfs    hdfs          0 2016-10-25 08:01 /demo
drwxr-xr-x   - hdfs    hdfs          0 2016-10-25 07:48 /hdp
drwxr-xr-x   - mapred  hdfs          0 2016-10-25 07:48 /mapred
drwxrwxrwx   - mapred  hadoop        0 2016-10-25 07:48 /mr-history
drwxr-xr-x   - hdfs    hdfs          0 2016-10-25 07:47 /ranger
drwxrwxrwx   - spark   hadoop        0 2020-06-12 23:55 /spark-history
drwxrwxrwx   - spark   hadoop        0 2016-10-25 08:14 /spark2-history
drwxrwxrwx   - hdfs    hdfs          0 2016-10-25 08:11 /tmp
drwxr-xr-x   - hdfs    hdfs          0 2016-10-25 08:11 /user
```

- Create a hadoopdemo directory: hadoop fs -mkdir /hadoopdemo

```
[maria_dev@sandbox ~]$ hadoop fs -mkdir /hadoopdemo
```

```
drwxr-xr-x   - raj_ops   hdfs            0 2020-06-12 14:30 /DDA
drwxrwxrwx   - yarn      hadoop          0 2016-10-25 08:10 /app-logs
drwxr-xr-x   - hdfs      hdfs            0 2016-10-25 07:54 /apps
drwxr-xr-x   - yarn      hadoop          0 2016-10-25 07:48 /ats
drwxr-xr-x   - hdfs      hdfs            0 2016-10-25 08:01 /demo
drwxr-xr-x   - maria_dev hdfs            0 2020-06-13 00:06 /hadoopdemo
drwxr-xr-x   - hdfs      hdfs            0 2016-10-25 07:48 /hdp
drwxr-xr-x   - mapred    hdfs            0 2016-10-25 07:48 /mapred
drwxrwxrwx   - mapred    hadoop          0 2016-10-25 07:48 /mr-history
drwxr-xr-x   - hdfs      hdfs            0 2016-10-25 07:47 /ranger
drwxrwxrwx   - spark     hadoop          0 2020-06-13 00:07 /spark-history
drwxrwxrwx   - spark     hadoop          0 2016-10-25 08:14 /spark2-history
drwxrwxrwx   - hdfs      hdfs            0 2016-10-25 08:11 /tmp
drwxr-xr-x   - hdfs      hdfs            0 2016-10-25 08:11 /user
```

- Create several sub-directories nested in hadoopdemo, e.g. text files, raw data

```
[maria_dev@sandbox ~]$ hadoop fs -ls /hadoopdemo
Found 2 items
drwxr-xr-x   - maria_dev hdfs          0 2020-06-13 00:41 /hadoopdemo/raw_data
drwxr-xr-x   - maria_dev hdfs          0 2020-06-13 00:42 /hadoopdemo/text_files
```

- Transfer and store data from the Local System to Hadoop

```
[maria_dev@sandbox ~]$ hadoop fs -ls /hadoopdemo/text_files
Found 1 items
-rw-r--r--   3 maria_dev hdfs    1419393 2020-06-13 01:10 /hadoopdemo/text_files/test_hadoop.txt
```

- Check directories and files using Hadoop User Interface

Browse Directory

/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	raj_ops	hdfs	0 B	12/6/2020 9:30:50	0	0 B	DDA
drwxrwxrwx	yarn	hadoop	0 B	25/10/2016 3:10:48	0	0 B	app-logs
drwxr-xr-x	hdfs	hdfs	0 B	25/10/2016 2:54:17	0	0 B	apps
drwxr-xr-x	yarn	hadoop	0 B	25/10/2016 2:48:04	0	0 B	ats
drwxr-xr-x	hdfs	hdfs	0 B	25/10/2016 3:01:20	0	0 B	demo
drwxr-xr-x	maria_dev	hdfs	0 B	12/6/2020 19:42:40	0	0 B	hadoopdemo
drwxr-xr-x	hdfs	hdfs	0 B	25/10/2016 2:48:14	0	0 B	hdp
drwxr-xr-x	mapred	hdfs	0 B	25/10/2016 2:48:13	0	0 B	mapred
drwxrwxrwx	mapred	hadoop	0 B	25/10/2016 2:48:19	0	0 B	mr-history
drwxr-xr-x	hdfs	hdfs	0 B	25/10/2016 2:47:59	0	0 B	ranger
drwxrwxrwx	spark	hadoop	0 B	12/6/2020 20:13:57	0	0 B	spark-history
drwxrwxrwx	spark	hadoop	0 B	25/10/2016 3:14:12	0	0 B	spark2-history
drwxrwxrwx	hdfs	hdfs	0 B	25/10/2016 3:11:16	0	0 B	tmp
drwxr-xr-x	hdfs	hdfs	0 B	25/10/2016 3:11:23	0	0 B	user

- Remove the sub-directory hadoop text_files

```
[maria_dev@sandbox ~]$ hadoop fs -rm -r /hadoopdemo/text files
rm: `/hadoopdemo/text': No such file or directory
rm: `files': No such file or directory
[maria_dev@sandbox ~]$ hadoop fs -rm -r /hadoopdemo/text_files
20/06/13 01:17:19 INFO fs.TrashPolicyDefault: Moved: 'hdfs://sandbox.hortonworks.com:8020/hadoopdemo/text_files' to trash a
rtonworks.com:8020/user/maria_dev/.Trash/Current/hadoopdemo/text_files
[maria_dev@sandbox ~]$ hadoop fs -ls /hadoopdemo
Found 1 items
drwxr-xr-x   - maria_dev hdfs          0 2020-06-13 00:41 /hadoopdemo/raw_data
```

- Change the content of file.txt in the local system and overwrite it in Hadoop hadoop fs -put -f file.txt /hadoopdemo/raw_data

```
Found 1 items
-rw-r--r--   3 maria_dev hdfs    1419393 2020-06-13 01:21 /hadoopdemo/raw_data/test_hadoop.txt
```

- Read the content of the file: hadoop fs -cat /hadoopdemo/raw_data/file.txt

```
[maria_dev@sandbox ~]$ hadoop fs -cat /hadoopdemo/raw_data/test_hadoop.txt
0.0740740740741 0:3 19:134 22:31 24:12 27:19 34:24 36:2 50:7 52:384 55:11 64:1 66:4 67:78 71:109 80:33 84:12 87:3 88:58 90:1 100:17 107:132 115
:30 121:105 141:95 151:45 160:1 165:13 177:1 178:4 181:1 193:12 197:1 198:3 203:455 209:1 225:7 230:4 231:12 232:2 238:4 240:17 245:30 354:61 3
55:96 357:6 359:4 361:2 363:1 368:2 369:2 370:1
0.684210526316 19:14 20:2 24:2 27:2 34:1 36:791 45:5084 47:2 50:5 52:7 57:3 61:55 62:1 67:1 68:51 71:54 80:5085 84:5 87:2 93:2 100:1 104:3 107:
67 109:1 111:1 115:3 121:65 123:1 128:3 129:1 132:3 140:2 141:2 142:1 151:9 154:789 160:3 169:2 174:2 186:1 190:1 198:1 203:8 209:4 230:1 232:3
 236:3 240:1 245:1 353:1 354:7 355:54 357:2 362:2 364:1 367:1 368:1 369:3 370:2 422:1 469:1
```

I use a small txt file from exercise 4.

- ## Hadoop fs -count

Count the number of directories in a folder or path.

```
[maria_dev@sandbox ~]$ hadoop fs -count /hadoopdemo
           2            1                716 /hadoopdemo
```

- ## Hadoop fs -du <path>

Outputs the size of files and directories in the selected path

```
[maria_dev@sandbox ~]$ hadoop fs -du /hadoopdemo
716  /hadoopdemo/raw_data
```

- ## Hadoop fs -help

prints out all the documentation of functions that can be used in the shell.

```
[maria_dev@sandbox ~]$ hadoop fs -help
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
```

1.3 Word Count Map Reduce Example

The very beginning step for this exercise is to download the **gutenberg ebook** and save it as a txt file.

1.3.1 Create a directory for all the process in root of the hadoop environment

By using **hadoop fs -mkdir** I created a folder called wordCount in the root directory. In the folder will be stored the input data and the jar file (code for the wordCount algorithm)

drwxr-xr-x	maria_dev	hdfs	0 B	13/6/2020 19:26:58	0	0 B	wordCount

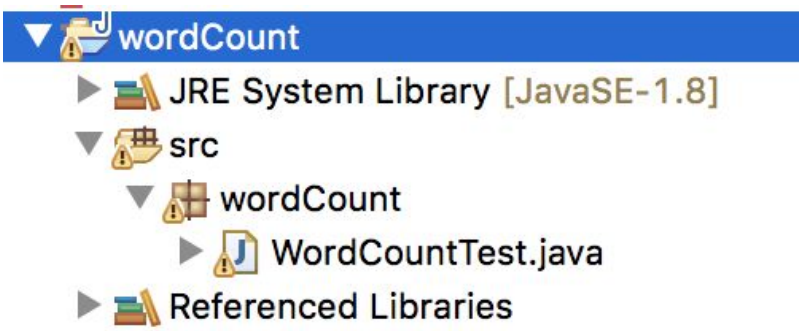
1.3.2 Create a subdirectory inside the wordCount folder

I created a subdirectory called **Input** where the **gutenberg ebook** is stored and from which it will be called in the later steps of the process.

drwxr-xr-x	maria_dev	hdfs	0 B	13/6/2020 19:26:11	0	0 B	input

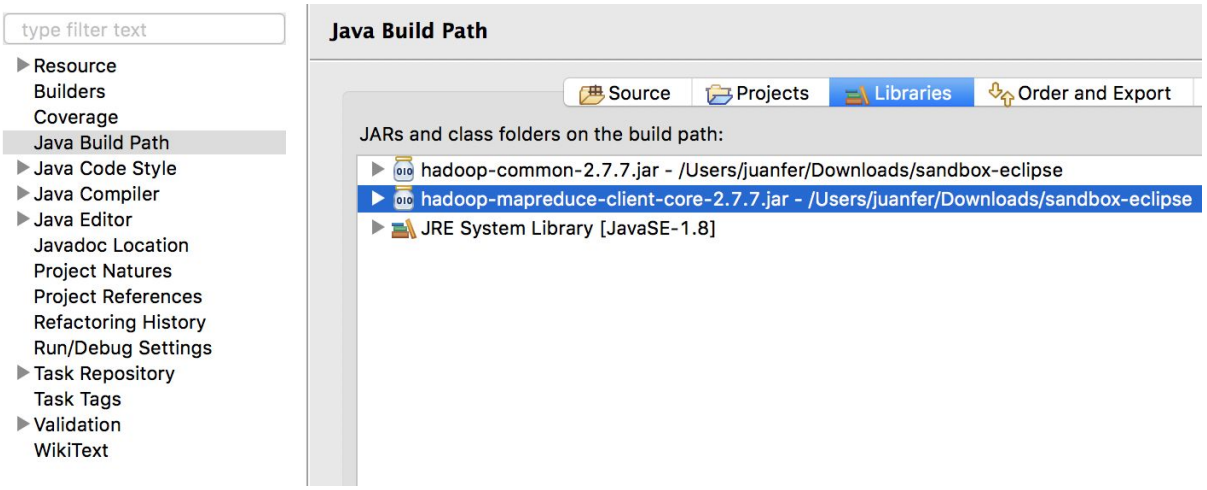
1.3.3 Create a project in Eclipse IDE - java

I created a project called wordCount. In this project is stored the wordCountTest.java file I downloaded from [1]. As a matter of fact of the Eclipse project the name of the java file has to be exactly the same as the main function inside it. Hence, the name as the file downloaded from the source is kept.



1.3.4 Add to the project: Hadoop common and hadoop mapreduce jar files

To do so, in the wordCount project head to settings, Java build path, and select Libraries. The next step is to add the external jars. Pick the 2 files mentioned in the process title and save it.



1.3.5 Export the project as a jar file

Moving forward, I saved the project as a jar file, which is required for **mapReduce**, containing:

- Hadoop basic libs
- Hadoop main code containing the classes: Count Mapper & Count Reducer

1.3.6 Save the jar file in the same directory where the input folder is located in hadoop environment

By using **hadoop fs -put** I copy the file from the local folder to the required folder in the hadoop environment. The file name is **textcount.jar**

1.3.7 Run the code using the input file and save the result in a new folder called output

Once the wordCount directory contains the input folder, the jar file containing the algorithm as well as the common mapreduce jar files, we can proceed to run the model and get an output.

We go to the folder containing all the files

```
[root@sandbox ~]# hadoop fs -ls /wordCount
Found 5 items
drwxr-xr-x   - maria_dev hdfs          0 2020-06-13 14:43 /wordCount/Classes
-rwxrwxrwx   3 maria_dev hdfs       2148 2020-06-13 14:44 /wordCount/WordCount.java
-rw-r--r--   3 admin     hdfs       2377 2020-06-13 23:28 /wordCount/documentcount.jar
drwxr-xr-x   - maria_dev hdfs          0 2020-06-13 04:39 /wordCount/input
-rw-r--r--   3 admin     hdfs       3366 2020-06-13 23:57 /wordCount/textcounter.jar
[root@sandbox ~]# hadoop fs -get /wordCount/textcounter.jar
```

The following step is to run the algorithm through the input data and **store** the result in a new folder inside that directory called **output**

```
  Answer various questions about <path>, with result via exit status.
    -d  return 0 if <path> is a directory.
[root@sandbox ~]# hadoop jar /textcounter.jar /wordCount/input  /wordCount/output
```

RED: calling the jar file containing the algorithm to run
GREEN: The path where the input data is stored, in this case the **gutenberg ebook**
YELLOW: Finally I stated a new directory where the output file of the algorithm will be stored.

1.3.8 Run the output files from the folder and check the results

As a final step, first it is important to check if effectively the folder with the output has been created.

---- inside the folder ----

_SUCCESS	0.1 kB	2020-06-13 19:02	root	hdfs	-rw-r--r--
part-r-00000	0.1 kB	2020-06-13 19:02	root	hdfs	-rw-r--r--

It is possible to appreciate that the output folder with the inputs inside has been created accordingly.

Finally, running the output to check the answers by executing the following command:

```
[root@sandbox ~]# hadoop fs -ls /wordCount/output
Found 2 items
-rw-r--r--   1 root hdfs          0 2020-06-14 00:02 /wordCount/output/_SUCCESS
-rw-r--r--   1 root hdfs         56 2020-06-14 00:02 /wordCount/output/part-r-00000
[root@sandbox ~]# hadoop fs -cat /wordCount/output/part-r-00000
```

The outputs gotten by applying the above-mentioned steps follows the manner:

```
STRONGER,       1
STRONGER.'      2
STRUCK  19
STRUCK, 1
STRUCK; 1
STRUGGLING      1
STUCK   13
STUDENT 4
STUDENT,        1
STUDENT'S       1
STUDIED 1
STUDY   1
STUDY,  1
STUFFED 1
STUMBLES        1
STUMP   1
STUPID  3
STUPID, 2
STY.    1
SUBDUE  1
SUBJECT 1
SUBSCRIBE       1
SUCCEED 1
SUCCEED,        1
SUCCEEDED       1
SUCCESSFULLY.   1
SUCCESSIVE      1
SUCCESSOR.      1
SUCH    81
SUCK    1
SUCKLE  1
SUDDEN  7
SUDDEN, 1
```

Conclusion: The algorithm has been executed successfully. Hence, Hadoop is correctly installed and ready to use.

Note: The algorithm has some mistakes because it has not erased stopwords from the text.

Sources: [1], [2]

Exercise 2: Analysis of Airport efficiency with Map Reduce (10 points)

1.1 Computing Max, Min and Average in a single Mapreduce run.

The first step as it is easy to infer is to download the dataset, and create a directory on Hadoop saving the file in there. Next, I proceed to code the main Mapper and Reducer.

- The mapping goal is just to merge together into a (key, value) structure all the variables that are required. For this purpose, the key is the **ORIGIN** and the values are going to be the **Departure Delay.**

The code, which is mostly used to read text lines, calls first the sys.stdin which helps to read the file. Next, stripping and splitting in a comma separated values from each “line” in the csv file is applied. Then a small idea is implemented: Since it is required to use only 2 columns a condition is inserted and from which is extracted the Origin ID and the departure delay values. The second last step is made in order to avoid the **empty values** the departue delay field has: Skip those rows so that it won’t affect the final output. Finally the (key,value) output is ready to be used by the **reducer.**

```
#!/usr/bin/python
import sys


for line in sys.stdin:
    line = line.strip()
    #line = line.splitlines()
    line = line.split(",")

    if len(line) >=3:
        airport = line[3]
        delay_dep = line[6]
        if delay_dep == "" or delay_dep == ' ':
            continue
        print('%s\t%s' % (airport, delay_dep))
```

- The reducer main goal is to take the (key,values) gotten by the mapper, group similar keys and perform some operations to the values.

As it is required in the exercise, the reducer goal is to take the maximum value from the values of all origin airports, the average, as well as the minimum.

First, initialization with a for loop: sys.stdin. Next, in order to avoid taking into account the first row I skipped it. Next, performing a separation between the key and the value so that it is possible to work with the information.

As dictionaries are easy to process by the machine, and easy to iterate through, I created a dictionary that is saving each key as origin of the flight and saving in an array with all the values for each specific **origin.** Finally, from the dictionary I iterate through the keys and perform the required operations with the values of those keys.

```
#!/usr/bin/python
import sys


airport_delays = {}

for i, line in enumerate(sys.stdin):
    if i == 0:
        continue
    line = line.strip()
    (airport,delay_dep) = line.split('\t')

    if airport in airport_delays:
        airport_delays[airport].append(float(delay_dep))
    else:
        airport_delays[airport] = []
        airport_delays[airport].append(float(delay_dep))

for airport in airport_delays.keys():
    max_delay = max(airport_delays[airport])
    avg_delay = sum(airport_delays[airport])*1.0 / len(airport_delays[airport])
    min_delay = min(airport_delays[airport])
    print('Maximum',(airport, max_delay))
    print('Average', (airport, avg_delay))
    print('Minimum', (airport, min_delay))
```

Now as we want to run into hadoop the following steps were applied:

- import the mapper and reducer to hdfs files and making sure it is possible to execute the files
- Next, as it is necessary to trigger the hadoop streaming to be able to run custom mapper and reducer, I found the ideal path by inserting the following command : **find / -name 'hadoop-streaming*.jar'**
- Finally, once the mapper are uploaded and executable, the hadoop streaming has been identified, it is time to run the code considering the initial directory where the input data is saved and specify an outpur folder where the answer will be stored
- **hadoop jar /usr/hdp/2.5.0.0-1245/hadoop-mapreduce/hadoop-streaming-2.7.3.2.5.0.0-1245.jar -input /mapred/system/airport-efficiency.csv -output /mapred/system/output -mapper ./mapper.py -reducer ./reducer.py -file ./mapper.py -file ./reducer.py**

```
[raj_ops@sandbox ~]$ hadoop jar /usr/hdp/2.5.0.0-1245/hadoop-mapreduce/hadoop-streaming-2.7.3.2.5.0.0-1245.jar -input /mapred/system/airport-
efficiency.csv -output /mapred/system/output -mapper ./mapper.py -reducer ./reducer.py -file ./mapper.py -file ./reducer.py
```

Finally, the output looks as follow:

```
('Maximum', ('"OTZ"', 154.0))
('Average', ('"OTZ"', 6.982142857142857))
('Minimum', ('"OTZ"', -26.0))
('Maximum', ('"DAB"', 462.0))
('Average', ('"DAB"', 11.68))
('Minimum', ('"DAB"', -19.0))
('Maximum', ('"TXK"', 215.0))
('Average', ('"TXK"', 10.541666666666666))
('Minimum', ('"TXK"', -21.0))
('Maximum', ('"ACT"', 202.0))
('Average', ('"ACT"', 13.744897959183673))
('Minimum', ('"ACT"', -17.0))
('Maximum', ('"ONT"', 352.0))
('Average', ('"ONT"', 12.89056831922612))
('Minimum', ('"ONT"', -21.0))
('Maximum', ('"CLL"', 418.0))
('Average', ('"CLL"', 10.009345794392523))
('Minimum', ('"CLL"', -20.0))
('Maximum', ('"MFR"', 917.0))
('Average', ('"MFR"', 26.899497487437184))
('Minimum', ('"MFR"', -19.0))
('Maximum', ('"SAT"', 579.0))
('Average', ('"SAT"', 9.568044961862705))
('Minimum', ('"SAT"', -19.0))
('Maximum', ('"DSM"', 705.0))
('Average', ('"DSM"', 10.604135893648449))
('Minimum', ('"DSM"', -18.0))
('Maximum', ('"FLG"', 123.0))
('Average', ('"FLG"', 0.2625))
('Minimum', ('"FLG"', -17.0))
('Maximum', ('"MFE"', 327.0))
```

sources= [3], [4], [5]

1.2 Find the top 20 airports regarding their average Arrival delay

For this exercise, as the mapper and the input dataset remain unchanged, I am going to focus on the reducer.

The same process applied before keeps the same until we get the dictionary (keys, array of values). Next, as the goal is to find the top 10 airports a for loop is created iterating through the keys: We get the average value for **Arrival delay.** The next step is to save as a [Airport, avg value] list of list from which later extract the index of the 10 **minimum** value and finally print those 10 airports with the respective value for arrival delays. The code and result are presented below:

```python
#!/usr/bin/python
import sys
import numpy as np

airport_delays = {}
avg_delay_airports = []
for i, line in enumerate(sys.stdin):
    if i == 0:
        continue
    line = line.strip()
    (airport,delay_dep) = line.split('\t')

    if airport in airport_delays:
        airport_delays[airport].append(float(delay_dep))
    else:
        airport_delays[airport] = []
        airport_delays[airport].append(float(delay_dep))

for i, airport in enumerate(airport_delays.keys()):
    avg_delay = sum(airport_delays[airport])*1.0 / len(airport_delays[airport])
    avg_delay_airports.append([airport, avg_delay])
avg_delay_airports = np.array(avg_delay_airports)
idx = (avg_delay_airports[:,1]).argsort()[:10]
top_10 = avg_delay_airports[idx]
print('top_10',top_10)
```

OUTPUT:

```
('top_10', array([['"RSW"', '-0.0982892690513'],
       ['"CIU"', '-0.553191489362'],
       ['"JNU"', '-1.23076923077'],
       ['"CLT"', '-1.25189543963'],
       ['"GCC"', '-1.37209302326'],
       ['"MSO"', '-1.64661654135'],
       ['"PAH"', '-1.74'],
       ['"HLN"', '-1.79646017699'],
       ['"ANC"', '-1.85606060606'],
       ['"HIB"', '-11.3461538462']], dtype='|S16'))
```

Note: While running some tests something went wrong in the virtual machine in the hadoop shell. Therefore, I couldn't run more tests so the output result is gotten by printing the output in the local shell.

sources= [3], [4], [5]

References:

[1] https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox/1.html

[2] https://www.youtube.com/watch?v=T46_6_aR47Y

[3] http://rare-chiller-615.appspot.com/mr1.html

[4] https://maelfabien.github.io/bigdata/MRJobP/#hadoop-streaming

[5] community.hortonworks.com