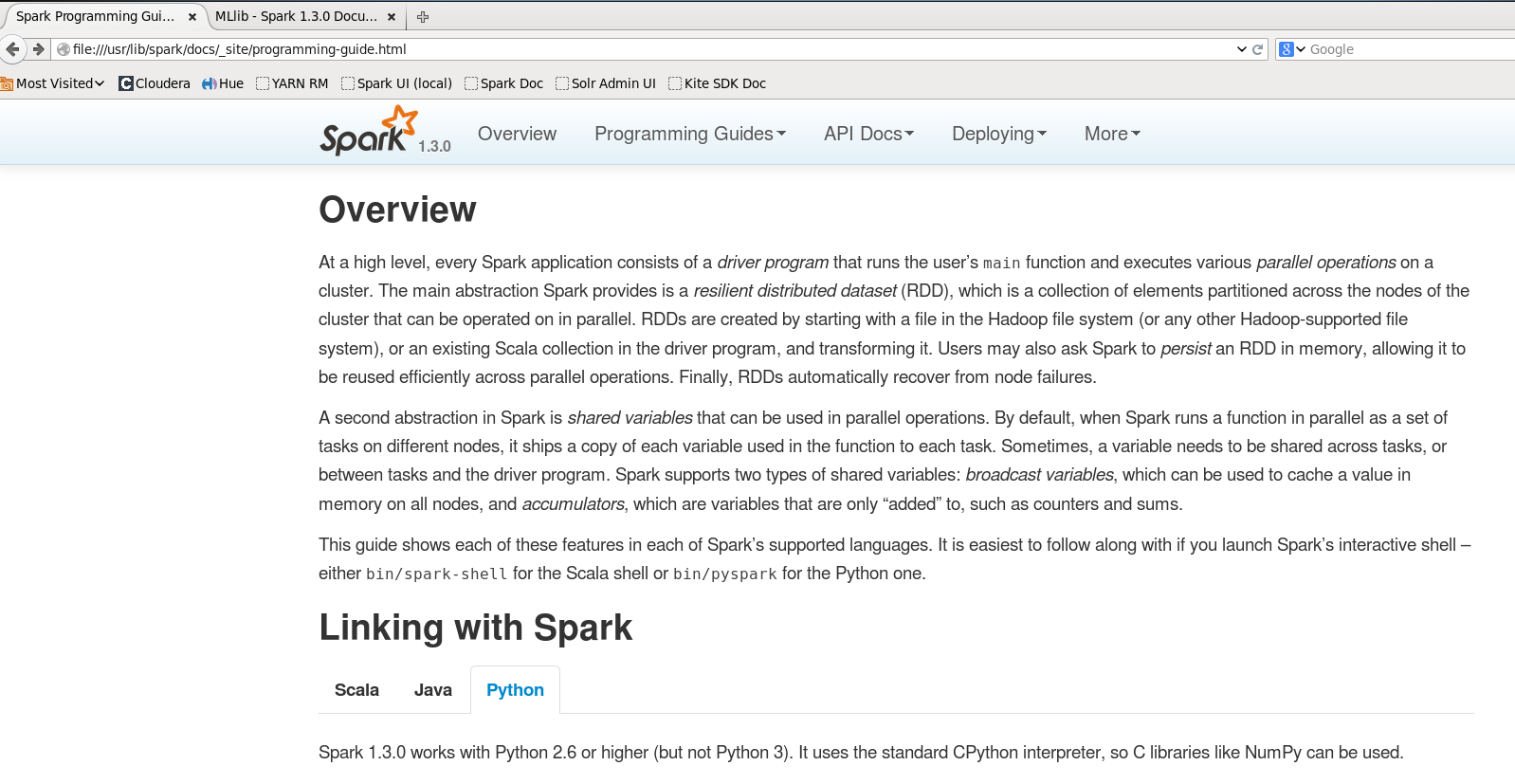
# Spark tutorial

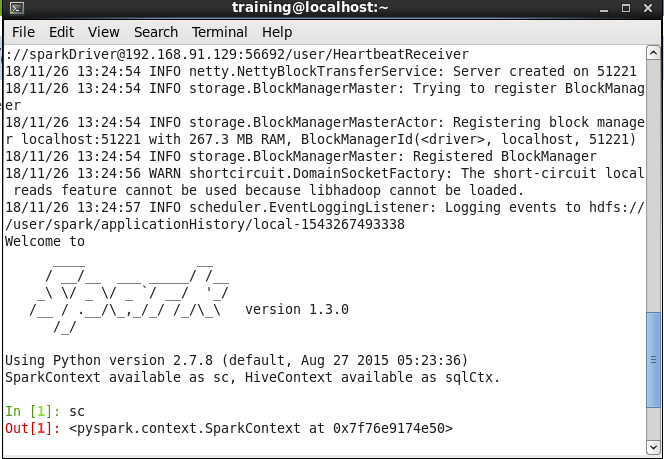
# Spark docs

Read the Spark programming guide using Mozilla on virtual machine and bookmarked the Python API page.

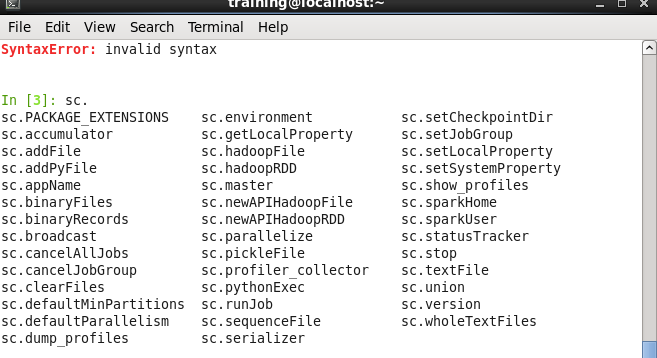


# Spark Shell RDDs

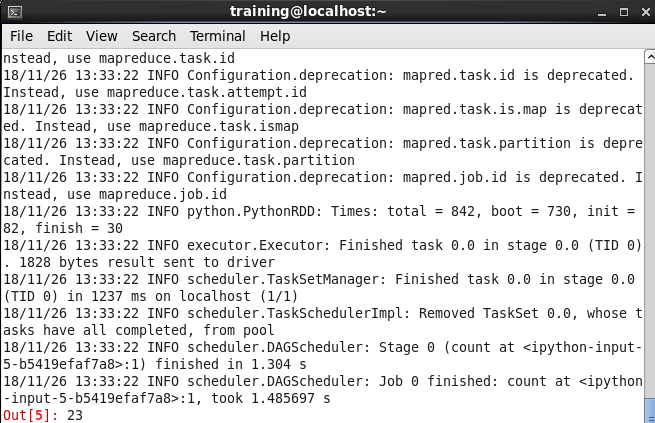
Opened python spark shell using command pyspark. Checked whether sc exists:



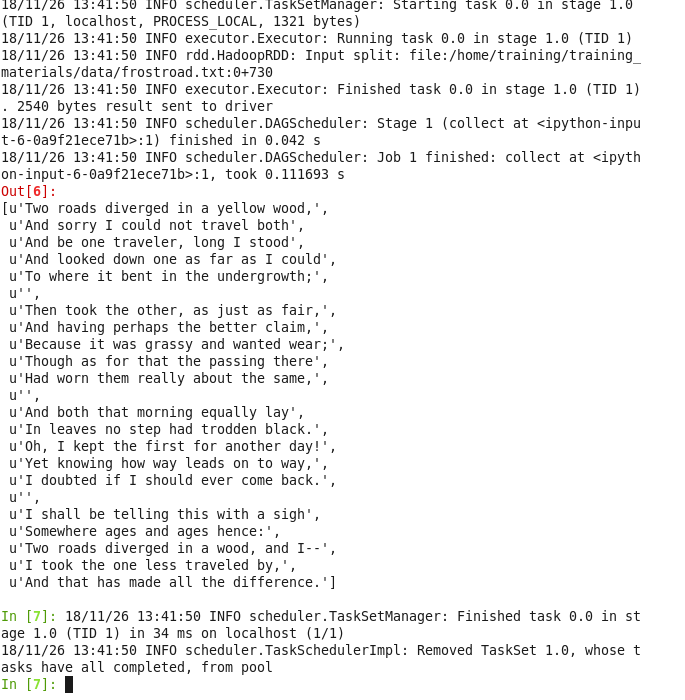
Checked command completion options of sc. :



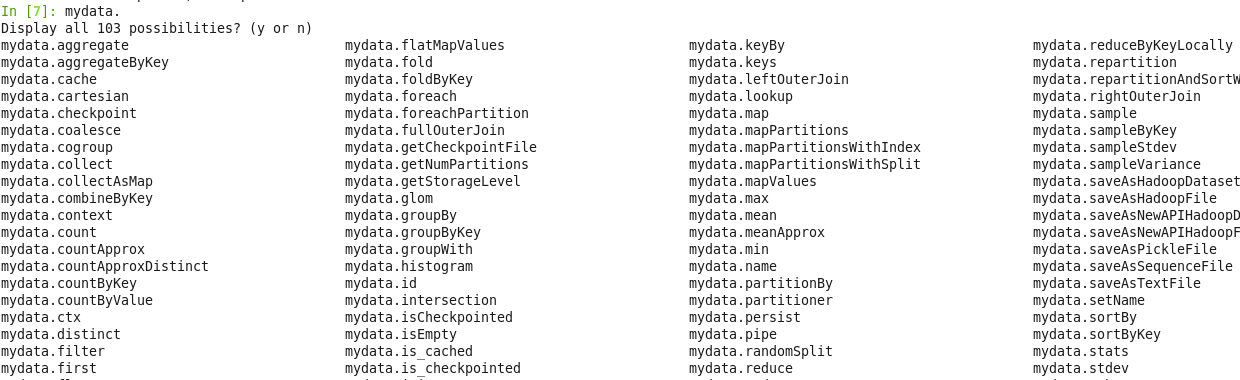
Created an RDD to read ‘Frostroad.txt’ file and performed operation of counting the number of lines so that Spark reads the RDD. This count operation materializes the RDD.



The collect operation displays the entire output:



The following are some of the possible commands we can perform on mydata, obtained using command completion:



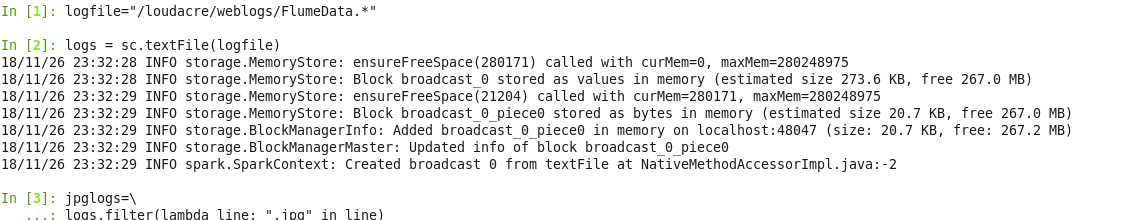
# Use RDDs

We will be using the HDFS loudacre/weblogs files that were created in earlier exercise.

The contents of the files are as follows:



We assigned a variable of the file path, so we don’t need to retype it; created an RDD from the data file containing only requests for jpg files:



Using take() command, we viewed first 10 lines of the jpglogs RDD:

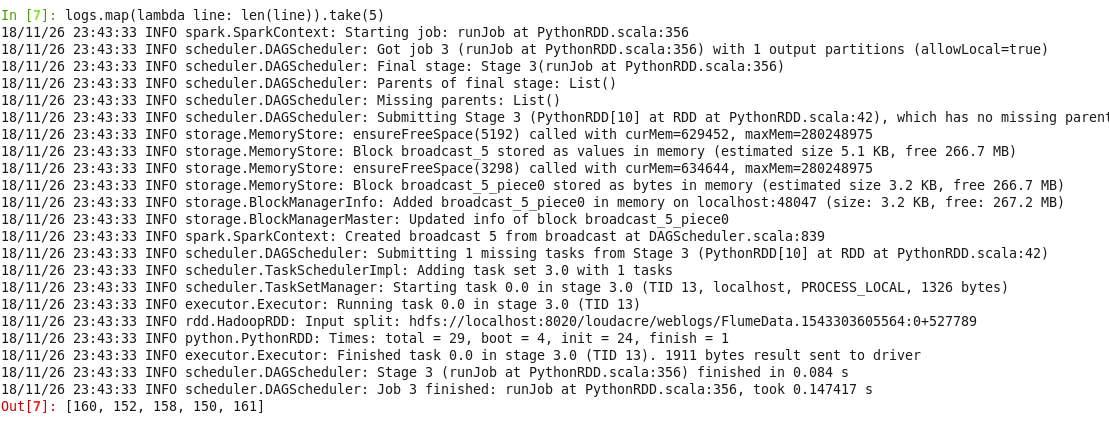


If we just wanted to count the number of lines, we can skip assigning an intermediate variable.

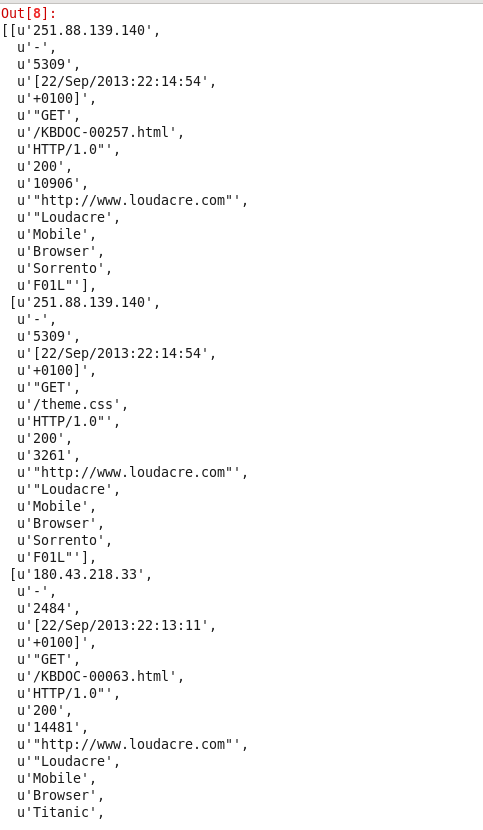
The count of number of jpg requests appears to be 2290, by using the command given in exercise.



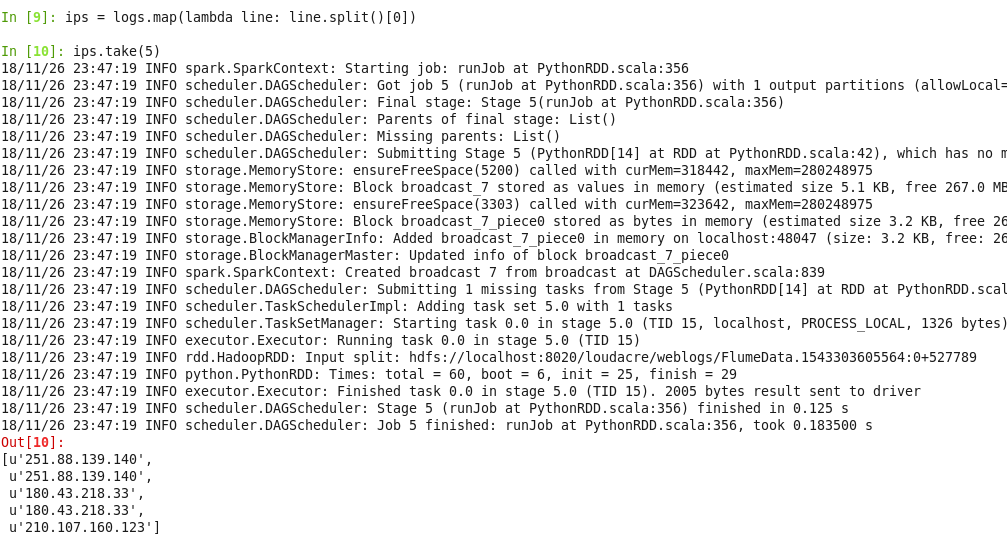
Now using the map function, we count the length of each line and display top 5 lines:



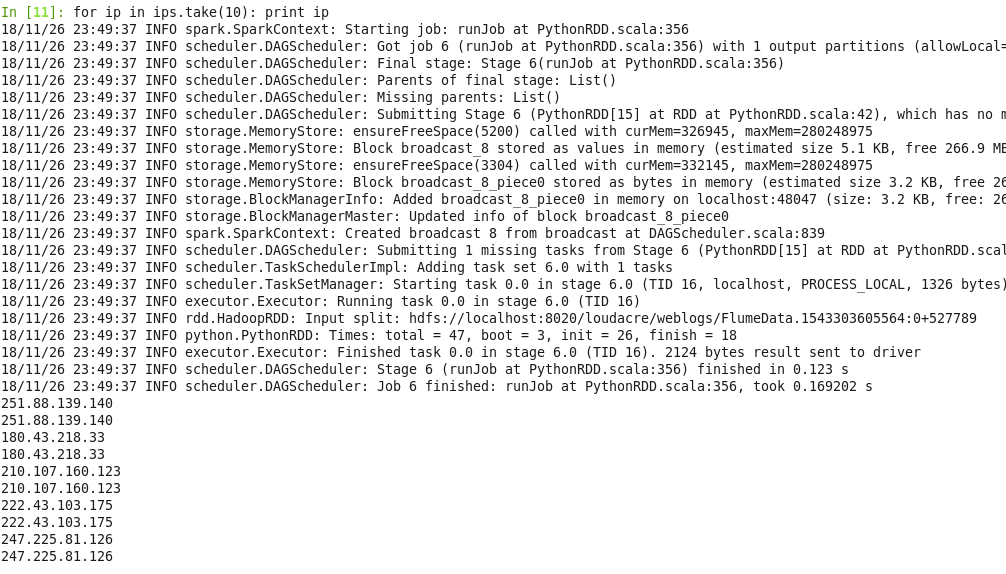
As this result is not very useful, we split the words of each line. The output looks like this:



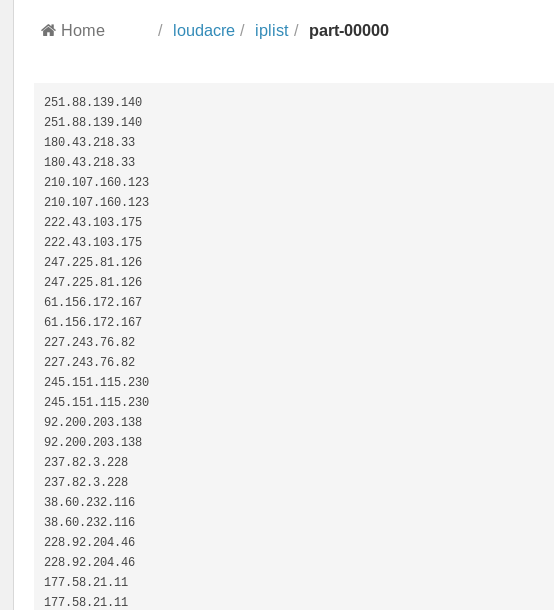
Now, we use the map function to extract IP addresses from the data. The top 5 IPs are as follows:



Then we learnt how to use for loop to iterate through the data output:



We saved the IP addresses output in a text files in the loudacre/iplist directory. The files look as follows:

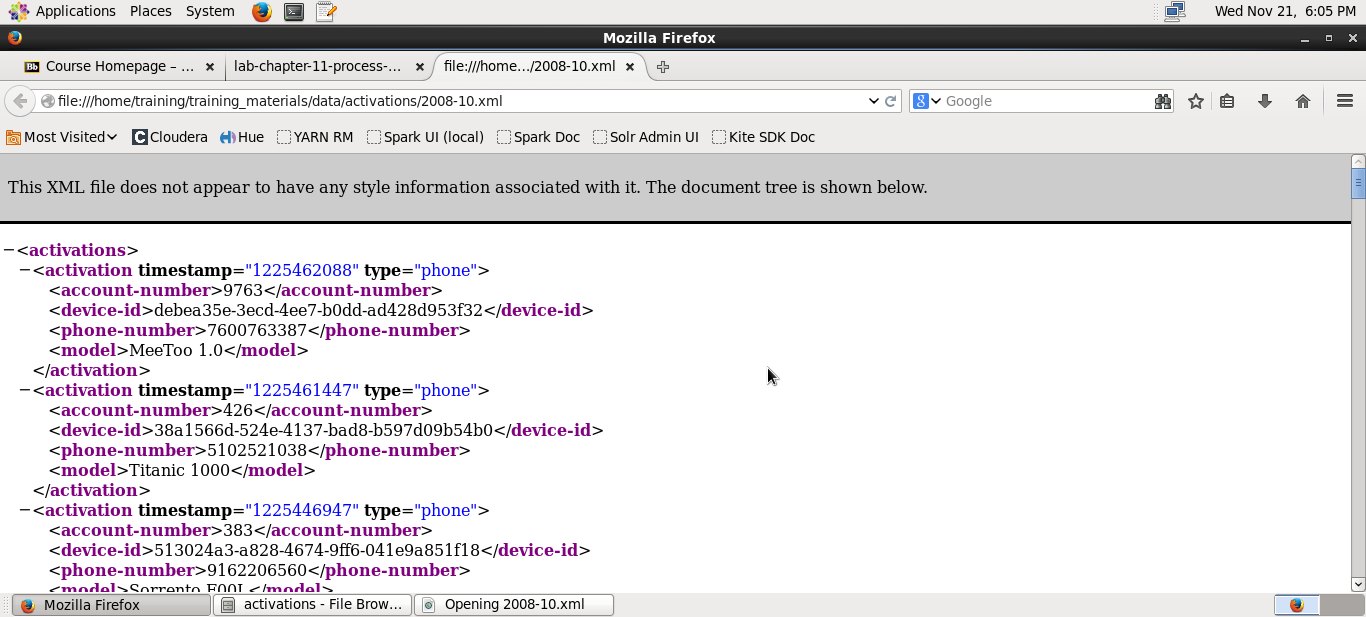


# Process with Spark

In this lab, We parse a set of activation records in XML format to extract account numbers and model names:

Reviewed the Data directory given for XML files and the format of the XML file is as follows

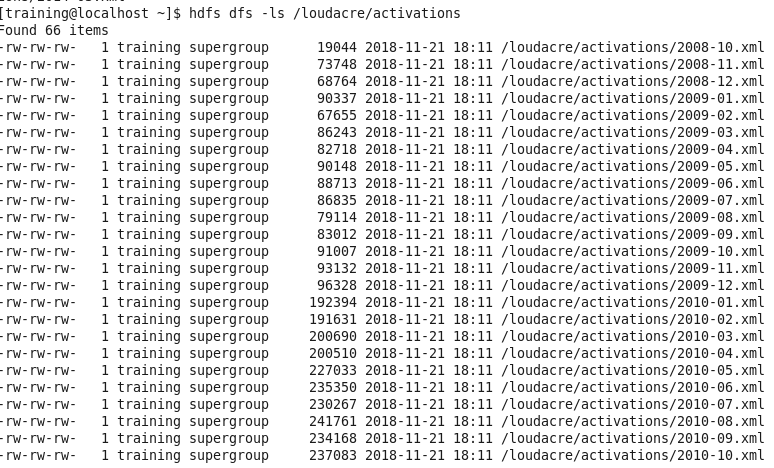
Data: /home/training/training\_materials/data/activations



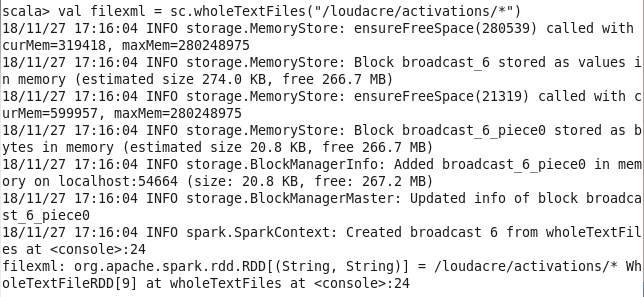
Then, we Copied the XML data files from local directory to HDFS.

In HDFS the files are stored in /loudacre/Activations and the list of directories in it are as follows.

Command: $ hdfs dfs -put /home/training/training\_materials/data/activations /loudacre



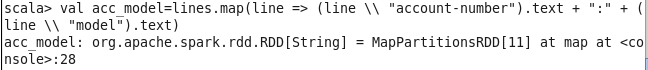
Then created an RDD which takes all the xml files from loudacre/activations directory



Using flatMap to map the contents of each file to a collection of XML records by calling the provided XML.loadString function

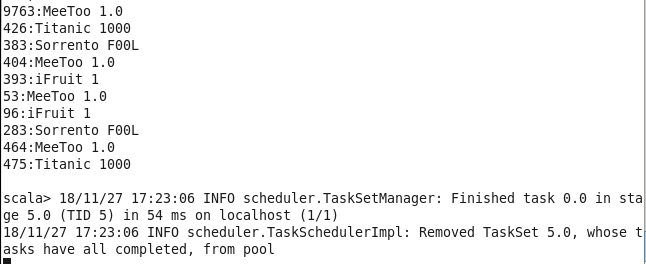


Mapping each activation record to a string in the format account-number:model.

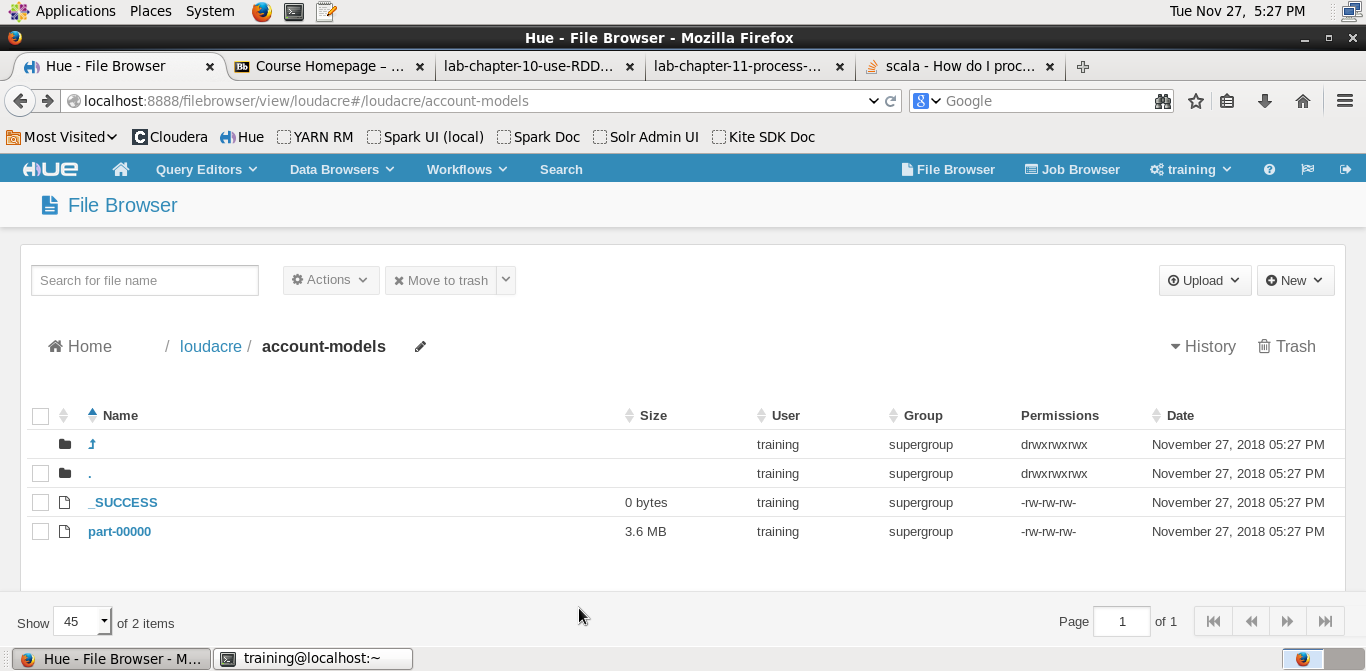


Printed the first 10 values from acc\_model

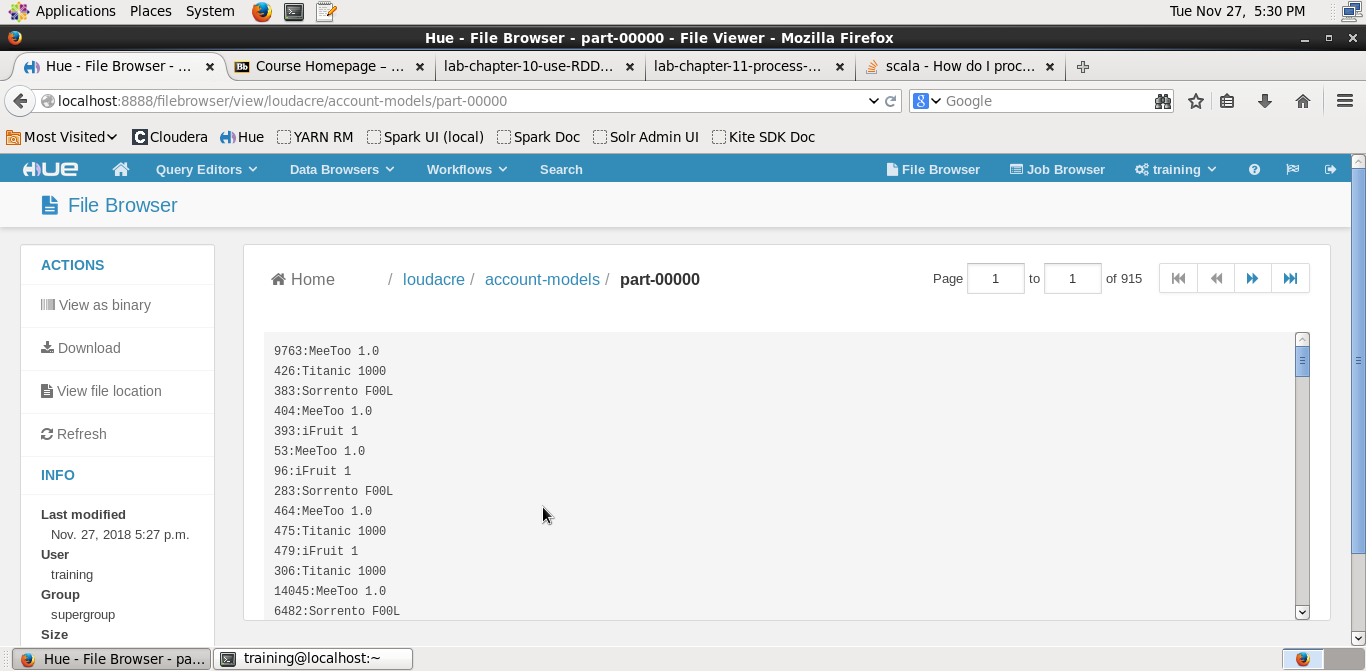




save the acc-models as a text file to loudacre with the name account-models. The below screenshot shows the part-00000 file created which has all the activations made in the format of account-number:model



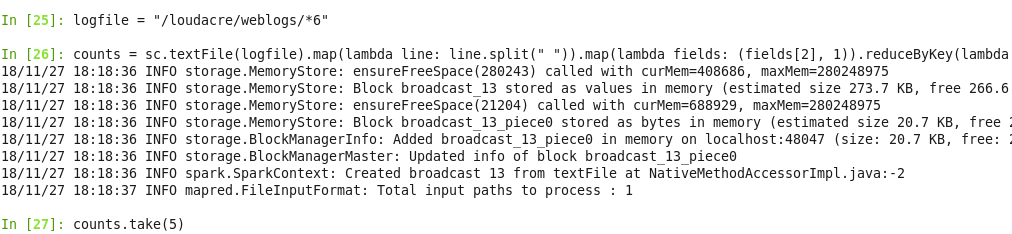
The contents of the file created are as below:



# Pair RDDs

First, using map and then reduce, we counted the number of requests from each user.

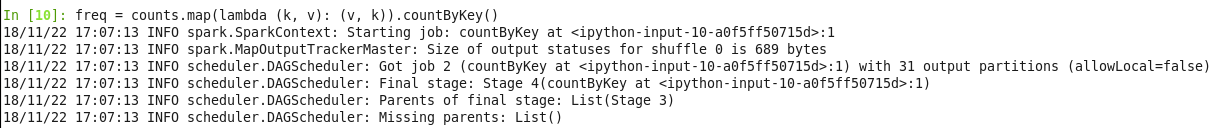
And displayed first 5 records of the result.

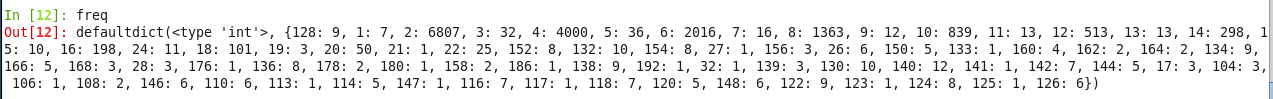




Then we utilized countByKey function to count users for each frequency:

freq = counts.map(lambda (k, v): (v, k)).countByKey()





After this, we create an RDD where user id is the key and list of IP addresses that user has used.

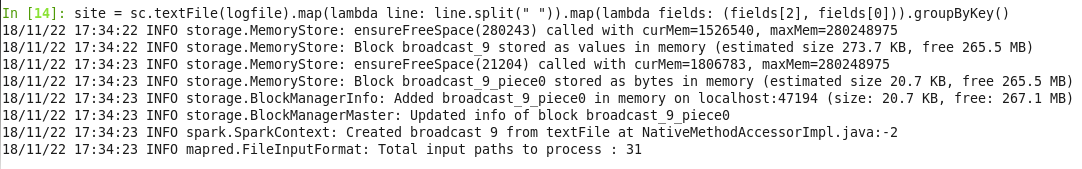
We used iteration command to view the output:

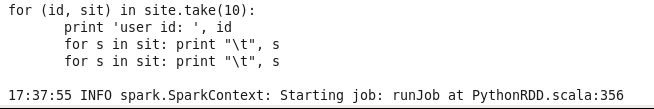
site = sc.textFile(logfile).map(lambda line: line.split(" ")).map(lambda fields: (fields[2],fields[0])).groupByKey()

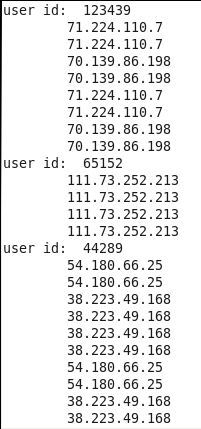
for (id, sit) in site.take(10):

print 'user id: ', id

for s in sit: print "\t", s







**Joining Weblog Data with Accounts Data**

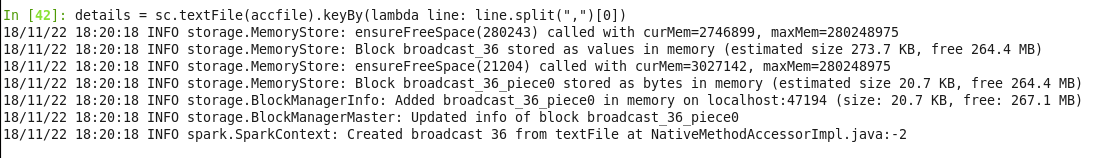
First, we create an RDD based on accounts data. We already have an RDD for Weblog data named ‘counts.’ Then, we join them to produce new data keyed by user ID.

accfile = "/loudacre/accounts/p\*"

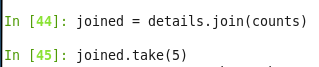
details = sc.textFile(accfile).keyBy(lambda line: line.split(",")[0])

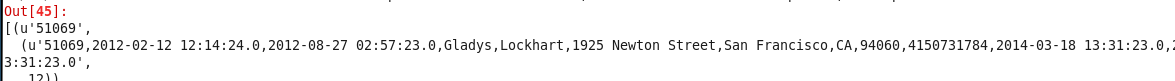
joined = details.join(counts)





We printed the first 5 records using take command:





# Spark app

In this lab, we write a Spark application rather than using the shell:

The following is the program that we write: [CountJPGs.py]

import sys

from pyspark import SparkContext

if \_\_name\_\_ == "\_\_main\_\_":

if len(sys.argv) < 2:

print >> sys.stderr, "Usage: WordCount <file>"

exit(-1)

sc = SparkContext()

count = sc.textFile(sys.argv[1]) \

.filter(lambda line: ".jpg" in line).count()

print "Number of .jpg files is ",count

sc.stop()



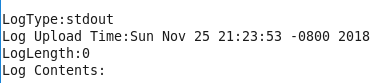
We then run the program as follows:

spark-submit --master yarn-client CountJPGs.py /loudacre/weblogs/\*



We take application id from above and give it to yarn:

yarn logs -applicationId application\_1536542472919\_0016



# Config Spark app

Ran the Count JPGs Python file from previous exercise:

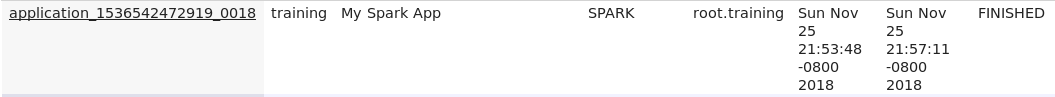


After the above step, changed the directory and created a myspark.conf file with the parameters mentioned in the lab exercise.

Then we rerun the application, using properties file instead of the script using the following command:

spark-submit --properties-file myspark.conf CountJPGs.py /loudacre/weblogs/\*

We can view the file using YARN UI to cross validate:



# 

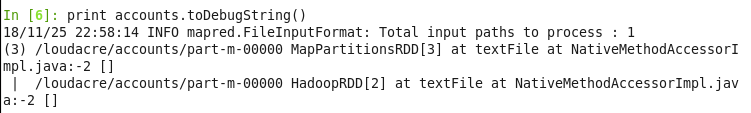
# View Jobs

Since, VM has only 1 cluster, and we want to simulate more realistic multi-node cluster, we run following command:

pyspark --master local[2]

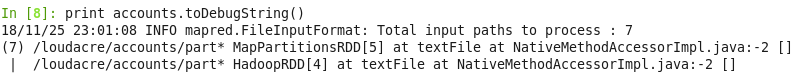
Created Partitions:





RDD - based on all files creates 7 partiton:

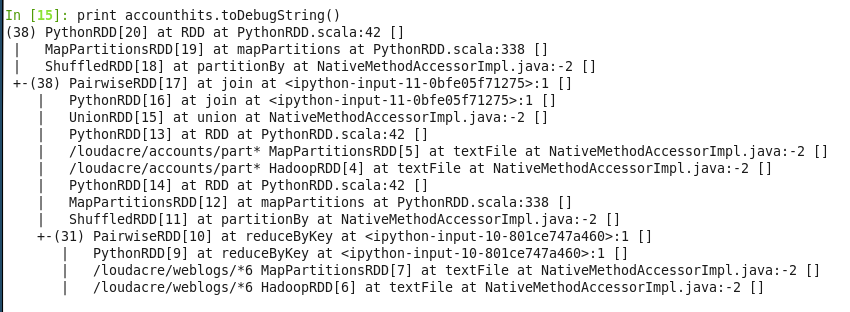




accountsByID = accounts.map(lambda s: s.split(',')).map(lambda values:(values[0],values[4] + ',' + values[3]))

userreqs = sc.textFile("/loudacre/weblogs/\*6").map(lambda line: line.split()).map(lambda words: (words[2],1)).reduceByKey(lambda v1,v2: v1+v2)

accounthits = accountsByID.join(userreqs).values()



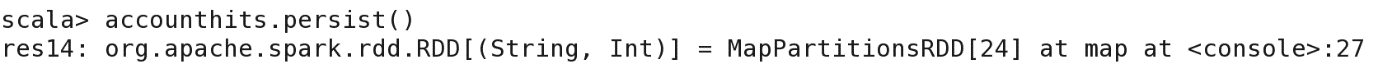
# Persist RDDs

Counting the number of user accounts with total hit count greater than five, we get 2410 results:

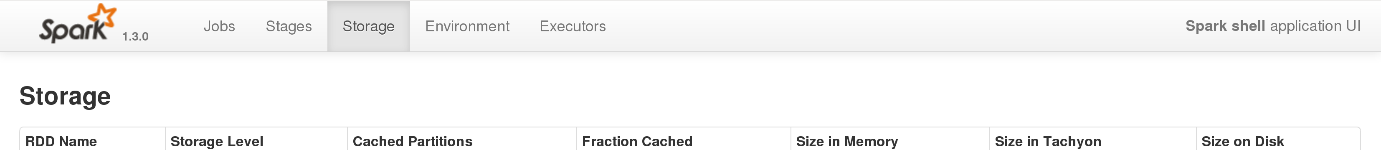




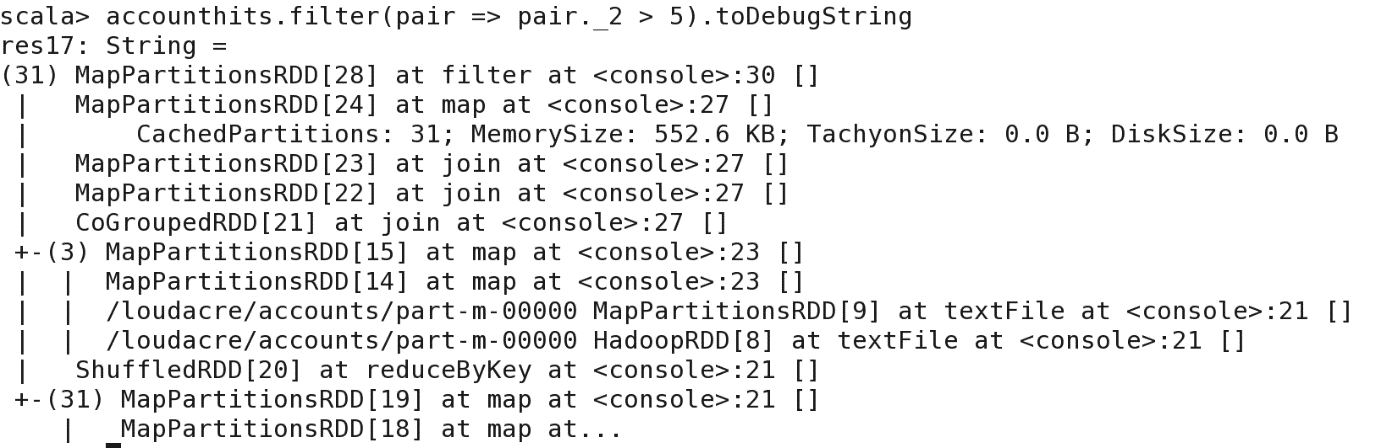
Cache the RDD by doing the following:



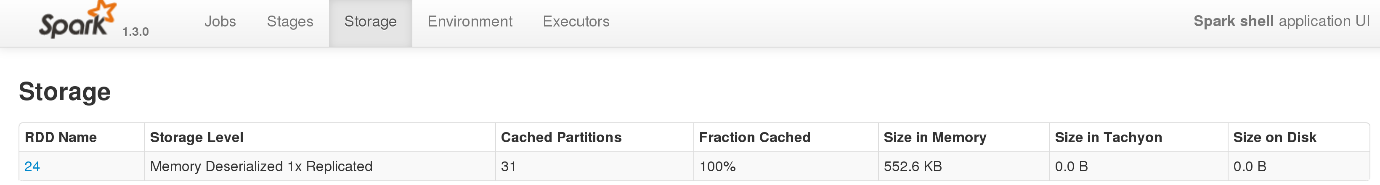
The Storage section of SPARK UI browser shows nothing as the RDD is persisted but no action is taken to be materialized:



Re-running the counting command and running toDebugString command:

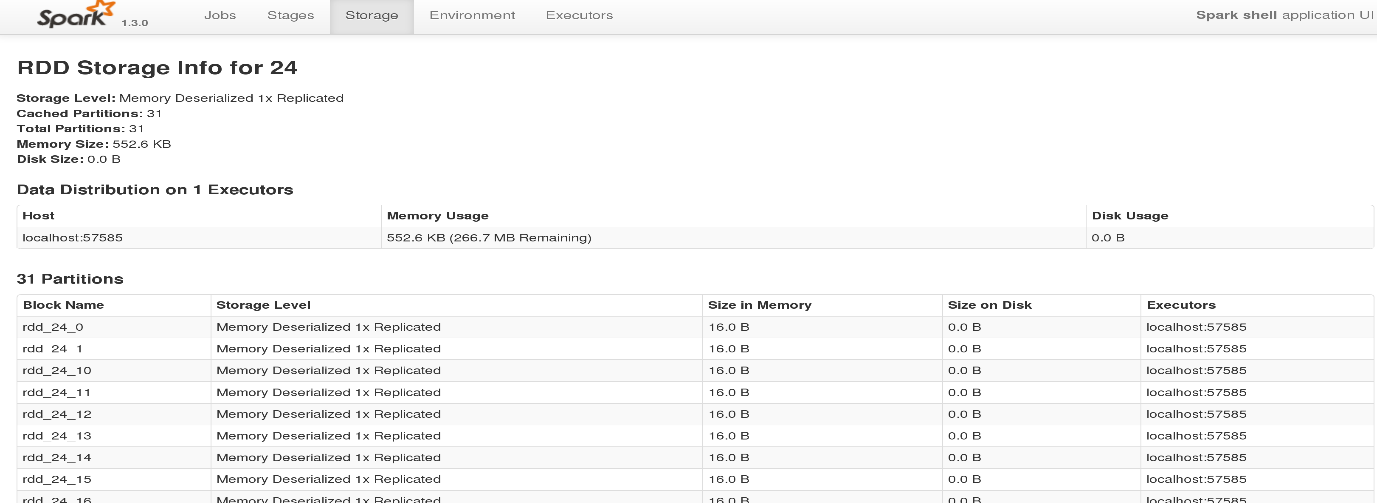


On refreshing Spark UI storage tab, we get:

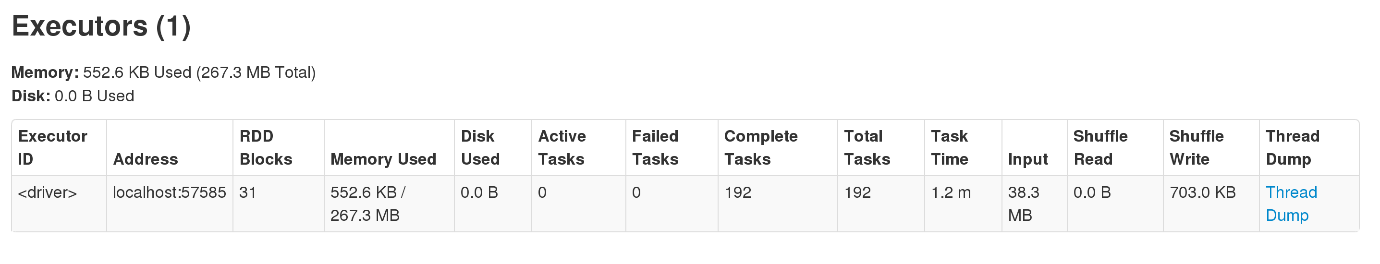


RDD which is persisted is shown above.

Details of RDD persistence and partitions are shown below:



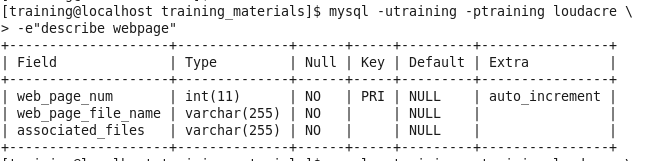
Amount of memory used is 552.6KB and 267.3 MB is available for one worker node



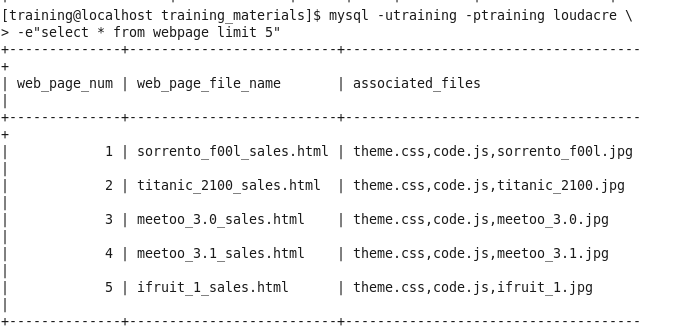
# Spark SQL

Review the contents from the loudacre mysql table:

Checked the table structure



Viewed the first 5 records from the table:



To normalize the data in the ‘associated\_files’, we perform the following steps:

We imported SQLContext and defined a local variable.

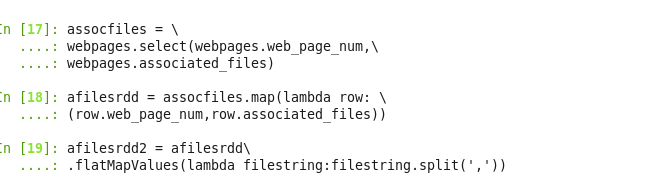
We created a new DataFrame ‘webpages’ based on webpage table and examined its contents using printSchema()



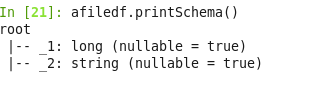
After the above steps, we selected required columns and created another dataframe called assocfiles.

We mapped this dataframe to an RDD to perform Spark operations.

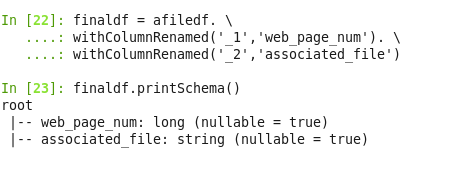
We split this new dataframe by specifying comma as a specifier.



We created another dataframe called ‘afiledf’ from the above dataframe and inspected its contents:

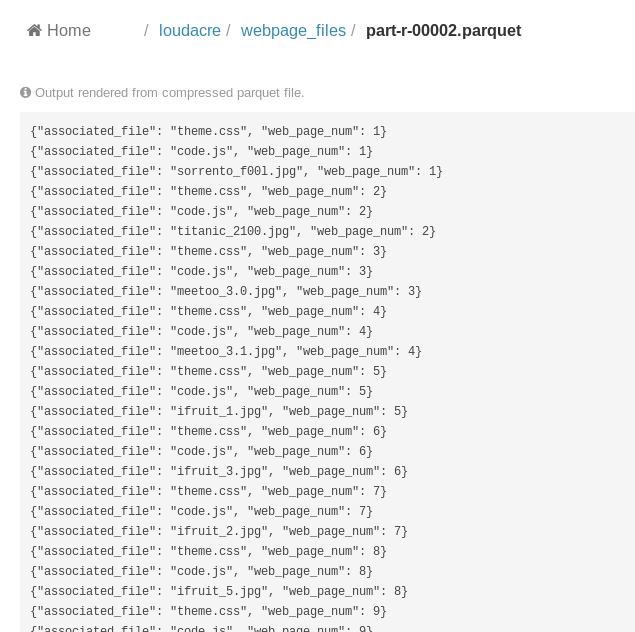


We renamed these columns as our requirements and assigned it to a new dataframe. We then checked the contents of this ‘finaldf’:



After this, we saved the contents of finaldf into ‘loudacre/webpage\_files/’ directory:

The files look like the following:



We then created the ‘webpage\_files’ table in Impala using the command:

CREATE EXTERNAL TABLE webpage\_files LIKE PARQUET

'/loudacre/webpage\_files/part-r-00001.parquet'

STORED AS PARQUET

LOCATION '/loudacre/webpage\_files'