Analyzing the Million Song Dataset

CIS 602: Project Presentation

Instructor: Dr. David Koop

Kevin D'Cruz
ID: 01656193
kdcruz@umassd.edu

Overview: An overview of the Project Report is as follows:

Goal: My goal of this project was to analyze the Million Song Dataset, available publicly, and answer questions/create visualizations relating to the dataset while focusing on the scalability factor.

Dataset Information:

Title: Million Song Dataset
URL: https://labrosa.ee.columbia.edu/millionsong/

This Dataset is made available by the Echo Nest & LabROSA (Columbia University). It consists of audio derived features & metadata for a million contemporary popular music tracks. It consists of 54 attributes, and is available in two versions. The sample subset of the dataset of 10,000 Records (1.8 GB), consists of 10,000 records. It is available here: https://labrosa.ee.columbia.edu/millionsong/pages/getting-dataset.

The Complete Dataset made available is of 280 GB and consists of 1 million records. Each record consists of 1 song, it's metadata and features. The dataset is in an HDF5 format, which is the Hierarchical Data Format. This Format is that of a Tree Structure, where data was stored in Nested folders. This page shows an example Track Description: https://labrosa.ee.columbia.edu/millionsong/pages/example-track-description.

This Dataset is also available on AWS Public Datasets where the size is of 500GB.

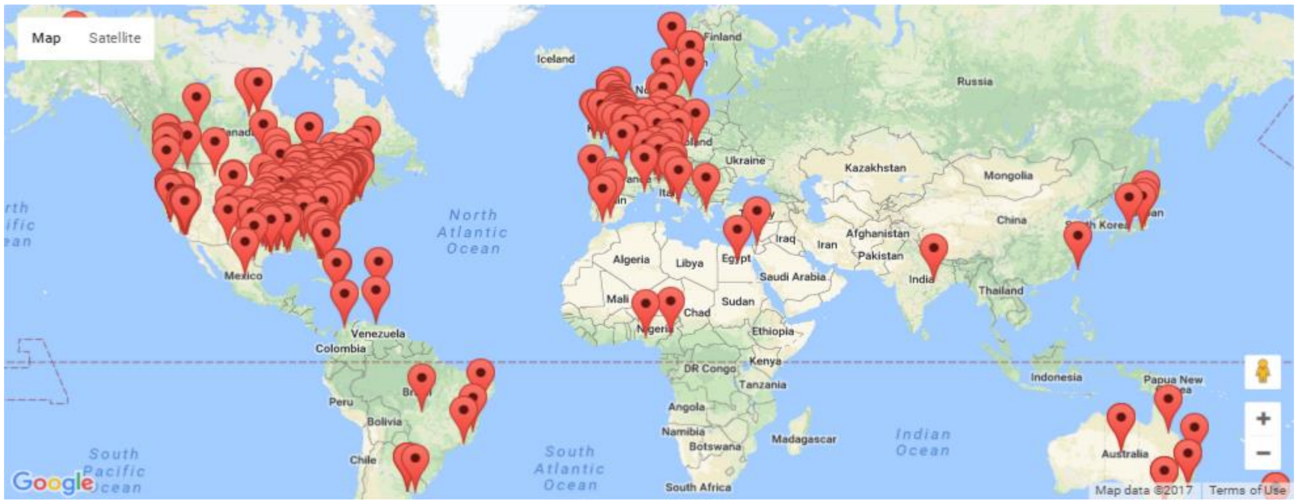
Technology Used:

- Jupyter Notebook
- Python
  - Pandas
  - Numpy/Pytables
  - Pyspark
- Amazon Web Services (AWS)

Analysis:

The Initial Analysis I carried out, was on the sample subset of 1.8 GB, of 10,000 songs. Since this data was available in HDF5 format(.h5), the first step was to convert it to a csv format. I used the following GitHub link(script) for conversion of the files to csv format: https://github.com/amgreenstreet/Million-Song-Dataset-HDF5-to-CSV. This conversion also needed another file called hdf5_getters.py which was available from the dataset's website. Upon conversion, I first used Pandas (only on the subset), followed by Pyspark (subset+complete) to answer the following questions:

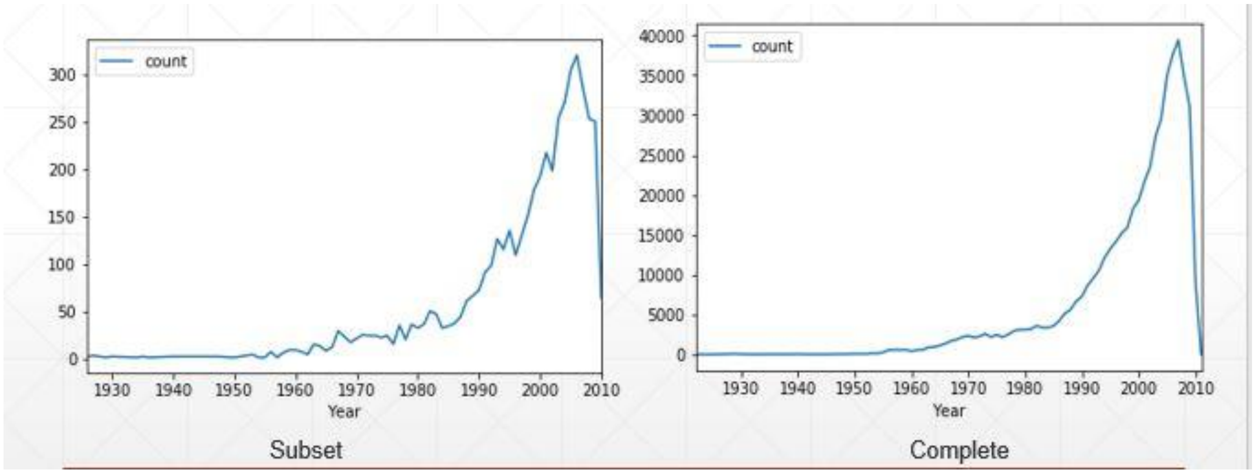
Question1: Which location most of the artists came from?



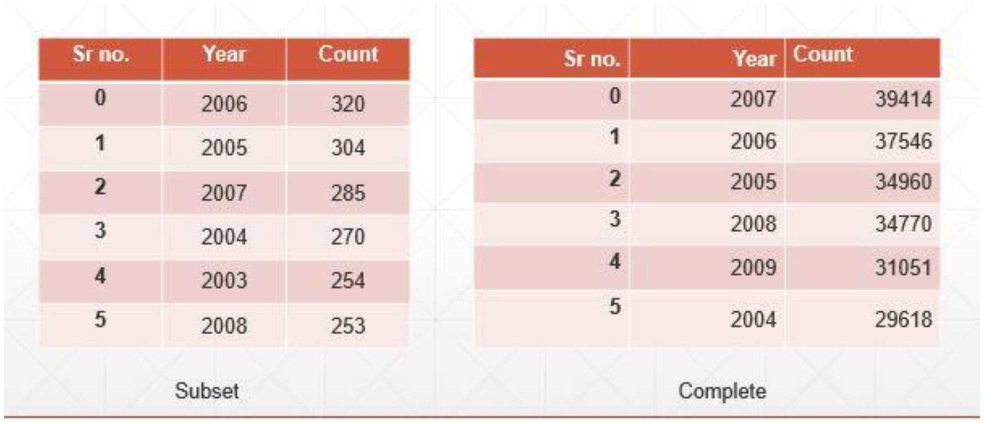
Map provided from Google Maps API

I plotted the Artist Locations for only 500 artists, since they were overlapping. Most of the artists came from the United States and Europe as seen in the map

Question2: What years had the highest number of album/song releases?



Line Graph showing highest number of song releases as per years. The graph peaked over the years 2006-07 for both Subset and complete data



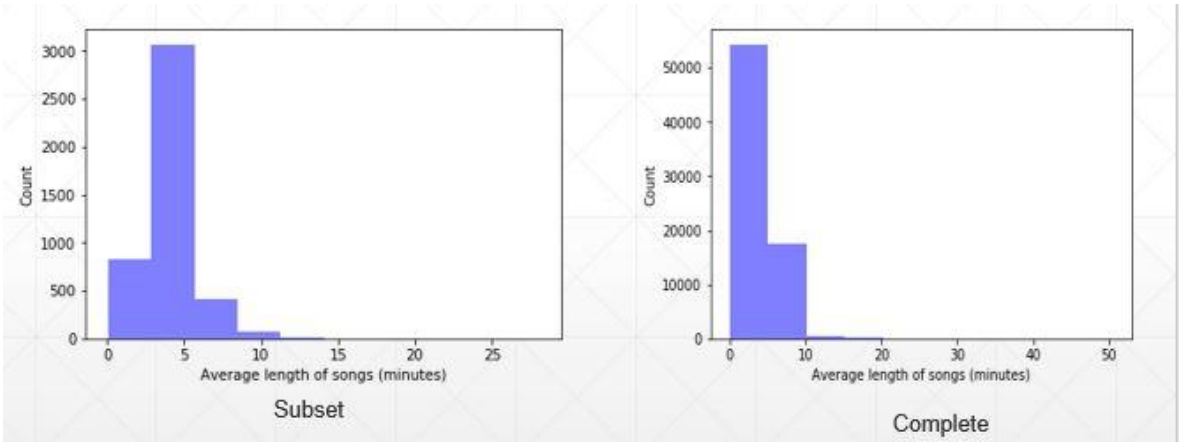
Sr no.	Year	Count
0	2006	320
1	2005	304
2	2007	285
3	2004	270
4	2003	254
5	2008	253

Subset

Sr no.	Year	Count
0	2007	39414
1	2006	37546
2	2005	34960
3	2008	34770
4	2009	31051
5	2004	29618

Complete

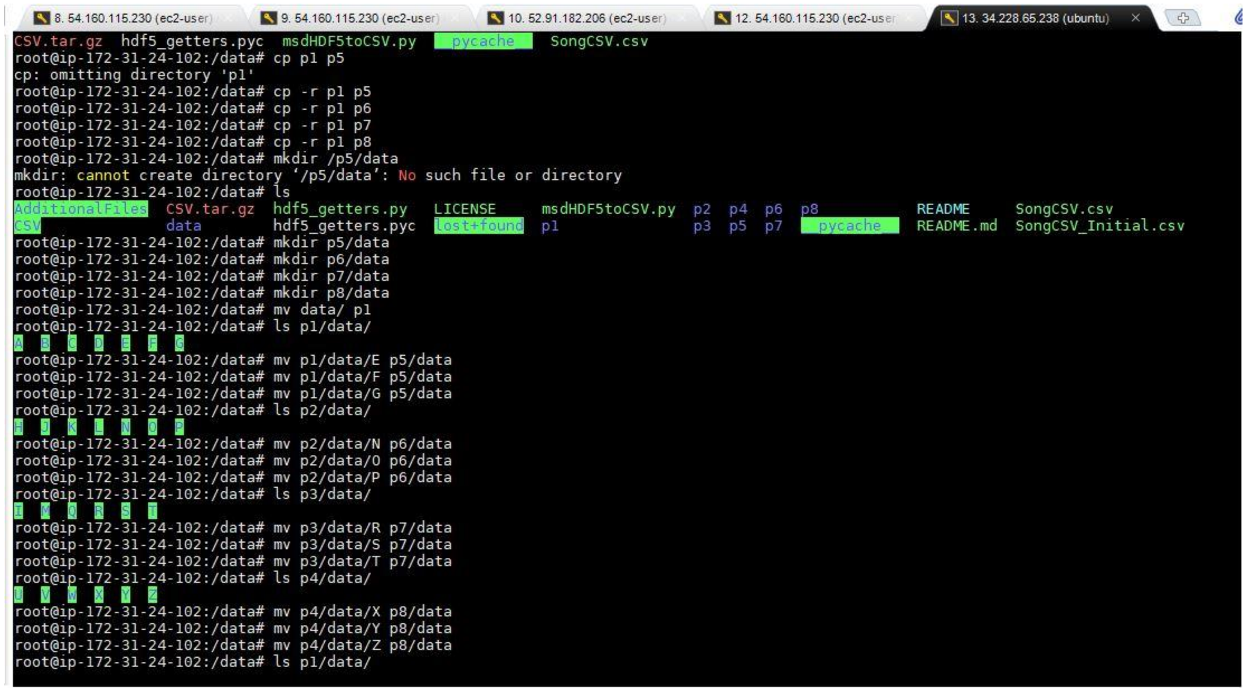
Question 3: What artists had the highest mean Duration of songs (in minutes)?



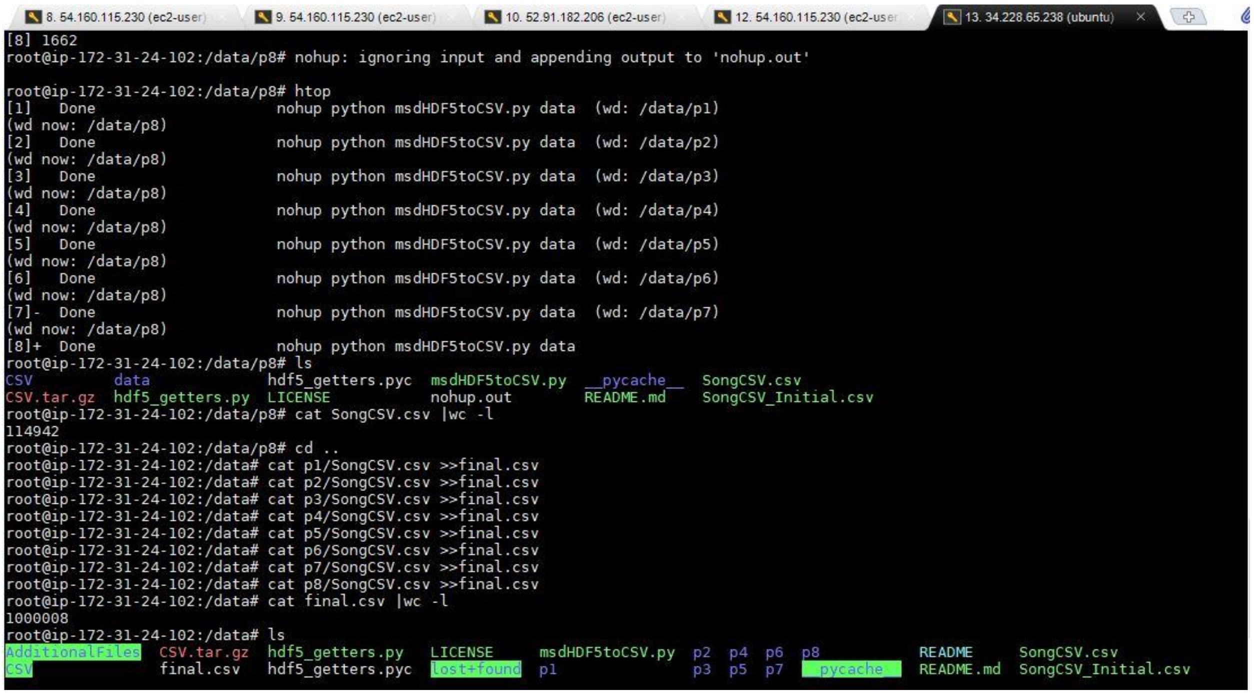
Most songs had a mean duration of 4-5 minutes for both subset & complete data

Challenges/Problems Faced:
The first challenge I faced was handling this large amount of data in HDF5 format. I initially planned working on loading this data directly in a Dataframe format without conversion to csv. I was unable to go about this and hence, converted it to a csv format, from the code referenced [3]. Another challenge faced was configuring AWS and loading the complete data (500 GB) for conversion.

Scalability Challenges:
The focus of this project was handling the scalability focus. The first scalability challenge was: Large data for Crunching. Handling 500 GB of data on a local computer would have scalability issues, in comparison with that of an AWS Cluster. I started this conversion on Amazon EMR by creating an EC2 Instance of m4.large. This provided me 2 Cores and 8 GB of RAM. I used Mobax for accessing the cluster from my local machine. Upon loading the data using this instance, it was taking a lot of time for conversion. The full data would roughly take about 12+ hours for conversion. Hence, I upgraded the setting by choosing a higher EC2 instance. This was the c4.2xlarge which provided 8 Cores and 15GB RAM. Another setting done was that I manually tweaked the loading of files by creating 8 folders(p1-p8), which replicated the 8 cores. I assigned 3-4 folders to each replicated core and begin loading the data, thus accelerating the process 8 times faster. This file conversion took the roughly 2 hours for obtaining the 1 million records in a CSV format. Following 3 screenshots show this process:


8. 54.160.115.230 (ec2-user)
9. 54.160.115.230 (ec2-user)
10. 52.91.182.206 (ec2-user)
12. 54.160.115.230 (ec2-user
13. 34.228.65.238 (ubuntu)

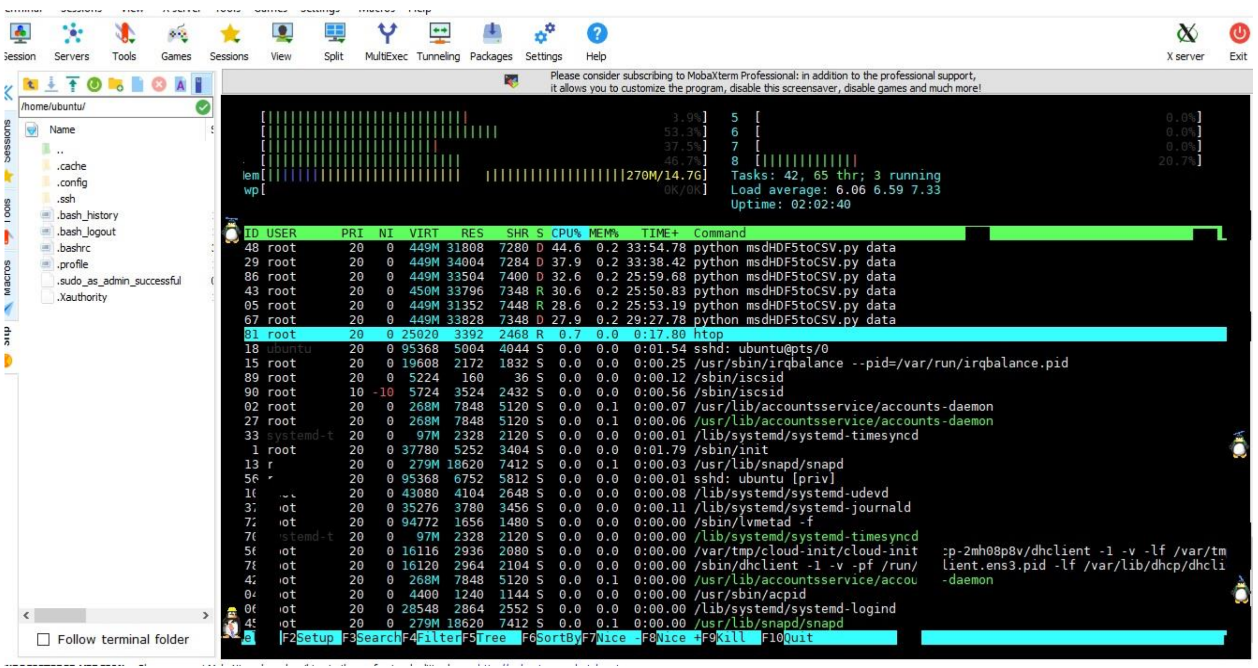
```
CSV.tar.gz  hdf5_getters.pyc  msdHDF5toCSV.py   __pycache__   SongCSV.csv
root@ip-172-31-24-102:/data# cp p1 p5
cp: omitting directory 'p1'
root@ip-172-31-24-102:/data# cp -r p1 p5
root@ip-172-31-24-102:/data# cp -r p1 p6
root@ip-172-31-24-102:/data# cp -r p1 p7
root@ip-172-31-24-102:/data# cp -r p1 p8
root@ip-172-31-24-102:/data# mkdir /p5/data
mkdir: cannot create directory '/p5/data': No such file or directory
root@ip-172-31-24-102:/data# ls
AdditionalFiles  CSV.tar.gz  hdf5_getters.py   LICENSE     msdHDF5toCSV.py  p2  p4  p6  p8           README     SongCSV.csv
CSV              data        hdf5_getters.pyc  lost+found  p1               p3  p5  p7  __pycache__  README.md  SongCSV_Initial.csv
root@ip-172-31-24-102:/data# mkdir p5/data
root@ip-172-31-24-102:/data# mkdir p6/data
root@ip-172-31-24-102:/data# mkdir p7/data
root@ip-172-31-24-102:/data# mkdir p8/data
root@ip-172-31-24-102:/data# mv data/ p1
root@ip-172-31-24-102:/data# ls p1/data/
A  B  C  D  E  F  G
root@ip-172-31-24-102:/data# mv p1/data/E p5/data
root@ip-172-31-24-102:/data# mv p1/data/F p5/data
root@ip-172-31-24-102:/data# mv p1/data/G p5/data
root@ip-172-31-24-102:/data# ls p2/data/
H  J  K  L  N  O  P
root@ip-172-31-24-102:/data# mv p2/data/N p6/data
root@ip-172-31-24-102:/data# mv p2/data/O p6/data
root@ip-172-31-24-102:/data# mv p2/data/P p6/data
root@ip-172-31-24-102:/data# ls p3/data/
I  M  Q  R  S  T
root@ip-172-31-24-102:/data# mv p3/data/R p7/data
root@ip-172-31-24-102:/data# mv p3/data/S p7/data
root@ip-172-31-24-102:/data# mv p3/data/T p7/data
root@ip-172-31-24-102:/data# ls p4/data/
U  V  W  X  Y  Z
root@ip-172-31-24-102:/data# mv p4/data/X p8/data
root@ip-172-31-24-102:/data# mv p4/data/Y p8/data
root@ip-172-31-24-102:/data# mv p4/data/Z p8/data
root@ip-172-31-24-102:/data# ls p1/data/
```



8. 54.160.115.230 (ec2-user) 9. 54.160.115.230 (ec2-user) 10. 52.91.182.206 (ec2-user) 12. 54.160.115.230 (ec2-user) 13. 34.228.65.238 (ubuntu)

```
[8] 1662
root@ip-172-31-24-102:/data/p8# nohup: ignoring input and appending output to 'nohup.out'

root@ip-172-31-24-102:/data/p8# htop
[1]   Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p1)
(wd now: /data/p8)
[2]   Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p2)
(wd now: /data/p8)
[3]   Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p3)
(wd now: /data/p8)
[4]   Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p4)
(wd now: /data/p8)
[5]   Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p5)
(wd now: /data/p8)
[6]   Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p6)
(wd now: /data/p8)
[7]-  Done                    nohup python msdHDF5toCSV.py data  (wd: /data/p7)
(wd now: /data/p8)
[8]+  Done                    nohup python msdHDF5toCSV.py data
root@ip-172-31-24-102:/data/p8# ls
CSV         data         hdf5_getters.pyc  msdHDF5toCSV.py  __pycache__  SongCSV.csv
CSV.tar.gz  hdf5_getters.py  LICENSE       nohup.out        README.md    SongCSV_Initial.csv
root@ip-172-31-24-102:/data/p8# cat SongCSV.csv |wc -l
114942
root@ip-172-31-24-102:/data/p8# cd ..
root@ip-172-31-24-102:/data# cat p1/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p2/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p3/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p4/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p5/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p6/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p7/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat p8/SongCSV.csv >>final.csv
root@ip-172-31-24-102:/data# cat final.csv |wc -l
1000008
root@ip-172-31-24-102:/data# ls
AdditionalFiles  CSV.tar.gz  hdf5_getters.py   LICENSE      msdHDF5toCSV.py  p2  p4  p6  p8           README     SongCSV.csv
CSV              final.csv   hdf5_getters.pyc  lost+found   p1               p3  p5  p7  __pycache__  README.md  SongCSV_Initial.csv
```



Please consider subscribing to MobaXterm Professional: in addition to the professional support, it allows you to customize the program, disable this screensaver, disable games and much more!

```
  [|||||||||||||||||||||||||||              3.9%]   5  [                        0.0%]
  [|||||||||||||||||||||||||||||||||       53.3%]   6  [                        0.0%]
  [|||||||||||||||||||||||                 37.5%]   7  [                        0.0%]
  [||||||||||||||||||||||||                46.7%]   8  [|||||||||||||          20.7%]
lem[|||||||||||||||||||  ||||||||||||||||||270M/14.7G]  Tasks: 42, 65 thr; 3 running
wp[                                         0K/0K]  Load average: 6.06 6.59 7.33
                                                    Uptime: 02:02:40

ID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
48 root       20   0  449M 31808  7280 D 44.6  0.2 33:54.78 python msdHDF5toCSV.py data
29 root       20   0  449M 34004  7284 D 37.9  0.2 33:38.42 python msdHDF5toCSV.py data
86 root       20   0  449M 33504  7400 D 32.6  0.2 25:59.68 python msdHDF5toCSV.py data
43 root       20   0  450M 33796  7348 R 30.6  0.2 25:50.83 python msdHDF5toCSV.py data
05 root       20   0  449M 31352  7448 R 28.6  0.2 25:53.19 python msdHDF5toCSV.py data
67 root       20   0  449M 33828  7348 D 27.9  0.2 29:27.78 python msdHDF5toCSV.py data
81 root       20   0 25020  3392  2468 R  0.7  0.0  0:17.80 htop
18 ubuntu     20   0 95368  5004  4044 S  0.0  0.0  0:01.54 sshd: ubuntu@pts/0
15 root       20   0 19608  2172  1832 S  0.0  0.0  0:00.25 /usr/sbin/irqbalance --pid=/var/run/irqbalance.pid
89 root       20   0  5224   160    36 S  0.0  0.0  0:00.12 /sbin/iscsid
90 root       10 -10  5724  3524  2432 S  0.0  0.0  0:00.56 /sbin/iscsid
02 root       20   0  268M  7848  5120 S  0.0  0.1  0:00.07 /usr/lib/accountsservice/accounts-daemon
27 root       20   0  268M  7848  5120 S  0.0  0.1  0:00.06 /usr/lib/accountsservice/accounts-daemon
33 systemd-t  20   0   97M  2328  2120 S  0.0  0.0  0:00.01 /lib/systemd/systemd-timesyncd
 1 root       20   0 37780  5252  3404 S  0.0  0.0  0:01.79 /sbin/init
13 r          20   0  279M 18620  7412 S  0.0  0.1  0:00.03 /usr/lib/snapd/snapd
56            20   0 95368  6752  5812 S  0.0  0.0  0:00.01 sshd: ubuntu [priv]
1(            20   0 43080  4104  2648 S  0.0  0.0  0:00.08 /lib/systemd/systemd-udevd
37  ot        20   0 35276  3780  3456 S  0.0  0.0  0:00.11 /lib/systemd/systemd-journald
72  ot        20   0 94772  1656  1480 S  0.0  0.0  0:00.00 /sbin/lvmetad -f
7(  stemd-t   20   0   97M  2328  2120 S  0.0  0.0  0:00.00 /lib/systemd/systemd-timesyncd
5(  ot        20   0 16116  2936  2080 S  0.0  0.0  0:00.00 /var/tmp/cloud-init/cloud-init   :p-2mh08p8v/dhclient -1 -v -lf /var/tm
7{  ot        20   0 16120  2964  2104 S  0.0  0.0  0:00.00 /sbin/dhclient -1 -v -pf /run/   lient.ens3.pid -lf /var/lib/dhcp/dhcli
4:  ot        20   0  268M  7848  5120 S  0.0  0.1  0:00.00 /usr/lib/accountsservice/accou   -daemon
0.  ot        20   0  4400  1240  1144 S  0.0  0.0  0:00.00 /usr/sbin/acpid
0(  ot        20   0 28548  2864  2552 S  0.0  0.0  0:00.00 /lib/systemd/systemd-logind
4:  ot        20   0  279M 18620  7412 S  0.0  0.1  0:00.00 /usr/lib/snapd/snapd
el   F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

For focusing on scalability, I used Pyspark on Jupyter Notebook which by default enables parallel processing.

Learning Outcomes:

This project helped me learn Pyspark and more importantly working and configuring AWS. I had never worked on such technologies before and hence got to learn a lot from it.

Conclusion:

I converted the sample subset of data on my local computer and the complete data on AWS EMR. Analysis was carried out on Jupyter Notebook using Pandas for the subset data and then Pyspark for both subset and complete data. Pandas when tried on the complete dataset was slightly slower than Pyspark and hence, I didn’t emphasize on it much.

References:

1. The Million Song Dataset, T. Bertin-Mahieux, D. Ellis, B. Whitman and P. Lamere, ISMIR '11
http://ismir2011.ismir.net/papers/OS6-1.pdf

2. The Million Song Dataset Challenge, B. McFee, T. Bertin-Mahieux, D. Ellis and G. Lanckriet, AdMIRe
https://pdfs.semanticscholar.org/73d2/16afb813f53134a3e635c3dcbecb612cb23e.pdf

3. Conversion (HDF5 to CSV) reference
https://github.com/amgreenstreet/Million-Song-Dataset-HDF5-to-CSV

4. Pyspark/AWS/Pandas Documentation