PROJECT - 3

Music Data Analysis

A leading music-catering company is planning to analyse large amount of data received from varieties of sources, namely mobile app and website to track the behaviour of users, classify users, calculate royalties associated with the song and make appropriate business strategies. The file server receives data files periodically after every 3 hours.

DataFiles:

Google Drive Link:

https://drive.google.com/drive/folders/0B_P3pWagdIrrMjJGVlNsSUEtbG8?usp=sharing

Fields present in the data files

Data files contain below fields.

Column Name/Field Name	Column Description/Field Description
User_id	Unique identifier of every user
Song_id	Unique identifier of every song
Artist_id	Unique identifier of the lead artist of the song
Timestamp	Timestamp when the record was generated
Start_ts	Start timestamp when the song started to play
End_ts	End timestamp when the song was stopped
Geo_cd	Can be 'A' for USA region, 'AP' for asia pacific region, 'J' for Japan region, 'E' for europe and 'AU' for australia region
Station_id	Unique identifier of the station from where the song was played
Song_end_type	How the song was terminated. 0 means completed successfully 1 means song was skipped 2 means song was paused 3 means other type of failure like device issue, network error etc.
Like	0 means song was not liked 1 means song was liked
Dislike	0 means song was not disliked 1 means song was disliked

Data Files:

Below is the data coming from web applications, that reside in /data/web and has xml format

```
<records>
<record>
<user_id>U106</user_id>
<song_id>S205</song_id>
<artist_id>A300</artist_id>
<timestamp>2016-05-10 12:24:22</timestamp>
<start_ts>2016-05-10 12:24:22</start_ts>
<end_ts>2017-05-09 08:09:22</end_ts>
<geo_cd>AP</geo_cd>
<station_id>ST407</station_id>
<song_end_type>2</song_end_type>
<like>1</like>
<dislike>1</dislike>
</record>
<record>
<user_id>U114</user_id>
<song_id>S209</song_id>
<artist_id>A303</artist_id>
<timestamp>2016-06-09 22:12:36</timestamp>
<start_ts>2016-05-10 12:24:22</start_ts>
<end_ts>2017-05-09 08:09:22</end_ts>
<geo_cd>U</geo_cd>
<station_id>ST411</station_id>
<song_end_type>2</song_end_type>
<like>1</like>
<dislike>0</dislike>
</record>
```

Below is a sample of the data coming from mobile applications, that reside in /data/mob and has csv format

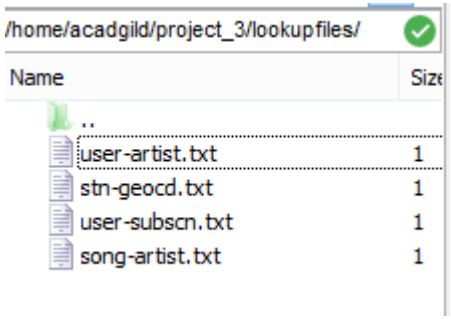
mob_input_file.txt

```
U114,S207,A303,1465130523,1465230523,1475130523,A,ST415,3,1,0
U107,S202,A303,1495130523,1465230523,1465230523,U,ST415,0,1,1
U100,S204,A302,1495130523,1475130523,1465130523,AU,ST408,2,1,1
U104,S202,A303,1465230523,1475130523,1465130523,A,ST409,2,0,1
U102,S207,A301,1465230523,1485130523,1465230523,AU,ST403,3,1,1
,S203,A302,1495130523,1475130523,1465230523,E,ST400,0,0,1
U106,S202,A302,1465230523,1465130523,1465130523,AU,ST408,0,1,1
U105,S207,A300,1465230523,1485130523,1465130523,U,ST400,2,0,1
U108,S205,A304,1465130523,1465130523,1475130523,,ST410,2,1,0
U105,S203,,1475130523,1465230523,1465130523,AU,ST408,2,0,1
U110,S203,A300,1465230523,1465130523,1485130523,A,ST415,0,1,1
U113,S200,A303,1465230523,1475130523,1465130523,E,ST413,3,1,1
U119,S208,A302,1495130523,1465230523,1465230523,U,ST415,3,0,0
U118,S208,A303,1475130523,1465130523,1465230523,E,ST415,3,0,0
U107,S210,A302,1475130523,1485130523,1485130523,AP,ST404,2,1,0
U118,S202,A300,1495130523,1465230523,1465230523,AP,ST410,1,0,0
U111,S206,A305,1465130523,1465130523,1485130523,AU,ST415,0,1,1
U116,S208,A303,1465230523,1485130523,1475130523,A,ST413,1,0,1
U101,S202,A300,1465230523,1465130523,1475130523,U,ST401,0,0,1
U120,S206,A303,1495130523,1485130523,1465130523,AU,ST414,0,0,0
```

Look-Up Tables Files:

There are some existing lookup tables present in NoSQL Databases that is used to perform data enrichment and analysis.

This data is present in lookup directory and loaded in HBase. There are 4 tables



song-artist : Columns: song_id, artist_id

song-artist.txt

```
S200,A300
S201,A301
S202,A302
S203,A303
S204,A304
S205,A301
S206,A302
S207,A303
S208,A304
S209,A305
```

stn-geocd : Columns: station_id, geo_cd

stn-geocd.txt

```
ST400,A
ST401,AU
ST402,AP
ST403,J
ST404,E
ST405,A
ST406,AU
ST407,AP
ST408,E
ST409,E
ST410,A
ST411,A
ST412,AP
ST413,J
ST414,E
```

user-subscn

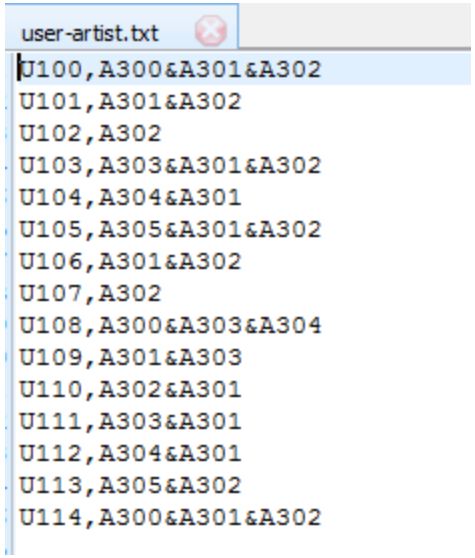
Columns: user_id, subscn_start_dt, subscn_end_dt

user-subscn.txt

```
U100,1465230523,1465130523
U101,1465230523,1475130523
U102,1465230523,1475130523
U103,1465230523,1475130523
U104,1465230523,1475130523
U105,1465230523,1475130523
U106,1465230523,1485130523
U107,1465230523,1455130523
U108,1465230523,1465230623
U109,1465230523,1475130523
U110,1465230523,1475130523
U111,1465230523,1475130523
U112,1465230523,1475130523
U113,1465230523,1485130523
U114,1465230523,1468130523
```

user-artist

Columns: user_id, artists_array



Below are the Steps followed to perform data analysis on the Music Data:

Step 1: Launch all necessary daemons

Step 2: Populate Look-Up tables (i.e. Load all data to HBase)

Step 3: Start Job Scheduling (using Crontab)

Step 4: Perform Data Formatting (using Spark)

Step 5: Perform Data Enrichment and Cleaning (using Spark)

Step 6: Perform Data Analysis (using Spark)

Step 7: Post Analysis

The steps followd are explained in detail.

Step 1: Launch all necessary daemons

- Run the shell script **start-daemons.sh**

```
[acadgild@localhost project_3]$ sh /home/acadgild/project_3/scripts/start-daemons.sh
/home/acadgild/project_3/logs/current-batch.txt: line 1: 1: command not found
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop-2.6.0/logs/hadoop-acadgild-namenode-localhost.localdomain.out
localhost: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/hadoop-acadgild-datanode-localhost.localdomain.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop-2.6.0/logs/hadoop-acadgild-secondarynamenode-localhost.localdomain.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop-2.6.0/logs/yarn-acadgild-resourcemanager-localhost.localdomain.out
localhost: starting nodemanager, logging to /usr/local/hadoop-2.6.0/logs/yarn-acadgild-nodemanager-localhost.localdomain.out
starting master, logging to /usr/local/hbase/logs/hbase-acadgild-master-localhost.localdomain.out
Safe mode is OFF
historyserver running as process 3530. Stop it first.
[acadgild@localhost project_3]$
```

In the shell script **start-daemons.sh**, we perform the following operations:

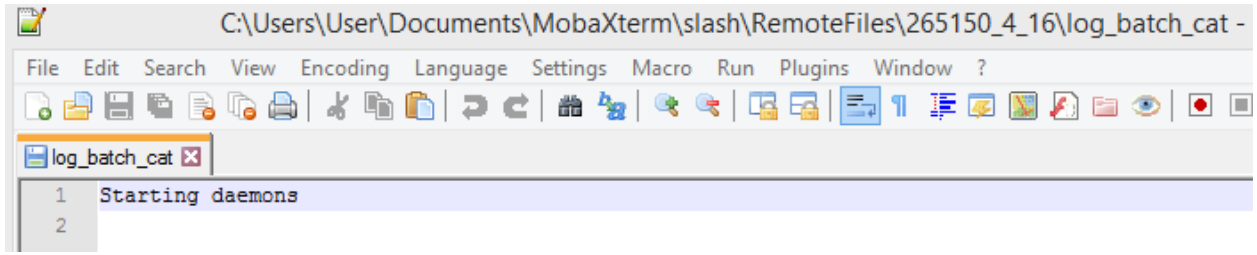
- Check if a file current-batch.txt has been created or not,If already created, print Batch File Found! else create the file and add 1 to it to signify batch 1.
- Get the batch id number from the batch file created and create a Log File for the batch using the batch id. This will be log_batch_1.
- Give permissions to the file, so that we are able to modify it on the run.First using the chmod command, modify the access permission to the scripts folder of project_3, so we are able to run scripts from the bash shell.
- Through out the course of the analysis process logs folder will document the tasks that are performed for the Music Data Analysis.
- Add a log to the Log File signifying that the all necessary daemons have been started.
- Start the dfs, yarn, hbase and jobhistory daemons. Added command to leave the namenode from safemode

start-daemons.sh

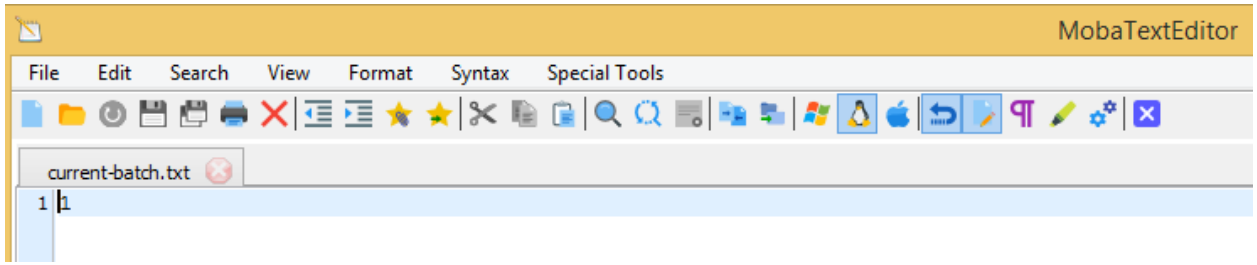
```
#!/bin/bash

if [ -f "/home/acadgild/project_3/logs/current-batch.txt" ]
then
 echo "Batch File Found!"
else
 echo -n "1" > "/home/acadgild/project_3/logs/current-batch.txt"
fi

chmod 775 /home/acadgild/project_3/logs/current-batch.txt
batchid=cat `/home/acadgild/project_3/logs/current-batch.txt`
LOGFILE=/home/acadgild/project_3/logs/log_batch_$batchid

echo "Starting daemons" >> $LOGFILE

start-all.sh
start-hbase.sh
hdfs dfsadmin -safemode leave
mr-jobhistory-daemon.sh start historyserver
```

- Once the **start-daemons.sh** command is run, the logmessage and batch number generated are as below.

Log message



Batch Number



Step 2: Populate Look-Up tables (i.e. Loaded all data to HBase)

Next populate the hbase lookuptables with the data files from lookup folder. Below is the shell script **populate-lookup.sh** to load lookup data into HBase tables.

The following operations are performed while running **populate-lookup.sh**:

- Get the batch id number from the batch file and generate the **Log File** for populate lookup process using the batch id. This name of the file will be **populate_lookup_log_batch_1**
- Add log messages to the Log File signifying that the lookup tables are being created and populated
- Create the HBase tables for the lookup data files: **song-artist**, **stn-geocd** and **user-subscn** with their column families
- The **user-artist** lookup has an array column that is difficult to populate in HBase, so do not create user-artist table in HBase.

Executing the populate lookup shell Script

```
[acadgild@localhost project_3]$ sh /home/acadgild/project_3/scripts/populate-lookup.sh
2018-01-07 06:29:30,217 INFO  [main] Configuration.deprecation: hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.14-hadoop2, r4e4aabb93b52f1b0fef6b66edd06ec8923014dec, Tue Aug 25 22:35:44 PDT 2015
```

Creating Hbase Tables

populate-lookup.sh

```
#!/bin/bash

batchid=`cat /home/acadgild/project_3/logs/current-batch.txt`

LOGFILE=/home/acadgild/project_3/logs/populate_lookup_log_batch_$batchid

echo "Creating LookUp Tables" >> $LOGFILE

echo "create 'station-geo-map', 'geo'" | hbase shell
echo "create 'subscribed-users', 'subscn'" | hbase shell
echo "create 'song-artist-map', 'artist'" | hbase shell
```

- Once HBase table is created, in every lookup data file, read each line, extract the columns (comma separated) and add the data as rows to the corresponding HBase tables created above

Loading the data from lookup files into the Hbase Tables

```
13
14 echo "Populating LookUp Tables" >> $LOGFILE
15
16 file="/home/acadgild/project_3/lookupfiles/stn-geocd.txt"
17 while IFS= read -r line
18 do
19  stnid=`echo $line | cut -d',' -f1`
20  geocd=`echo $line | cut -d',' -f2`
21  echo "put 'station-geo-map', '$stnid', 'geo:geo_cd', '$geocd'" | hbase shell
22 done <"$file"
23  echo "scan 'station-geo-map'" | hbase shell
24
25
26 file="/home/acadgild/project_3/lookupfiles/song-artist.txt"
27 while IFS= read -r line
28 do
29  songid=`echo $line | cut -d',' -f1`
30  artistid=`echo $line | cut -d',' -f2`
31  echo "put 'song-artist-map', '$songid', 'artist:artistid', '$artistid'" | hbase shell
32 done <"$file"
33  echo "scan 'song-artist-map'" | hbase shell
34
35
36 file="/home/acadgild/project_3/lookupfiles/user-subscn.txt"
37 while IFS= read -r line
38 do
39  userid=`echo $line | cut -d',' -f1`
40  startdt=`echo $line | cut -d',' -f2`
41  enddt=`echo $line | cut -d',' -f3`
42  echo "put 'subscribed-users', '$userid', 'subscn:startdt', '$startdt'" | hbase shell
43  echo "put 'subscribed-users', '$userid', 'subscn:enddt', '$enddt'" | hbase shell
44 done <"$file"
45 echo "scan 'subscribed-users'" | hbase shell
```

- Once all the commands in **populate-lookup.sh** are executed, the logmessage generated is as below.

```
populate_lookup_l...
1 Creating LookUp Tables
2 Populating LookUp Tables
3
```

- The contents of HBase table are as below.

```
hbase(main):002:0> scan 'song-artist-map'
ROW                                COLUMN+CELL
 S200                              column=artist:artistid, timestamp=1515255632129, value=A300
 S201                              column=artist:artistid, timestamp=1515255632129, value=A301
 S202                              column=artist:artistid, timestamp=1515255632129, value=A302
 S203                              column=artist:artistid, timestamp=1515255632129, value=A303
 S204                              column=artist:artistid, timestamp=1515255632129, value=A304
 S205                              column=artist:artistid, timestamp=1515255632129, value=A301
 S206                              column=artist:artistid, timestamp=1515255632129, value=A302
 S207                              column=artist:artistid, timestamp=1515255632129, value=A303
 S208                              column=artist:artistid, timestamp=1515255632129, value=A304
 S209                              column=artist:artistid, timestamp=1515255632129, value=A305
10 row(s) in 0.7400 seconds
```

```
hbase(main):004:0> scan 'station-geo-map'
ROW                                   COLUMN+CELL
 ST400                                column=geo:geo_cd, timestamp=1515255490235, value=A
 ST401                                column=geo:geo_cd, timestamp=1515255490235, value=AU
 ST402                                column=geo:geo_cd, timestamp=1515255490235, value=AP
 ST403                                column=geo:geo_cd, timestamp=1515255490235, value=J
 ST404                                column=geo:geo_cd, timestamp=1515255490235, value=E
 ST405                                column=geo:geo_cd, timestamp=1515255490235, value=A
 ST406                                column=geo:geo_cd, timestamp=1515255490235, value=AU
 ST407                                column=geo:geo_cd, timestamp=1515255490235, value=AP
 ST408                                column=geo:geo_cd, timestamp=1515255490235, value=E
 ST409                                column=geo:geo_cd, timestamp=1515255490235, value=E
 ST410                                column=geo:geo_cd, timestamp=1515255490235, value=A
 ST411                                column=geo:geo_cd, timestamp=1515255490235, value=A
 ST412                                column=geo:geo_cd, timestamp=1515255490235, value=AP
 ST413                                column=geo:geo_cd, timestamp=1515255490235, value=J
 ST414                                column=geo:geo_cd, timestamp=1515255490235, value=E
15 row(s) in 0.4950 seconds
```

```
hbase(main):003:0> scan 'subscribed-users'
ROW                                   COLUMN+CELL
 U100                                 column=subscn:enddt, timestamp=1515255795318, value=1465130523
 U100                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U101                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U101                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U102                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U102                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U103                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U103                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U104                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U104                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U105                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U105                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U106                                 column=subscn:enddt, timestamp=1515255795318, value=1485130523
 U106                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U107                                 column=subscn:enddt, timestamp=1515255795318, value=1455130523
 U107                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U108                                 column=subscn:enddt, timestamp=1515255795318, value=1465230623
 U108                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U109                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U109                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U110                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U110                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U111                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U111                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U112                                 column=subscn:enddt, timestamp=1515255795318, value=1475130523
 U112                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U113                                 column=subscn:enddt, timestamp=1515255795318, value=1485130523
 U113                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
 U114                                 column=subscn:enddt, timestamp=1515255795318, value=1468130523
 U114                                 column=subscn:startdt, timestamp=1515255795318, value=1465230523
15 row(s) in 1.0700 seconds
```

Step 3: Start Job Scheduling (using Crontab)

- To automate the Data Cleaning, Validation, Enrichment, Analysis and Post Analysis, open the crontab file:

```
[acadgild@localhost project_3]$ sudo crontab -e
[sudo] password for acadgild:
crontab: installing new crontab
[acadgild@localhost project_3]$
```

- Insert the below statement

* */3 * * * /home/acadgild/project/scripts/wrapper.sh

```
* */3 * * * /home/acadgild/project_3/scripts/wrapper.sh
~
```

- Crontab is used for Job Scheduling. In the -e mode, Crontab schedules execution of commands by a regular user.

The statement above runs the wrapper.sh shell script every 3 hours.

wrapper.sh

```
1 #!/bin/bash
2
3 sh /home/acadgild/project/scripts/dataformatting.sh
4
5 sh /home/acadgild/project/scripts/data_enrichment_cleaning.sh
6
7 sh /home/acadgild/project/scripts/data_analysis.sh
8
```

Step 4: Perform Data Formatting (using Spark)

Perform Data Formatting

Below is the shell script **dataformatting.sh** that is used to:

- Format the web xml file using databricks and mob csv file using Spark SQL
- Load the 2 data files, mob and web into Spark.

Executing the Dataformatting Script

```
[acadgild@localhost project_3]$ sh /home/acadgild/project_3/scripts/dataformatting.sh
Ivy Default Cache set to: /home/acadgild/.ivy2/cache
The jars for the packages stored in: /home/acadgild/.ivy2/jars
:: loading settings :: url = jar:file:/home/acadgild/spark-2.2.0-bin-hadoop2.7/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysetti
ngs.xml
com.databricks#spark-csv_2.11 added as a dependency
```

dataformatting.sh

```
#!/bin/bash

batchid=`cat /home/acadgild/project_3/logs/current-batch.txt`
LOGFILE=/home/acadgild/project_3/logs/dataformatting_log_batch_$batchid

echo "Running Spark script for data formatting the data files..." >> $LOGFILE

echo "Converting the web data timestamp, startdate, enddate to milliseconds..." >> $LOGFILE

cat /home/acadgild/project_3/scripts/data_formatting.scala | spark-shell  --jars "spark-xml_2.11-0.4.1.jar,spark-csv_2.11-1.4.0.jar"  --packages com.databricks:spark-
-xml_2.11:0.4.1   --packages com.databricks:spark-csv_2.11:1.4.0

echo "Combining web and mobile data input data using unionall" >> $LOGFILE

echo "Storing the union results in Local file as CSV" >> $LOGFILE

```

The following operations are performed while running **dataformatting.sh**:

- Get the batch id number from the batch file and generate the Log File for the batch for populate lookup process using the batch id. This will be dataformatting_log_batch_1
- Start the spark shell with databricks jars, that are needed to parse xml data and running the **data_formatting.scala** file.

The following operations are performed in **data_formatting.scala**:

- Using databricks, parsing the xml file as Spark SQL table and storing the data with variable web_input.
- Converting the timestamp, start_date and end_date values of web_input into milliseconds by using the function **parseTimeStampToMillies**, as the lookup and mobile datetime are in milliseconds
- Creating a case class Music_Data to infer the schema of web_input.
- Parsing the csv data as mob_input and inferring the schema using case class Music data.
- Finally, Since both the web and mobile data are inferred as Music Data, combine both the data and store the combined data in the HDFS the location /user/acadgild/project_3/processed/formatted_input.csv , which will be used for data enrichment.

- The contents of formatted_input.csv from HDFS is as below..

[acadgild@localhost project_3]$ hadoop fs -ls /user/acadgild/project_3/processed/formatted_input.csv

Found 2 items

-rw-r--r-- 3 acadgild supergroup 0 2018-01-07 07:35 /user/acadgild/project_3/processed/formatted_input.csv/_SUCCESS

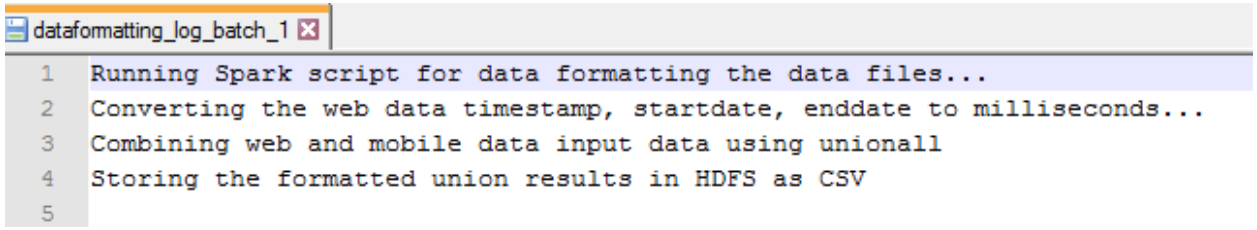
-rw-r--r-- 3 acadgild supergroup 2572 2018-01-07 07:35 /user/acadgild/project_3/processed/formatted_input.csv/part-00000-94c6b386-5f47-4855-a10a-80999a8d3a3a-c000.csv

[acadgild@localhost project_3]$ hadoop fs -cat /user/acadgild/project_3/processed/formatted_input.csv/part-00000-94c6b386-5f47-4855-a10a-80999a8d3a3a-c000.csv

user_id,song_id,artist_id,timestamp,start_ts,end_ts,geo_cd,station_id,song_end_type,like,dislike
U106,S205,A300,1462863262,1462863262,1494297562,AP,ST407,2,1,1
U114,S209,A303,1465490556,1462863262,1494297562,U,ST411,2,1,0
U113,S203,A304,1465490556,1465490556,1462863262,U,ST405,0,0,1
U108,S200,A302,1468094889,1462863262,1468094889,U,ST414,0,0,1
U102,S203,A305,1465490556,1465490556,1494297562,U,ST404,2,0,0
,S208,A300,1465490556,1494297562,1465490556,U,ST411,1,0,1
U115,S200,A300,1465490556,1494297562,1465490556,AU,ST404,3,0,0
U111,S204,A300,1465490556,1465490556,1468094889,U,ST410,3,1,1
U120,S201,A300,1494297562,1465490556,1468094889,,ST410,3,0,1
U113,S203,,1465490556,1465490556,1465490556,A,ST402,1,1,0
U109,S203,A304,1462863262,1494297562,1468094889,E,ST405,1,1,1
U110,S202,A303,1494297562,1494297562,1468094889,AU,ST402,2,1,0
U100,S200,A301,1494297562,1494297562,1494297562,AP,ST410,3,1,1
U101,S208,A300,1462863262,1468094889,1462863262,E,ST408,0,1,1
U106,S206,A300,1494297562,1465490556,1462863262,A,ST405,3,1,0
U107,S202,A304,1494297562,1468094889,1462863262,U,ST409,0,0,0
U103,S204,A300,1468094889,1494297562,1465490556,AU,ST411,2,1,0
U103,S202,A300,1465490556,1465490556,1465490556,A,ST415,2,1,1
U113,S203,A303,1462863262,1468094889,1494297562,U,ST408,2,0,0
U113,S204,A301,1494297562,1494297562,1465490556,E,ST415,3,0,1
U114,S207,A303,1465130523,1465230523,1475130523,A,ST415,3,1,0

U107,S202,A303,1495130523,1465230523,1465230523,U,ST415,0,1,1
U100,S204,A302,1495130523,1475130523,1465130523,AU,ST408,2,1,1
U104,S202,A303,1465230523,1475130523,1465130523,A,ST409,2,0,1
U102,S207,A301,1465230523,1485130523,1465230523,AU,ST403,3,1,1
,S203,A302,1495130523,1475130523,1465230523,E,ST400,0,0,1
U106,S202,A302,1465230523,1465130523,1465130523,AU,ST408,0,1,1
U105,S207,A300,1465230523,1485130523,1465130523,U,ST400,2,0,1
U108,S205,A304,1465130523,1465130523,1475130523,,ST410,2,1,0
U105,S203,,1475130523,1465230523,1465130523,AU,ST408,2,0,1
U110,S203,A300,1465230523,1465130523,1485130523,A,ST415,0,1,1
U113,S200,A303,1465230523,1475130523,1465130523,E,ST413,3,1,1
U119,S208,A302,1495130523,1465230523,1465230523,U,ST415,3,0,0
U118,S208,A303,1475130523,1465130523,1465230523,E,ST415,3,0,0
U107,S210,A302,1475130523,1485130523,1485130523,AP,ST404,2,1,0
U118,S202,A300,1495130523,1465230523,1465230523,AP,ST410,1,0,0
U111,S206,A305,1465130523,1465130523,1485130523,AU,ST415,0,1,1
U116,S208,A303,1465230523,1485130523,1475130523,A,ST413,1,0,1
U101,S202,A300,1465230523,1465130523,1475130523,U,ST401,0,0,1
U120,S206,A303,1495130523,1485130523,1465130523,AU,ST414,0,0,0

- Once the **dataformatting.sh** command completed the execution, logmessage generated are as below.



Loading the Web and Mobile Data in Spark and formatting

data_formatting.scala

```
import org.apache.spark.sql.functions._
import com.databricks.spark.xml._
import com.databricks.spark.csv._
//Retriving web data.
val web_file = spark.read.format("com.databricks.spark.xml").option("rootTag", "records").option("rowTag", "record").load("/home/acadgild/project_3/data/Web/file.xml")
val records_web = web_file.select("user_id","song_id","artist_id","timestamp","start_ts","end_ts","geo_cd","station_id","song_end_type","like","dislike")

import org.joda.time.{DateTimeZone}
import org.joda.time.format.DateTimeFormat
import org.joda.time.DateTime
import java.sql.Timestamp
import java.text.SimpleDateFormat;
def parseTimeStampToMillies(timestamp :Timestamp):Long = (timestamp.getTime()/1000)

//Function to convert timestamp
val findTimestampVal = udf(parseTimeStampToMillies(_:Timestamp))
//convert timestamp for web data to milliseconds
val records_web_modified = ((records_web.withColumn("timestamp", findTimestampVal($"timestamp")).withColumn("start_ts", findTimestampVal($"start_ts"))).withColumn("end_ts",
findTimestampVal($"end_ts")))

//case class
case class Music_Data(user_id: String, song_id: String,artist_id: String,timestamp: Long, start_ts:Long,end_ts:Long, geo_cd: String,station_id:String, song_end_type: Long,
like:Long,dislike:Long)  //columns and data types
//creating a case class from row:
val records_web_final = records_web_modified.map(row=> Music_Data(row.getString(0), row.getString(1), row.getString(2), row.getLong(3), row.getLong(4),
row.getLong(5),row.getString(6),row.getString(7),row.getLong(8),row.getLong(9), row.getLong(10)))

//Retriving mobile data.
val mob_file = spark.read.textFile("/home/acadgild/project_3/data/Mob/file.txt")
val records_mob = mob_file.map(line => line.split(",")).map(rec=> Music_Data(rec(0), rec(1), rec(2), rec(3).toLong, rec(4).toLong,
rec(5).toLong,rec(6),rec(7),rec(8).toLong,rec(9).toLong,rec(10).toLong))

//combining mobile and web data
val combined = records_web_final.unionAll(records_mob)
combined.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").save("hdfs://localhost:9000/user/acadgild/project_3/processed/formatted_input.csv")
```

Step 5: Perform Data Enrichment and Cleaning (using Spark)

Perform Data Enrichment

Below is the shell script **data_enrichment_cleaning.sh** is used to:

- Load HBase tables in Spark shell and create Spark SQL tables.
- Load the formatted CSV data from HDFS for data enrichment and cleaning

Executing the Dataformatting Script

data_enrichment_c...

```
#!/bin/bash

batchid=`cat /home/acadgild/project_3/logs/current-batch.txt`
LOGFILE=/home/acadgild/project_3/logs/data_enrichment_log_batch_$batchid

echo "Running Spark script for data enrichment..." >> $LOGFILE

echo "Retriving HBase data using spark hbaseTable function" >> $LOGFILE

cat /home/acadgild/project_3/scripts/data_enrichment_cleaning.scala | spark-shell  --jars "hbase-annotations.jar,hbase-common.jar,hbase-server.jar,hbase-client.jar
,hbase-protocol.jar,hbase-hadoop2-compat.jar,hbase-annotations.jar,zookeeper.jar,spark-hbase-connector_2.10-0.9.2.jar,spark-csv_2.11-1.4.0.jar"  --packages com
.databricks:spark-csv_2.11:1.4.0

echo "Running spark script for data enrichment, cleaning and filterving valid data" >> $LOGFILE

echo "Storing the cleaned and valid data in HDFS as CSV..." >> $LOGFILE


```

The following operations are performed while running **data_enrichment_cleaning.sh**:

- Get the batch id number from the batch file and generate the Log File for data enrichment_cleaning using the batch id. This will be data_enrichment_log_batch_1
- Start the spark shell with HBase jars,which are needed to connect to HBase, databricks jar which needed to parse csv data and running the **data_enrichment_cleaning.scala** file.

```
[acadgild@localhost project_3]$ sh /home/acadgild/project_3/scripts/data_enrichment_cleaning.sh
Ivy Default Cache set to: /home/acadgild/.ivy2/cache
The jars for the packages stored in: /home/acadgild/.ivy2/jars
:: loading settings :: url = jar:file:/home/acadgild/spark-2.2.0-bin-hadoop2.7/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysetti
ngs.xml
com.databricks#spark-csv_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
        confs: [default]
        found com.databricks#spark-csv_2.11;1.4.0 in central
        found org.apache.commons#commons-csv;1.1 in central
        found com.univocity#univocity-parsers;1.5.1 in central
:: resolution report :: resolve 2308ms :: artifacts dl 78ms
```

The following operations are performed in **data_enrichment_cleaning.scala**:

Exporting the lookup tables from HBase to Spark

- Using the HBase host details, connect to HBase database from spark.
- From sparkcontext, using select query, query the HBase table and columns family to fetch the table data and create a rdd using the table data. .
- Convert the rdd to dataframe and register as temporary table.
- Perform the step for all the 3 HBase tables and export into Spark dataframe

Exporting the HBase Tables into Spark

data_enrichment_cleaning.scala

```
import org.apache.spark._
import it.nerdammer.spark.hbase._
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.functions._
import com.databricks.spark.csv._
val sparkConf = new SparkConf().setAppName("Spark-HBase").setMaster("local[4]")
sparkConf.set("spark.hbase.host", "127.0.0.1")
val sc = new SparkContext(sparkConf)
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._

//Loading lookup tables into spark from HBase
//station_geo_map
val station_geo_map_table = sc.hbaseTable[(Option[String], Option[String])]("station-geo-map").select("geo_cd").inColumnFamily("geo")
val station_geo_map_data = station_geo_map_table.collect().toList
val station_geo_map_rdd =  station_geo_map_data.map(x=> ((x._1).get.toString,(x._2).get.toString))
val station_geo_map_df = station_geo_map_rdd.toDF().registerTempTable("station_geo_map_df")

//song_artist
val song_artist_map_table = sc.hbaseTable[(Option[String], Option[String])]("song-artist-map").select("artistid").inColumnFamily("artist")
val song_artist_map_data = song_artist_map_table.collect().toList
val song_artist_map_rdd =  song_artist_map_data.map(x=> ((x._1).get.toString,(x._2).get.toString))
case class Song_Artist(song_id: String, artist_id: String)
val song_artist_map_df = song_artist_map_rdd.toDF().registerTempTable("song_artist_map_df")

//subscribed_users
val subscribed_users_table = sc.hbaseTable[(Option[String],Option[String]
,Option[String])]("subscribed-users").select("startdt","enddt").inColumnFamily("subscn")
val subscribed_users_data = subscribed_users_table.collect()
case class Subsribed_Users(user_id: String, sub_start_ts: Long,sub_end_ts: Long)
val subscribed_users_rdd = spark.createDataset(subscribed_users_data.map(x=> Subsribed_Users((x._1).get.toString,(x._2).get.toLong,(x._3).get.toLong)))
val subscribed_users_df= subscribed_users_rdd.toDF()
subscribed_users_df.registerTempTable("subscribed_users_df")
```

HBase tables exported into spark

Song_Artist

```
scala> val song_artist_map_table = sc.hbaseTable[(Option[String], Option[String])]("song-artist-map").select("artistid").inColumnFamily(
"artist")
song_artist_map_table: it.nerdammer.spark.hbase.HBaseReaderBuilder[(Option[String], Option[String])] = HBaseReaderBuilder(org.apache.spa
rk.SparkContext@4e502162,song-artist-map,Some(artist),WrappedArray(artistid),None,None,List())

scala>

scala> val song_artist_map_data = song_artist_map_table.collect().toList
song_artist_map_data: List[(Option[String], Option[String])] = List((Some(S200),Some(A300)), (Some(S201),Some(A301)), (Some(S202),Some(A
302)), (Some(S203),Some(A303)), (Some(S204),Some(A304)), (Some(S205),Some(A301)), (Some(S206),Some(A302)), (Some(S207),Some(A303)), (Som
e(S208),Some(A304)), (Some(S209),Some(A305)))
```

Subsribed_users

```
scala> val subscribed_users_table = sc.hbaseTable[(Option[String],Option[String] ,Option[String])]("subscribed-users").select("startdt",
"enddt").inColumnFamily("subscn")
subscribed_users_table: it.nerdammer.spark.hbase.HBaseReaderBuilder[(Option[String], Option[String], Option[String])] = HBaseReaderBuild
er(org.apache.spark.SparkContext@4e502162,subscribed-users,Some(subscn),WrappedArray(startdt, enddt),None,None,List())

scala>

scala> val subscribed_users_data = subscribed_users_table.collect()
subscribed_users_data: Array[(Option[String], Option[String], Option[String])] = Array((Some(U100),Some(1465230523),Some(1465130523)), (
Some(U101),Some(1465230523),Some(1475130523)), (Some(U102),Some(1465230523),Some(1475130523)), (Some(U103),Some(1465230523),Some(1475130
523)), (Some(U104),Some(1465230523),Some(1475130523)), (Some(U105),Some(1465230523),Some(1475130523)), (Some(U106),Some(1465230523),Some
(1485130523)), (Some(U107),Some(1465230523),Some(1455130523)), (Some(U108),Some(1465230523),Some(1465230623)), (Some(U109),Some(14652305
23),Some(1475130523)), (Some(U110),Some(1465230523),Some(1475130523)), (Some(U111),Some(1465230523),Some(1475130523)), (Some(U112),Some(
1465230523),Some(1475130523)), (Some(U113),Some(1465230523),Some(1485130523)), (Some(U114),Some(1465230523),Some(1468...
scala>
```

Station_geo

```
scala> val station_geo_map_table = sc.hbaseTable[(Option[String], Option[String])]("station-geo-map").select("geo_cd").inColumnFamily("g
eo")
station_geo_map_table: it.nerdammer.spark.hbase.HBaseReaderBuilder[(Option[String], Option[String])] = HBaseReaderBuilder(org.apache.spa
rk.SparkContext@4e502162,station-geo-map,Some(geo),WrappedArray(geo_cd),None,None,List())
```

```
scala> val station_geo_map_rdd =  station_geo_map_data.map(x=> ((x._1).get.toString,(x._2).get.toString))
station_geo_map_rdd: List[(String, String)] = List((ST400,A), (ST401,AU), (ST402,AP), (ST403,J), (ST404,E), (ST405,A), (ST406,AU), (ST40
7,AP), (ST408,E), (ST409,E), (ST410,A), (ST411,A), (ST412,AP), (ST413,J), (ST414,E))
```

Data Enrichment

- Load the /user/acadgild/project_3/processed/formatted_input.csv (generated from data_formatting) into Spark, from HDFS.
- Create a datafarme to perform data enrichment. Check the formatted input data to find, if fields like Geo_cd and Artist_id are NULL or absent.
- If null or absent, consult the lookup table dataframes using fields Station_id and Song_id respectively to get the values of Geo_cd and Artist_id as below.

NULL or absent field	Look up field	Look up table (Table from which record can be updated)
Geo_cd	Station_id	Station_Geo_Map
Artist_id	Song_id	Song_Artist_Map

- After searching in the lookup tables dataframes, if corresponding entry is null, using spark UDF mark the rows in formatted_input.csv as invalid.

Loading the formatted Data from HDFS and Data Enrichment

data_enrichment_cleaning.scala

```
34  //Input the formatted_input_data
35  val formatted_input = spark.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load("
    hdfs://localhost:9000/user/acadgild/project_3/processed/formatted_input.csv")
36
37  //Data Enrichment ::: geo_cd null
38  def findGeo(station_id: String) = station_geo_map_rdd.filter(x=> ((x._1).equals(station_id))).take(1).map(x=>x._2)
39   def findGeo_Cd(geo_cd: String, station_id: String) :String={
40   if (geo_cd==null ||  geo_cd.isEmpty )
41    if((findGeo(station_id).head).isEmpty)
42     "invalid"
43     else findGeo(station_id).head
44   else geo_cd
45   }
46  val findGeo_Cd_udf = udf(findGeo_Cd(_:String,_:String))
47  val enrich_1 = formatted_input.withColumn("geo_cd", findGeo_Cd_udf($"geo_cd",$"station_id"))
48
49  //Data Enrichment :::  artist_id null
50   def findArtist(song_id: String) = song_artist_map_rdd.filter(x=> ((x._1).equals(song_id))).take(1).map(x=>x._2)
51   def findArtist_Id(artist_id: String, song_id: String) :String={
52   if (artist_id==null ||  artist_id.isEmpty )
53     if((findArtist(song_id).head).isEmpty)
54     "invalid"
55     else findArtist(song_id).head
56   else artist_id
57   }
58  val findArtist_Id_udf = udf(findArtist_Id(_:String,_:String))
59  val enrich_2 = enrich_1.withColumn("artist_id", findArtist_Id_udf($"artist_id",$"song_id"))
```

Data cleaning

- After referring to lookup table dataframes, to fill null or empty artist_id and geo cd, the enriched rdd may have some records with invalid geo_cd or artist_id.
- Using filter function, filter the enriched data by removing the invalid record(invalid geo_cd or artist_id or user_id null) and extract the cleaned data as below.
- Over the cleaned data, check if there is any record with date inconsistancies , that is having end data, which is earlier than the start date and fixed it.
- Store the cleaned data in the HDFS location hdfs://localhost:9000/user/acadgild/project_3/processed/cleaned_input.csv which will be used for analysis
- Store the HBase table subscribed users dataframe as CSV records which will also be used for analysis.

Data cleaning and storing in HDFS for Analysis

data_enrichment_cleaning.scala

```
81  //Data cleaning
82  //Searching and filtering datas where, user_id not null, artist_id not (invalid or null) & geo_cd not (invalid or null)
83  val cleaned_1 = enrich_2.filter(row=> (!( row.getString(0)==null ||  row.getString(0).isEmpty )&& !(row.getString(2).equals("invalid")) && !(row.getString(6).equals("invalid"))))
84
85  //Checking the start date and end date incostistances and fixing.
86  val cleaned_2 = (cleaned_1.withColumn("start_ts_corrected", when($"start_ts">$"end_ts", $"end_ts").otherwise($"start_ts")).withColumn("end_ts_corrected", when($"start_ts"<$"end_ts",
    $"end_ts").otherwise($"start_ts")))
87  val cleaned_3 = cleaned_2.select("user_id","song_id","artist_id","timestamp","start_ts_corrected","end_ts_corrected","geo_cd","station_id","song_end_type","like","dislike")
88  val cleaned_4 = cleaned_3.toDF()
89
90  //Storing the cleaned Music Data dataframe as CSV to be used for data_analysis
91  cleaned_4.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").save("hdfs://localhost:9000/user/acadgild/project_3/processed/cleaned_input.csv")
92
93  //Storing the subscribed_users dataframe as CSV for analysis
94  subscribed_users_df.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").save("hdfs://localhost:9000/user/acadgild/project_3/processed/subscribed_users.csv")
```

- The contents of cleaned_input.csv from HDFS is as below..

[acadgild@localhost project_3]$ hadoop fs -ls /user/acadgild/project_3/processed/cleaned_input.csv

Found 2 items

-rw-r--r-- 3 acadgild supergroup 0 2018-01-07 07:22 /user/acadgild/project_3/processed/cleaned_input.csv/_SUCCESS

-rw-r--r-- 3 acadgild supergroup 2486 2018-01-07 07:22 /user/acadgild/project_3/processed/cleaned_input.csv/part-00000-65f75a3a-8a7b-4d49-bc85-3cd2ade767f0-c000.csv

[acadgild@localhost project_3]$ hadoop fs -cat /user/acadgild/project_3/processed/cleaned_input.csv/part-00000-65f75a3a-8a7b-4d49-bc85-3cd2ade767f0-c000.csv

user_id,song_id,artist_id,timestamp,start_ts_corrected,end_ts_corrected,geo_cd,station_id,song_end_type,like,dislike
U114,S207,A303,1465130523,1465230523,1475130523,A,ST415,3,1,0
U107,S202,A303,1495130523,1465230523,1465230523,U,ST415,0,1,1
U100,S204,A302,1495130523,1465130523,1475130523,AU,ST408,2,1,1
U104,S202,A303,1465230523,1465130523,1475130523,A,ST409,2,0,1
U102,S207,A301,1465230523,1465230523,1485130523,AU,ST403,3,1,1
U106,S202,A302,1465230523,1465130523,1465130523,AU,ST408,0,1,1
U105,S207,A300,1465230523,1465130523,1485130523,U,ST400,2,0,1
U108,S205,A304,1465130523,1465130523,1475130523,A,ST410,2,1,0
U105,S203,A303,1475130523,1465130523,1465230523,AU,ST408,2,0,1
U110,S203,A300,1465230523,1465130523,1485130523,A,ST415,0,1,1
U113,S200,A303,1465230523,1465130523,1475130523,E,ST413,3,1,1
U119,S208,A302,1495130523,1465230523,1465230523,U,ST415,3,0,0
U118,S208,A303,1475130523,1465130523,1465230523,E,ST415,3,0,0
U107,S210,A302,1475130523,1485130523,1485130523,AP,ST404,2,1,0
U118,S202,A300,1495130523,1465230523,1465230523,AP,ST410,1,0,0
U111,S206,A305,1465130523,1465130523,1485130523,AU,ST415,0,1,1
U116,S208,A303,1465230523,1475130523,1485130523,A,ST413,1,0,1
U101,S202,A300,1465230523,1465130523,1475130523,U,ST401,0,0,1
U120,S206,A303,1495130523,1465130523,1485130523,AU,ST414,0,0,0
U106,S205,A300,1462863262,1462863262,1494297562,AP,ST407,2,1,1
U114,S209,A303,1465490556,1462863262,1494297562,U,ST411,2,1,0

U113,S203,A304,1465490556,1462863262,1465490556,U,ST405,0,0,1
U108,S200,A302,1468094889,1462863262,1468094889,U,ST414,0,0,1
U102,S203,A305,1465490556,1465490556,1494297562,U,ST404,2,0,0
U115,S200,A300,1465490556,1465490556,1494297562,AU,ST404,3,0,0
U111,S204,A300,1465490556,1465490556,1468094889,U,ST410,3,1,1
U120,S201,A300,1494297562,1465490556,1468094889,A,ST410,3,0,1
U113,S203,A303,1465490556,1465490556,1465490556,A,ST402,1,1,0
U109,S203,A304,1462863262,1468094889,1494297562,E,ST405,1,1,1
U110,S202,A303,1494297562,1468094889,1494297562,AU,ST402,2,1,0
U100,S200,A301,1494297562,1494297562,1494297562,AP,ST410,3,1,1
U101,S208,A300,1462863262,1462863262,1468094889,E,ST408,0,1,1
U106,S206,A300,1494297562,1462863262,1465490556,A,ST405,3,1,0
U107,S202,A304,1494297562,1462863262,1468094889,U,ST409,0,0,0
U103,S204,A300,1468094889,1465490556,1494297562,AU,ST411,2,1,0
U103,S202,A300,1465490556,1465490556,1465490556,A,ST415,2,1,1
U113,S203,A303,1462863262,1468094889,1494297562,U,ST408,2,0,0
U113,S204,A301,1494297562,1465490556,1494297562,E,ST415,3,0,1

- The contents of subscribed_users.csv from HDFS is as below..

[acadgild@localhost project_3]$ hadoop fs -ls /user/acadgild/project_3/processed/subscribed_users.csv

Found 2 items

-rw-r--r-- 3 acadgild supergroup 0 2018-01-07 07:22 /user/acadgild/project_3/processed/subscribed_users.csv/_SUCCESS

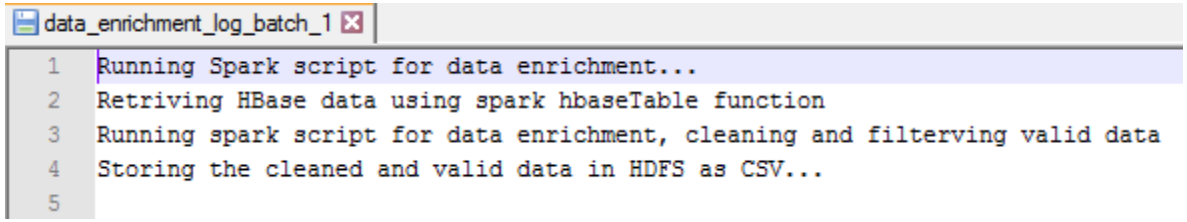
-rw-r--r-- 3 acadgild supergroup 437 2018-01-07 07:22 /user/acadgild/project_3/processed/subscribed_users.csv/part-00000-110ac3c1-43c7-4701-953e-dddf910b9be2-c000.csv

[acadgild@localhost project_3]$ hadoop fs -cat /user/acadgild/project_3/processed/subscribed_users.csv/part-00000-110ac3c1-43c7-4701-953e-dddf910b9be2-c000.csv

user_id,sub_start_ts,sub_end_ts
U100,1465230523,1465130523
U101,1465230523,1475130523

U102,1465230523,1475130523
U103,1465230523,1475130523
U104,1465230523,1475130523
U105,1465230523,1475130523
U106,1465230523,1485130523
U107,1465230523,1455130523
U108,1465230523,1465230623
U109,1465230523,1475130523
U110,1465230523,1475130523
U111,1465230523,1475130523
U112,1465230523,1475130523
U113,1465230523,1485130523
U114,1465230523,1468130523

- Once the command executed, the logmessage generated are as below.



Step 6: Perform Data Analysis (using Spark)

Perform Data Analysis

Below is the shell script **data_analysis.sh** is used to:.

- Load the cleaned_input CSV data, subsribed users CSV data from HDFS for data analysis
- Load the user-artist data from user-artist lookupfiles into Spark SQL table and create a dataframe.
- Write Spark SQL queries to find solution for the problem statements.

Executing the Data Analysis Script

data_analysis.sh

```
#!/bin/bash

batchid=`cat /home/acadgild/project_3/logs/current-batch.txt`
LOGFILE=/home/acadgild/project_3/logs/data_analysis_log_batch_$batchid

echo "Running spark script for data analysis..." >> $LOGFILE

cat /home/acadgild/project_3/scripts/data_analysis.scala | spark-shell  --jars "spark-xml_2.11-0.4.1.jar,spark-csv_2.11-1.4.0.jar"  --packages com.databricks:spark-xml_2.11:0.4.1   --packages com.databricks:spark-csv_2.11:1.4.0

echo "All Activities Complete..." >> $LOGFILE

echo "Incrementing batchid..." >> $LOGFILE

batchid=`expr $batchid + 1`
echo -n $batchid > /home/acadgild/project_3/logs/current-batch.txt
```

The following operations are performed while running **data_analysis.sh**:

- Get the batch id number from the batch file and generate the Log File for the batch for populate lookup process using the batch id. This will be data_analysis_log_batch_1
- Start the spark shell with databricks jar which needed to parse csv data and run the **data_analysis.scala** file.

```
[acadgild@localhost project_3]$ sh /home/acadgild/project_3/scripts/data_enrichment_cleaning.sh
Ivy Default Cache set to: /home/acadgild/.ivy2/cache
The jars for the packages stored in: /home/acadgild/.ivy2/jars
:: loading settings :: url = jar:file:/home/acadgild/spark-2.2.0-bin-hadoop2.7/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysetti
ngs.xml
com.databricks#spark-csv_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
        confs: [default]
        found com.databricks#spark-csv_2.11;1.4.0 in central
        found org.apache.commons#commons-csv;1.1 in central
        found com.univocity#univocity-parsers;1.5.1 in central
:: resolution report :: resolve 2308ms :: artifacts dl 78ms
```

The following operations are performed in **data_analysis.scala**:

- Load the enriched and cleaned CSV data from HDFS and create a dataframe.
- Load the subscribed_users data from HDFS and create a dataframe.
- Load the user Artist data from the lookupfile, into the spark and create dataframe

Loading the cleaned data and user artist data

data_analysis.scala

```
1  import org.apache.spark.sql.functions._
2  import com.databricks.spark.xml._
3  import com.databricks.spark.csv._
4
5  //Loading the enriched and cleaned input data
6  val cleaned_input = spark.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load("
   hdfs://localhost:9000/user/acadgild/project_3/processed/cleaned_input.csv")
7  val subscribed_users = spark.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load("
   hdfs://localhost:9000/user/acadgild/project_3/processed/subscribed_users.csv")
8  cleaned_input.registerTempTable("cleaned_input")
9  subscribed_users.registerTempTable("subscribed_users")
10
11 //user-artist lookup Loading
12 val user_artist_data= sc.textFile("/home/acadgild/project_3/lookupfiles/user-artist.txt")
13 case class User_Artist(user_id: String, artist_ids: Array[String])
14 val user_artist_rdd = user_artist_data.map(line => line.split(",")).map(rec=> (rec(0), (rec(1)split("&"))))
15 val user_artist_rdd2= user_artist_rdd.map(x => User_Artist((x._1), (x._2)))
16 val user_artist_df= user_artist_rdd2.toDF()
17 user_artist_df.registerTempTable("user_artist")
18
```

Data Analysis problem solution

data_analysis.scala

```
//Data Analysis
//Task 1
val top_10_stations = spark.sql("""SELECT station_id,count(DISTINCT song_id) AS song_count, count(DISTINCT user_id) AS dis_user FROM cleaned_input where like='1' GROUP by station_id ORDER BY song_count desc limit 10""")
//viewing the result and storing
top_10_stations.collect.foreach(println)
top_10_stations.rdd.saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/top_10_stations")

//Task 2
val task2_1 = spark.sql("SELECT T1.user_id AS user_id, T1.start_ts_corrected AS start_ts, T1.end_ts_corrected AS end_ts, T2.sub_start_ts AS sub_start_ts, T2.sub_end_ts AS sub_end_ts FROM  cleaned_input T1 LEFT JOIN  subscribed_users T2 ON T1.user_id == T2.user_id")
val task2_2 =task2_1.na.fill(0, Seq("sub_start_ts","sub_end_ts"))
 def findUser_Cat(user_id: String, start_ts: Long,end_ts: Long,sub_start_ts: Long,sub_end_ts: Long) :String={
 if (user_id==null ||  sub_end_ts==0 || sub_start_ts==0 )
   "unsubscribed"
  else if (end_ts>sub_end_ts)
   "unsubscribed"
 else "subscribed"
 }
val findUser_Cat_udf = udf(findUser_Cat(_:String,_:Long,_:Long,_:Long,_:Long))
val task2_3 = task2_2.withColumn("user_category", findUser_Cat_udf($"user_id",$"start_ts",$"end_ts",$"sub_start_ts",$"sub_end_ts"))
task2_3.collect.foreach(println)
 def findSong_duration(start_ts: Long,end_ts: Long) :Long={
  end_ts - start_ts
 }
 val findSong_duration_udf = udf(findSong_duration(_:Long,_:Long))
 val task2_4 = task2_3.withColumn("song_duration", findSong_duration_udf($"start_ts",$"end_ts"))
 task2_4.collect.foreach(println)
 task2_4.registerTempTable("task2_4")
val songs_liked = spark.sql("""SELECT user_category, SUM (song_duration) FROM task2_4 GROUP BY user_category""")
 //viewing the result and storing
songs_liked.collect.foreach(println)
songs_liked.rdd.saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/songs_liked")

```

```
50
51  //Task 3
52  val task3_1 = spark.sql("""SELECT T1.user_id AS user_id, T1.artist_id AS artist_id, T1.song_id AS song_id, T2.artist_ids AS followed_artists FROM  cleaned_input T1 LEFT JOIN
    user_artist T2 ON T1.user_id == T2.user_id""")
53  //Udf to find artist followed
54   def findArtist_followed(artist_id: String, followed_artists: scala.collection.mutable.WrappedArray[String]) :Int={
55   if(followed_artists!=null && followed_artists.contains(artist_id))
56   1
57   else 0
58   }
59  val findArtist_followed_udf = udf(findArtist_followed(_:String,_:scala.collection.mutable.WrappedArray[String]))
60  val task3_2 = task3_1.withColumn("artist_followed", findArtist_followed_udf($"artist_id",$"followed_artists"))
61  task3_2.registerTempTable("task3_2")
62  val maximum_followed= spark.sql("SELECT artist_id, count(DISTINCT user_id) AS dis_user FROM task3_2 GROUP by artist_id ORDER BY dis_user desc limit 10")
63  //viewing the result and storing
64  maximum_followed.collect.foreach(println)
65  maximum_followed.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/maximum_followed")
66
67  //Task 4:
68  val top_10_revenue = spark.sql("SELECT song_id,count(song_id) AS song_count FROM cleaned_input where like='1' OR song_end_type='0' GROUP by song_id ORDER BY song_count desc limit 10")
69  //viewing the result and storing
70  top_10_revenue.collect.foreach(println)
71  top_10_revenue.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/top_10_revenue")
72
73  //Task 5:
74  val top_10_unsubscribed = spark.sql("SELECT user_id,user_category,song_duration FROM task2_4 where user_category='unsubscribed' ORDER BY song_duration desc limit 10")
75  //viewing the result and storing
76  top_10_unsubscribed.collect.foreach(println)
77  top_10_unsubscribed.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/top_10_unsubscribed")
```

Problem Statement 1 :

Determine top 10 station_id(s) where maximum number of songs were played, which were liked by unique users.

Steps:

- Using select statement select station_id, count (distinct song_id) as song_count and count(distinct user_id) from the dataframe cleaned_input , where like has value 1, grouped by station_id and order by song_count in descending order and limiting the result to take top 10 records to determine top 10 station_id(s) where maximum number of songs were played, which were liked by unique users

Code:

```
scala> val top_10_stations = spark.sql("""SELECT station_id,count(DISTINCT song_id) AS song_count, count(DISTINCT user_id) AS dis_user F
ROM cleaned_input where like='1' GROUP by station_id ORDER BY song_count desc limit 10""")
top_10_stations: org.apache.spark.sql.DataFrame = [station_id: string, song_count: bigint ... 1 more field]

scala> //storing the result in HDFS

scala> top_10_stations.rdd.saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/top_10_stations")
```

Output in HDFS

```
[acadgild@localhost project_3]$ hadoop fs -ls hdfs://localhost:9000/user/acadgild/project_3/top_10_stations
Found 2 items
-rw-r--r--   3 acadgild supergroup          0 2018-01-07 09:23 hdfs://localhost:9000/user/acadgild/project_3/top_10_stations/_SUCCESS
-rw-r--r--   3 acadgild supergroup        120 2018-01-07 09:23 hdfs://localhost:9000/user/acadgild/project_3/top_10_stations/part-00000
[acadgild@localhost project_3]$ hadoop fs -cat hdfs://localhost:9000/user/acadgild/project_3/top_10_stations/part-00000
[ST415,4,5]
[ST410,3,3]
[ST408,3,3]
[ST402,2,2]
[ST405,2,2]
[ST411,2,2]
[ST404,1,1]
[ST403,1,1]
[ST407,1,1]
[ST413,1,1]
```

Problem Statement 2 :

Determine total duration of songs played by each type of user, where type of user can be 'subscribed' or 'unsubscribed'. An unsubscribed user is the one whose record is either not present in Subscribed_users lookup table or has subscription_end_date earlier than the timestamp of the song played by him.

Code:

```
//Task 2
val task2_1 = spark.sql("SELECT T1.user_id AS user_id, T1.start_ts_corrected AS start_ts, T1.end_ts_corrected AS end_ts, T2.sub_start_ts AS sub_start_ts, T2.sub_end_ts AS sub_end_ts
FROM  cleaned_input T1 LEFT JOIN  subscribed_users T2 ON T1.user_id == T2.user_id")
val task2_2 =task2_1.na.fill(0, Seq("sub_start_ts","sub_end_ts"))
 def findUser_Cat(user_id: String, start_ts: Long,end_ts: Long,sub_start_ts: Long,sub_end_ts: Long) :String={
 if (user_id==null ||  sub_end_ts==0 || sub_start_ts==0 )
   "unsubscribed"
  else if (end_ts>sub_end_ts)
   "unsubscribed"
 else "subscribed"
 }
val findUser_Cat_udf = udf(findUser_Cat(_:String,_:Long,_:Long,_:Long,_:Long))
val task2_3 = task2_2.withColumn("user_category", findUser_Cat_udf($"user_id",$"start_ts",$"end_ts",$"sub_start_ts",$"sub_end_ts"))
 def findSong_duration(start_ts: Long,end_ts: Long) :Long={
  end_ts - start_ts
 }
 val findSong_duration_udf = udf(findSong_duration(_:Long,_:Long))
 val task2_4 = task2_3.withColumn("song_duration", findSong_duration_udf($"start_ts",$"end_ts"))
 task2_4.registerTempTable("task2_4")
val songs_liked = spark.sql("""SELECT user_category, SUM (song_duration) FROM task2_4 GROUP BY user_category""")

//storing the result in HDFS
songs_liked.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/songs_liked")
```

Steps:

- Using select statement select user_id, start_time , end _ time from the dataframe cleaned_input, using left join join the dataframe subscribed_users and select start and endsubscription from subscribed_users , satisfying the condition user_id of cleaned_input = user_id of subscribed_users
- Find the user_category, if user is 'subscribed' or 'unsubscribed', using the condition **"**An unsubscribed user is the one whose record is either not present in Subscribed_users lookup table or has subscription_end_date earlier than the timestamp of the song played by him".
- Create a new column as 'user_category' to store if the user is subscribed or unsusbcribed.
- To find the user_category, write a spark UDF function. Take the user_id and song_end_date, subsrcripted_start_date and subsrcripted_end_date as input for the function, check if the user_id is null or sub_start or sub_end is 0 or song_end_date is greater than the subscription end , classify the user as unsubscribed who are all satisfying the above stated condition, else as subscribed.
- Write another UDF function to find the duration of the song played and add a new column as 'song_duration' to store.
- Then create a dataframe with newly added columns and create a temptable as task2_4
- Using Select statement, select user_category and sum(song_duration) and groupby user_category to total duration of songs played by each type of user .

Output in HDFS

```
[acadgild@localhost project_3]$ hadoop fs -ls hdfs://localhost:9000/user/acadgild/project_3/songs_liked
Found 2 items
-rw-r--r--   3 acadgild supergroup          0 2018-01-07 09:37 hdfs://localhost:9000/user/acadgild/project_3/songs_liked/_SUCCESS
-rw-r--r--   3 acadgild supergroup         47 2018-01-07 09:37 hdfs://localhost:9000/user/acadgild/project_3/songs_liked/part-00000
[acadgild@localhost project_3]$ hadoop fs -cat hdfs://localhost:9000/user/acadgild/project_3/songs_liked/part-00000
[unsubscribed,409672230]
[subscribed,43190548]
[acadgild@localhost project_3]$
```

Problem Statement 3 :

Determine top 10 connected artists. Connected artists are those whose songs are most listened by the unique users who follow them.

Code:

```
//Task 3
val task3_1 = spark.sql("""SELECT T1.user_id AS user_id, T1.artist_id AS artist_id, T1.song_id AS song_id, T2.artist_ids AS followed_artists FROM  cleaned_input T1 LEFT JOIN
user_artist T2 ON T1.user_id == T2.user_id""")
//Udf to find artist followed
 def findArtist_followed(artist_id: String, followed_artists: scala.collection.mutable.WrappedArray[String]) :Int={
 if(followed_artists!=null && followed_artists.contains(artist_id))
 1
 else 0
 }
val findArtist_followed_udf = udf(findArtist_followed(_:String,_:scala.collection.mutable.WrappedArray[String]))
val task3_2 = task3_1.withColumn("artist_followed", findArtist_followed_udf($"artist_id",$"followed_artists"))
task3_2.registerTempTable("task3_2")
val maximum_followed= spark.sql("SELECT artist_id, count(DISTINCT user_id) AS dis_user FROM task3_2 GROUP by artist_id ORDER BY dis_user desc limit 10")

//storing the result in HDFS
maximum_followed.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/maximum_followed")
```

Steps:

- Using select statement select user_id, artist_id, song_id from the dataframe cleaned_input , using left join joined the dataframe user_artist and select followed_artists from user_artist, satisfying the condition user_id of cleaned_input = user_id of user_artist
- Write a spark UDF which takes artist_id and followed artists as input, check if the the artist_id of cleaned_input is present in the followed artists list . If the list of followed artist_id's contain the artist_id, then return 1 else 0.
- Finally using SQL select, select the artist_id, count(distinct_user_id) and orderby distinct_user and limiting the result to take top 10 records to determine top 10 connected artists

Output in HDFS

```
[acadgild@localhost project_3]$ hadoop fs -ls hdfs://localhost:9000/user/acadgild/project_3/maximum_followed
Found 2 items
-rw-r--r--   3 acadgild supergroup          0 2018-01-07 09:24 hdfs://localhost:9000/user/acadgild/project_3/maximum_followed/_SUCCESS
-rw-r--r--   3 acadgild supergroup         54 2018-01-07 09:24 hdfs://localhost:9000/user/acadgild/project_3/maximum_followed/part-00000
[acadgild@localhost project_3]$ hadoop fs -cat  hdfs://localhost:9000/user/acadgild/project_3/maximum_followed/part-00000
[A300,9]
[A303,9]
[A302,5]
[A304,4]
[A301,3]
[A305,2]
```

Problem Statement 4 :

Determine top 10 songs who have generated the maximum revenue. Royalty applies to a song only if it was liked or was completed successfully or both

Steps:

- Using select statement select song_id, count song_id) as song_count from the dataframe cleaned_input , where like has value 1, or song_end_type has value 0, grouped by song_id and order by song_count in descending order, finally limiting the result to take top 10 records to determine top 10 songs which have generated the maximum revenue

Code:

```
scala> val top_10_revenue = spark.sql("SELECT song_id,count(song_id) AS song_count FROM cleaned_input where like='1' OR song_end_type='0
' GROUP by song_id ORDER BY song_count desc limit 10")
top_10_revenue: org.apache.spark.sql.DataFrame = [song_id: string, song_count: bigint]

scala> //storing the result in HDFS

scala> top_10_revenue.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/top_10_revenue")
```

Output in HDFS

```
[acadgild@localhost project_3]$ hadoop fs -ls hdfs://localhost:9000/user/acadgild/project_3/top_10_revenue
Found 2 items
-rw-r--r--   3 acadgild supergroup          0 2018-01-07 09:24 hdfs://localhost:9000/user/acadgild/project_3/top_10_revenue/_SUCCESS
-rw-r--r--   3 acadgild supergroup         90 2018-01-07 09:24 hdfs://localhost:9000/user/acadgild/project_3/top_10_revenue/part-00000
[acadgild@localhost project_3]$ hadoop fs -cat  hdfs://localhost:9000/user/acadgild/project_3/top_10_revenue/part-00000
[S202,6]
[S203,4]
[S204,3]
[S206,3]
[S200,3]
[S207,2]
[S205,2]
[S209,1]
[S208,1]
[S210,1]
```

Problem Statement 5 :

Determine top 10 unsubscribed users who listened to the songs for the longest duration.

Steps:

Using Spark SQL select, select the user_id, user_category and song_duration from task2_4 table where user_category is unsubscribed ,orderby Song_duration in descending and limiting the result to take top 10 records to determine top 10 unsubscribed users who listened to the songs for the longest duration

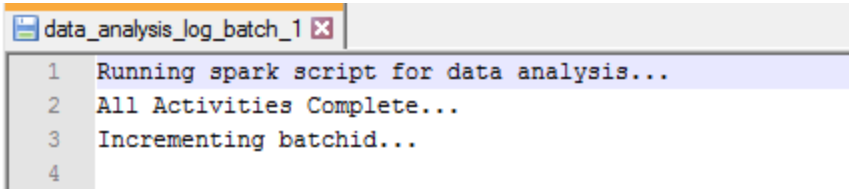
Code:

```
scala> val top_10_unsubscribed = spark.sql("SELECT user_id,user_category,song_duration FROM task2_4 where user_category='unsubscribed' O
RDER BY song_duration desc limit 10")
top_10_unsubscribed: org.apache.spark.sql.DataFrame = [user_id: string, user_category: string ... 1 more field]

scala> //storing the result in HDFS

scala> top_10_unsubscribed.rdd.coalesce(1).saveAsTextFile("hdfs://localhost:9000/user/acadgild/project_3/top_10_unsubscribed")
```

Output in HDFS

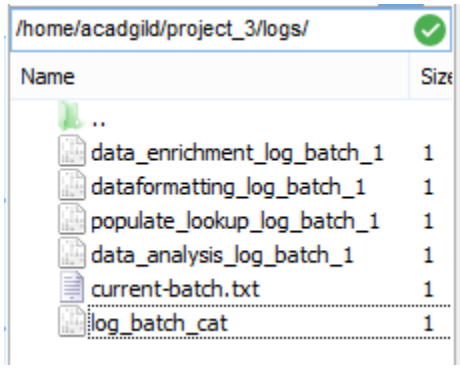
```
[acadgild@localhost project_3]$ hadoop fs -ls hdfs://localhost:9000/user/acadgild/project_3/top_10_unsubscribed
Found 2 items
-rw-r--r--   3 acadgild supergroup          0 2018-01-07 09:25 hdfs://localhost:9000/user/acadgild/project_3/top_10_unsubscribed/_SUCCES
S
-rw-r--r--   3 acadgild supergroup        290 2018-01-07 09:25 hdfs://localhost:9000/user/acadgild/project_3/top_10_unsubscribed/part-00
000
[acadgild@localhost project_3]$ hadoop fs -cat hdfs://localhost:9000/user/acadgild/project_3/top_10_unsubscribed/part-00000
[U106,unsubscribed,31434300]
[U114,unsubscribed,31434300]
[U103,unsubscribed,28807006]
[U102,unsubscribed,28807006]
[U115,unsubscribed,28807006]
[U113,unsubscribed,28807006]
[U109,unsubscribed,26202673]
[U110,unsubscribed,26202673]
[U113,unsubscribed,26202673]
[U105,unsubscribed,20000000]
```

- Once all the commands in the data_analysis.scala are executed, the logmessage generated are as below.



Step 7: Post Analysis

- The view of logs folder post analysis is



- The batchid is incremented from 1 to 2: