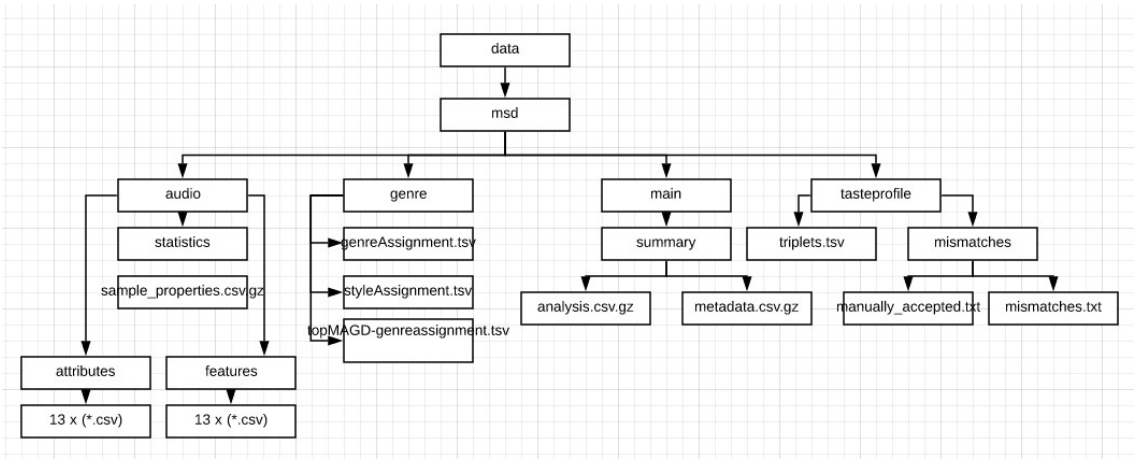
Ancy John
Student ID: 52770710

STAT420_Assignment 2

Data processing

Q1 Read through the documentation above and figure out how to read from each of the datasets. Make sure that you only read a subset of the larger datasets, so that you can develop your code efficiently without taking up too much of your time and the cluster resources.

(a) Give an overview of the structure of the datasets, including file formats, data types, and the expected level of parallelism that you can expect to achieve from HDFS.



Million song data set (msd) directory consists of 4 subdirectories and subsequent files in .csv, .csv.gs, .tsv, .tsv.gz, .txt formats. All files were loaded using inferSchema: true option. Using printSchema function, all data types were checked. The data types used are string, integer and double.

Ancy John
Student ID: 52770710

```
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd
Found 4 items
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:10 /data/msd/audio
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:17 /data/msd/genre
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:21 /data/msd/main
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:23 /data/msd/tasteprofile
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/main
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:22 /data/msd/main/summary
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/main/summary
Found 2 items
-rwxr-xr-x   8 hadoop supergroup   58658141 2019-05-06 14:21 /data/msd/main/summary/analysis.csv.gz
-rwxr-xr-x   8 hadoop supergroup  124211304 2019-05-06 14:22 /data/msd/main/summary/metadata.csv.gz
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/tasteprofile
Found 2 items
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:23 /data/msd/tasteprofile/mismatches
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:23 /data/msd/tasteprofile/triplets.tsv
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/tasteprofile/mismatches
Found 2 items
-rwxr-xr-x   8 hadoop supergroup      91342 2019-05-06 14:23 /data/msd/tasteprofile/mismatches/sid_matches_manually_accepted.txt
-rwxr-xr-x   8 hadoop supergroup    2026182 2019-05-06 14:23 /data/msd/tasteprofile/mismatches/sid_mismatches.txt
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ ^C
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/genre
Found 3 items
-rwxr-xr-x   8 hadoop supergroup   11625230 2019-05-06 14:17 /data/msd/genre/msd-MAGD-genreAssignment.tsv
-rwxr-xr-x   8 hadoop supergroup    8820054 2019-05-06 14:17 /data/msd/genre/msd-MASD-styleAssignment.tsv
-rwxr-xr-x   8 hadoop supergroup   11140605 2019-05-06 14:17 /data/msd/genre/msd-topMAGD-genreAssignment.tsv
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/audio
Found 3 items
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 13:52 /data/msd/audio/attributes
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:04 /data/msd/audio/features
drwxr-xr-x   - hadoop supergroup          0 2019-05-06 14:10 /data/msd/audio/statistics
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/audio/attributes
Found 13 items
-rwxr-xr-x   8 hadoop supergroup       1051 2019-05-06 13:52 /data/msd/audio/attributes/msd-jmir-area-of-moments-all-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup        671 2019-05-06 13:52 /data/msd/audio/attributes/msd-jmir-lpc-all-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup        484 2019-05-06 13:52 /data/msd/audio/attributes/msd-jmir-methods-of-moments-all-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup        898 2019-05-06 13:52 /data/msd/audio/attributes/msd-jmir-mfcc-all-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup        777 2019-05-06 13:52 /data/msd/audio/attributes/msd-jmir-spectral-all-all-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup        777 2019-05-06 13:52 /data/msd/audio/attributes/msd-jmir-spectral-derivatives-all-all-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup      12317 2019-05-06 13:52 /data/msd/audio/attributes/msd-marsyas-timbral-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup       9990 2019-05-06 13:52 /data/msd/audio/attributes/msd-mvd-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup       1390 2019-05-06 13:52 /data/msd/audio/attributes/msd-rh-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup      34913 2019-05-06 13:52 /data/msd/audio/attributes/msd-rp-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup       3942 2019-05-06 13:52 /data/msd/audio/attributes/msd-ssd-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup       9990 2019-05-06 13:52 /data/msd/audio/attributes/msd-trh-v1.0.attributes.csv
-rwxr-xr-x   8 hadoop supergroup      28313 2019-05-06 13:52 /data/msd/audio/attributes/msd-tssd-v1.0.attributes.csv
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls /data/msd/audio/features
Found 13 items
```

Parallelism on spark defaults to the number of cores on all machines (worker nodes/ executors). To increase parallelism of spark processing, we may increase the number of executors on the cluster. Partitioning in Spark is a technique to handle parallelism more efficiently. One partition is created for each block of the file in HDFS of size 64MB. Custom partitioning can be used to optimize performance and achieve parallelism. Cluster configuration and requirements of the application are the deciding factors. Number of partitions in an RDD should be in parallel to the number of cores in the cluster. This will assure parallel processing of all the partitions and the resources will be utilized in an optimal way. (ProjectPro)

(b) Look up the repartition method. Do you think this method will be useful?

Repartition method is used for custom partitioning of the data frame. It can either increase or decrease the number of partitions in a data frame. This method is useful to achieve an optimized performance of the system resources. Yet repartitioning is a fairly expensive full shuffle operation, in which whole data is taken out from existing partitions and equally distributed into newly formed partitions. That is data over all the partitions are equally populated.

If we are only decreasing the number of RDD partitions, spark has a less expensive and faster method, coalesce(), that avoids data movement. Using this method data is not equally distributed, that is partitions size varies.

(c) Count the number of rows in each of the datasets. How do the counts compare to the total number of unique songs?

In audio folder, we have the subdirectory attributes, features and statistics. Features consists of 13 other subdirectories. wc -lc command is used to count the number of rows for each of these directories.

Ancy John
Student ID: 52770710

```
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-area-of-moments-all-v1.0.csv/*.csv.gz | gunzip | wc -l
994623
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-lpc-all-v1.0.csv/*.csv.gz | gunzip | wc -l
994623
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-methods-of-moments-all-v1.0.csv/*.csv.gz | gunzip | wc -l
994623
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-mfcc-all-v1.0.csv/*.csv.gz | gunzip | wc -l
994623
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-spectral-all-all-v1.0.csv/*.csv.gz | gunzip | wc -l
994623
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-spectral-derivatives-all-all-v1.0.csv/*.csv.gz | gunzip | wc -l
994623
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-marsyas-timbral-v1.0.csv/*.csv.gz | gunzip | wc -l
995001
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-mvd-v1.0.csv/*.csv.gz | gunzip | wc -l
994188
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-rh-v1.0.csv/*.csv.gz | gunzip | wc -l
994188
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-rp-v1.0.csv/*.csv.gz | gunzip | wc -l
994188
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-ssd-v1.0.csv/*.csv.gz | gunzip | wc -l
994188
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-trh-v1.0.csv/*.csv.gz | gunzip | wc -l
994188
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-tssd-v1.0.csv/*.csv.gz | gunzip | wc -l
994188
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/statistics/*.csv.gz | gunzip | wc -l
992866
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/attributes/*.csv | wc -l
3929
```

Same command is used to find out the number of rows genre and main directories

```
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/genre/*.tsv | wc -l
1103077
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/main/summary/*.csv.gz | gunzip | wc -l
2000002
```

Tasteprofile directory consists of two subdirectories; triplets and mismatches. Number of rows are printed as below.

```
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/tasteprofile/triplets.tsv/*.tsv.gz | gunzip | wc -l
48373586
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/tasteprofile/mismatches/*.txt| wc -l
20032
```

To count distinct song ids, schema is defined for the triplets file and counted total number of unique songs.

```
In [42]: triplets.count()
Out[42]: 48373586

In [43]: from pyspark.sql.functions import countDistinct
    ...: triplets.agg(countDistinct('song_id')).show()
+----------------------+
|count(DISTINCT song_id)|
+----------------------+
|                384546|
+----------------------+
```

Q2 Complete the following data preprocessing.

(a) Filter the Taste Profile dataset to remove the songs which were mismatched.

Schemas where defined for the matches manually accepted file and mismatched text file and are removed from the triplet's data frame that we had defined previously. 2578475 songs were filtered out.

```
In [17]: print(triplets.count())
48373586

In [18]: print(triplets_not_mismatched.count())
45795111
```

Ancy John
Student ID: 52770710

(b) Load the audio feature attribute names and types from the audio/attributes directory and use them to define schemas for the audio features themselves. Note that the attribute files and feature datasets share the same prefix and that the attribute types are named consistently. Think about how you can automate the creation of StructType by mapping attribute types to pyspark.sql.types objects.

The awk command is used to load the attribute types along with sort and uniq commands. Later we mapped these attribute types to spark data types. We created a list object to hold or those dataset names and later we created a dictionary object to automate the creation of StructType and to define schemas for all sub datasets in the features directory.

```
[1]+  Stopped                 start_pyspark_shell
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat "/data/msd/audio/attributes/*" | awk -F',' '{print $2}' | sort | uniq
NUMERIC
real
real
string
string
STRING
[ajo139@canterbury.ac.nz@mathmadslinux1p ~]$ fg
start_pyspark_shell
In [42]: audio_attribute_type_mapping = {
    ...:     "NUMERIC": DoubleType(),
    ...:     "real": DoubleType(),
    ...:     "string": StringType(),
    ...:     "STRING": StringType()
    ...: }
```

```
In [47]: audio_dataset_schemas = {}
    ...: for audio_dataset_name in audio_dataset_names:
    ...:     print(audio_dataset_name)
    ...:
    ...:     audio_dataset_path = f"/scratch-network/courses/2020/DATA420-20S1/data/msd/audio/attributes/{audio_dataset_name}.attributes.csv"
    ...:     with open(audio_dataset_path, "r") as f:
    ...:         rows = [line.strip().split(",") for line in f.readlines()]
    ...:
    ...:     audio_dataset_schemas[audio_dataset_name] = StructType([
    ...:         StructField(row[0], audio_attribute_type_mapping[row[1]], True) for row in rows
    ...:     ])
    ...:
msd-jmir-area-of-moments-all-v1.0
msd-jmir-lpc-all-v1.0
msd-jmir-methods-of-moments-all-v1.0
msd-jmir-mfcc-all-v1.0
msd-jmir-spectral-all-all-v1.0
msd-jmir-spectral-derivatives-all-all-v1.0
msd-marsyas-timbral-v1.0
msd-mvd-v1.0
msd-rh-v1.0
msd-rp-v1.0
msd-ssd-v1.0
msd-trh-v1.0
msd-tssd-v1.0
```

```
In [44]: audio_dataset_schemas.keys()
Out[44]: dict_keys(['msd-jmir-area-of-moments-all-v1.0', 'msd-jmir-lpc-all-v1.0', 'msd-jmir-methods-of-moments-all-v1.0', 'msd-jmir-mfcc-all-v1.0', 'msd-jmir-spectral-all-all-v1.0', 'msd-jmir-spectral-derivatives-all-all-v1.0', 'msd-marsyas-timbral-v1.0', 'msd-mvd-v1.0', 'msd-rh-v1.0', 'msd-rp-v1.0', 'msd-ssd-v1.0', 'msd-trh-v1.0', 'msd-tssd-v1.0'])

In [45]: type(audio_dataset_schemas)
Out[45]: dict
```

Now, each of the data sets can be loaded using this automated schema definition.

```
In [41]: audio_dataset_name = "msd-jmir-area-of-moments-all-v1.0"
    ...: features = (
    ...:     spark.read.format("com.databricks.spark.csv")
    ...:     .option("header", "false")
    ...:     .option("delimiter", ",")
    ...:     .option("codec", "gzip")
    ...:     .schema(audio_dataset_schemas[audio_dataset_name])
    ...:     .load(f"hdfs:///data/msd/audio/features/{audio_dataset_name}.csv/")
    ...:     .cache()
    ...: )
    ...: features.take(1)
Out[41]: [Row(Area_Method_of_Moments_Overall_Standard_Deviation_1=1.2, Area_Method_of_Moments_Overall_Standard_Deviation_2=3355.0, Area_Method_of_Moments_Overall_Standard_Deviation_3=26270.0, Area_Method_of_Moments_Overall_Standard_Deviation_4=39850000.0, Area_Method_of_Moments_Overall_Standard_Deviation_5=309600000.0, Area_Method_of_Moments_Overall_Standard_Deviation_6=2403000000.0, Area_Method_of_Moments_Overall_Standard_Deviation_7=874900000000.0, Area_Method_of_Moments_Overall_Standard_Deviation_8=27750000000.0, Area_Method_of_Moments_Overall_Standard_Deviation_9=21550000000.0, Area_Method_of_Moments_Overall_Standard_Deviation_10=406400000000000.0, Area_Method_of_Moments_Overall_Average_1=3.201, Area_Method_of_Moments_Overall_Average_2=5746.0, Area_Method_of_Moments_Overall_Average_3=43470.0, Area_Method_of_Moments_Overall_Average_4=-44220000.0, Area_Method_of_Moments_Overall_Average_5=-337600000.0, Area_Method_of_Moments_Overall_Average_6=-2576000000.0, Area_Method_of_Moments_Overall_Average_7=766500000000.0, Area_Method_of_Moments_Overall_Average_8=3015000000.0, Area_Method_of_Moments_Overall_Average_9=23020000000.0, Area_Method_of_Moments_Overall_Average_10=3460000000000000.0, MSD_TRACKID="'TRHFHQZ12903C9E2D5'")]
```

Audio Similarity

Ancy John
Student ID: 52770710

Q1 There are multiple audio feature datasets, with different levels of detail. Pick one of the small datasets to get started.

(a) The audio features are continuous values, obtained using methods such as digital signal processing and psycho-acoustic modeling.

Produce descriptive statistics for each feature column in the dataset you picked. Are any features strongly correlated?

```
In [84]: features.select('Area_Method_of_Moments_Overall_Standard_Deviation_1',
    ...:                 'Area_Method_of_Moments_Overall_Standard_Deviation_2',
    ...:                 'Area_Method_of_Moments_Overall_Standard_Deviation_3').describe().show()
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|summary|Area_Method_of_Moments_Overall_Standard_Deviation_1|Area_Method_of_Moments_Overall_Standard_Deviation_2|Area_Method_of_Moments_Overall_Standard_Deviation_3|
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|  count|                                         994604|                                         994604|                                         994604|
|   mean|                          1.2289212402698894|                          5500.568385244177|                          33817.88670268569|
| stddev|                          0.5282405038550926|                          2366.02971695583|                          18228.64684359016|
|    min|                                            0.0|                                            0.0|                                            0.0|
|    max|                                          9.346|                                        46860.0|                                       699400.0|
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+


In [85]: features.select('Area_Method_of_Moments_Overall_Standard_Deviation_4',
    ...:                 'Area_Method_of_Moments_Overall_Standard_Deviation_5',
    ...:                 'Area_Method_of_Moments_Overall_Standard_Deviation_6').describe().show()
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|summary|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_5|Area_Method_of_Moments_Overall_Standard_Deviation_6|
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|  count|                                         994604|                                         994604|                                         994604|
|   mean|                  1.2766992610567312E8|                  7.8348260678150 83E8|                  5.2548302081411 42E9|
| stddev|                  2.3963850813687998E8|                  1.5825917070672674E9|                  1.2189329483371 60...|
|    min|                                            0.0|                                            0.0|                                            0.0|
|    max|                                       7.864E9|                                      8.124E10|                                      1.453E12|
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+


In [86]: features.select('Area_Method_of_Moments_Overall_Standard_Deviation_7',
    ...:                 'Area_Method_of_Moments_Overall_Standard_Deviation_8',
    ...:                 'Area_Method_of_Moments_Overall_Standard_Deviation_9').describe().show()
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|summary|Area_Method_of_Moments_Overall_Standard_Deviation_7|Area_Method_of_Moments_Overall_Standard_Deviation_8|Area_Method_of_Moments_Overall_Standard_Deviation_9|
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|  count|                                         994604|                                         994604|                                         994604|
|   mean|                  7.77060040891755 2E12|                  7.0365467679634 97E9|                  4.7220269470331 77E10|
| stddev|                  5.6916733926020 13E13|                  1.4246254033845 38...|                  1.0978759554853 38...|
|    min|                                            0.0|                                            0.0|                                            0.0|
|    max|                                       2.43E15|                                      7.46E11|                                      1.31E13|
+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
```

```
In [88]: features.select('Area_Method_of_Moments_Overall_Standard_Deviation_10',
    ...:                 'Area_Method_of_Moments_Overall_Average_1',
    ...:                 'Area_Method_of_Moments_Overall_Average_2',
    ...:                 'Area_Method_of_Moments_Overall_Average_3').describe().show()
+-------+------------------------------------------------+-----------------------------------------+-----------------------------------------+-----------------------------------------+
|summary|Area_Method_of_Moments_Overall_Standard_Deviation_10|Area_Method_of_Moments_Overall_Average_1|Area_Method_of_Moments_Overall_Average_2|Area_Method_of_Moments_Overall_Average_3|
+-------+------------------------------------------------+-----------------------------------------+-----------------------------------------+-----------------------------------------+
|  count|                                          994604|                                   994604|                                   994604|                                   994604|
|   mean|                  2.323934674341006...|                  3.5167568498729502|                  9476.016837603713|                  58331.702334634174|
| stddev|                  2.456502306380 96E16|                  1.8600935007021133|                  4088.5238979084525|                  31372.664773021694|
|    min|                                             0.0|                                      0.0|                                      0.0|                                      0.0|
|    max|                                        5.817E18|                                    26.52|                                  81350.0|                                 1003000.0|
+-------+------------------------------------------------+-----------------------------------------+-----------------------------------------+-----------------------------------------+


In [89]: features.select('Area_Method_of_Moments_Overall_Average_4',
    ...:                 'Area_Method_of_Moments_Overall_Average_5',
    ...:                 'Area_Method_of_Moments_Overall_Average_6',
    ...:                 'Area_Method_of_Moments_Overall_Average_7').describe().show()
+-------+-----------------------------------------+-----------------------------------------+-----------------------------------------+-----------------------------------------+
|summary|Area_Method_of_Moments_Overall_Average_4|Area_Method_of_Moments_Overall_Average_5|Area_Method_of_Moments_Overall_Average_6|Area_Method_of_Moments_Overall_Average_7|
+-------+-----------------------------------------+-----------------------------------------+-----------------------------------------+-----------------------------------------+
|  count|                                   994604|                                   994604|                                   994604|                                   994604|
|   mean|                  -1.42298338698354...|                  -8.7257056983715 31E8|                  -5.85721993285564E9|                  6.8357428593570 37E12|
| stddev|                  2.6710276301530796E8|                  1.7589081294252555E9|                  1.358639080848827...|                  5.0084743252237 75E13|
|    min|                                 -8.802E9|                                 -9.005E10|                                 -1.495E12|                                      0.0|
|    max|                                      0.0|                                      0.0|                                      0.0|                                 2.144E15|
+-------+-----------------------------------------+-----------------------------------------+-----------------------------------------+-----------------------------------------+


In [90]: features.select('Area_Method_of_Moments_Overall_Average_8',
    ...:                 'Area_Method_of_Moments_Overall_Average_9',
    ...:                 'Area_Method_of_Moments_Overall_Average_10').describe().show()
+-------+-----------------------------------------+-----------------------------------------+------------------------------------------+
|summary|Area_Method_of_Moments_Overall_Average_8|Area_Method_of_Moments_Overall_Average_9|Area_Method_of_Moments_Overall_Average_10|
+-------+-----------------------------------------+-----------------------------------------+------------------------------------------+
|  count|                                   994604|                                   994604|                                    994604|
|   mean|                  7.8284320771835 38E9|                  5.2590121574711 61E10|                  2.0437364618507 39E15|
| stddev|                  1.5832310311166 43...|                  1.2236963207338 80...|                  2.1631793599553 53...|
|    min|                                      0.0|                                      0.0|                                       0.0|
|    max|                                 8.335E11|                                 1.347E13|                                  4.958E18|
+-------+-----------------------------------------+-----------------------------------------+------------------------------------------+
```

Using panda data frame features:

Ancy John
Student ID: 52770710

```
In [10]: numeric_features = [t[0] for t in features.dtypes if t[1] == 'double']

In [11]: features.select(numeric_features).describe().toPandas().transpose()
Out[11]:
                                                    0                  1                  2          3          4
summary                                         count               mean             stddev        min        max
Area_Method_of_Moments_Overall_Standard_Deviati...  994604  1.2289212402699894  0.5282405038550926        0.0      9.346
Area_Method_of_Moments_Overall_Standard_Deviati...  994604   5500.568385244177   2366.02971695583        0.0    46860.0
Area_Method_of_Moments_Overall_Standard_Deviati...  994604   33817.88670268569   18228.64684359016        0.0   699400.0
Area_Method_of_Moments_Overall_Standard_Deviati...  994604  1.2766992610567312E8  2.3963850813687998E8        0.0   7.864E9
Area_Method_of_Moments_Overall_Standard_Deviati...  994604   7.834826067815083E8  1.5825917070672674E9        0.0  8.124E10
Area_Method_of_Moments_Overall_Standard_Deviati...  994604   5.254830208141142E9  1.2189329483371603E10       0.0  1.453E12
Area_Method_of_Moments_Overall_Standard_Deviati...  994604  7.770600408917552E12  5.691673392602013E13        0.0   2.43E15
Area_Method_of_Moments_Overall_Standard_Deviati...  994604   7.036546767963497E9  1.4246254033845385E10       0.0   7.46E11
Area_Method_of_Moments_Overall_Standard_Deviati...  994604  4.722026947033177E10  1.0978759554853384E11       0.0   1.31E13
Area_Method_of_Moments_Overall_Standard_Deviati...  994604  2.3239346743410065E15  2.456502306380 96E16        0.0  5.817E18
Area_Method_of_Moments_Overall_Average_1            994604     3.5167568498729502   1.8600935007021133        0.0     26.52
Area_Method_of_Moments_Overall_Average_2            994604      9476.016837603713   4088.5238979084525        0.0     81350.0
Area_Method_of_Moments_Overall_Average_3            994604      58331.702334634174   31372.664773021694        0.0   1003000.0
Area_Method_of_Moments_Overall_Average_4            994604  -1.4229833869835463E8  2.6710276301530796E8   -8.802E9        0.0
Area_Method_of_Moments_Overall_Average_5            994604   -8.725705698371531E8  1.7589081294252555E9   -9.005E10       0.0
Area_Method_of_Moments_Overall_Average_6            994604    -5.85721993285564E9  1.3586390808488277E10  -1.495E12       0.0
Area_Method_of_Moments_Overall_Average_7            994604   6.835742859357037E12  5.008474325223775E13        0.0   2.144E15
Area_Method_of_Moments_Overall_Average_8            994604   7.828432077183538E9  1.5832310311166437E10       0.0   8.335E11
Area_Method_of_Moments_Overall_Average_9            994604   5.259012157471161E10  1.2236963207338809E11       0.0   1.347E13
Area_Method_of_Moments_Overall_Average_10           994604   2.043736461850739E15  2.163179359955 3532E16      0.0   4.958E18
```

I have tried correlation method in spark, that returned an error. Later I converted spark data frame into a panda data frame to use the corr() in panda.

```
In [56]: import pandas as pd
    ...: corr_fe = corr_fe.toPandas()

In [57]: matrix = corr_fe.corr()

In [58]: print(matrix)
                                                    Area_Method_of_Moments_Overall_Standard_Deviation_1  ...  Area_Method_of_Moments_Overall_Average_10
Area_Method_of_Moments_Overall_Standard_Deviati...                                           1.000000    ...                                   0.044959
Area_Method_of_Moments_Overall_Standard_Deviati...                                          -0.019970    ...                                   0.499856
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.227683    ...                                   0.549286
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.013724    ...                                   0.704140
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.076598    ...                                   0.847423
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.105152    ...                                   0.922531
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.024142    ...                                   0.739312
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.076524    ...                                   0.848093
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.104952    ...                                   0.923090
Area_Method_of_Moments_Overall_Standard_Deviati...                                           0.045113    ...                                   0.999017
Area_Method_of_Moments_Overall_Average_1                                                     0.724618    ...                                   0.035360
Area_Method_of_Moments_Overall_Average_2                                                    -0.020833    ...                                   0.498142
Area_Method_of_Moments_Overall_Average_3                                                     0.226315    ...                                   0.549867
Area_Method_of_Moments_Overall_Average_4                                                    -0.013612    ...                                  -0.704031
Area_Method_of_Moments_Overall_Average_5                                                    -0.076529    ...                                  -0.849610
Area_Method_of_Moments_Overall_Average_6                                                    -0.104984    ...                                  -0.924358
Area_Method_of_Moments_Overall_Average_7                                                     0.024111    ...                                   0.739086
Area_Method_of_Moments_Overall_Average_8                                                     0.076398    ...                                   0.850349
Area_Method_of_Moments_Overall_Average_9                                                     0.104699    ...                                   0.924946
Area_Method_of_Moments_Overall_Average_10                                                    0.044959    ...                                   1.000000

[20 rows x 20 columns]
```

For a better view, I have changed the column names

```
In [62]: corr_fe.columns = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','0','p','q','r','s','t']

In [63]: matrix = corr_fe.corr() # correlatin matrix using panda dataframe
    ...: print(matrix)
          a         b         c         d         e         f         g         h         i  ...         l         m         n         0         p         q         r         s         t
a  1.000000 -0.019970  0.227683  0.013724  0.076598  0.105152  0.024142  0.076524  0.104952  ... -0.020833  0.226315 -0.013612 -0.076529 -0.104984  0.024111  0.076398  0.104699  0.044959
b -0.019970  1.000000  0.786974  0.848546  0.796064  0.699139  0.692396  0.795602  0.698605  ...  0.999567  0.782582 -0.848483 -0.794764 -0.695877  0.692130  0.794210  0.695302  0.499856
c  0.227683  0.786974  1.000000  0.681866  0.783786  0.794040  0.563798  0.783546  0.793502  ...  0.786758  0.994414 -0.681873 -0.783322 -0.791362  0.563609  0.782950  0.790680  0.549286
d  0.013724  0.848546  0.681866  1.000000  0.945989  0.837624  0.962426  0.945701  0.837212  ...  0.846566  0.675535 -0.999961 -0.944002 -0.833638  0.962245  0.943679  0.833230  0.704140
e  0.076598  0.796064  0.783786  0.945989  1.000000  0.964883  0.914948  0.999981  0.964623  ...  0.794339  0.779299 -0.946023 -0.999485 -0.962367  0.914810  0.999448  0.962095  0.847423
f  0.105152  0.699139  0.794040  0.837624  0.964883  1.000000  0.814394  0.965123  0.999981  ...  0.697522  0.791113 -0.837580 -0.965340 -0.998560  0.814197  0.965573  0.998509  0.922531
g  0.024142  0.692396  0.563798  0.962426  0.914948  0.814394  1.000000  0.914822  0.814140  ...  0.690353  0.558051 -0.962579 -0.913193 -0.811050  0.999984  0.913074  0.810831  0.739312
h  0.076524  0.795602  0.783546  0.945701  0.999981  0.965123  0.914822  1.000000  0.964899  ...  0.793878  0.779069 -0.945736 -0.999475 -0.962624  0.914684  0.999478  0.962391  0.848093
i  0.104952  0.698605  0.793502  0.837212  0.964623  0.999981  0.814140  0.964899  1.000000  ...  0.696988  0.790588 -0.837168 -0.965089 -0.998560  0.813943  0.965360  0.998550  0.923090
j  0.045113  0.501698  0.550790  0.706433  0.849023  0.923738  0.741348  0.849679  0.924281  ...  0.499883  0.549730 -0.706266 -0.850114 -0.923494  0.741079  0.850832  0.924060  0.999017
k  0.724618  0.026396  0.327493  0.009332  0.089381  0.129501  0.002544  0.089259  0.129195  ...  0.027717  0.328806 -0.009474 -0.089839 -0.129689  0.002573  0.089641  0.129273  0.035360
l -0.020833  0.999567  0.786758  0.846566  0.794339  0.697522  0.690353  0.793878  0.696988  ...  1.000000  0.783084 -0.846688 -0.793236 -0.694467  0.690170  0.792683  0.693892  0.498142
m  0.226315  0.782582  0.994414  0.675535  0.779299  0.791113  0.558051  0.779069  0.790588  ...  0.783084  1.000000 -0.675710 -0.781236 -0.792306  0.557946  0.780885  0.791638  0.549867
n -0.013612 -0.848483 -0.681873 -0.999961 -0.946023 -0.837580 -0.962579 -0.945736 -0.837168  ... -0.846688 -0.675710  1.000000  0.944124  0.833691 -0.962446 -0.943802 -0.833284 -0.704031
0 -0.076529 -0.794764 -0.783322 -0.944002 -0.999485 -0.965340 -0.913193 -0.999475 -0.965089  ... -0.793236 -0.781236  0.944124  1.000000  0.964460 -0.913111 -0.999978 -0.964202 -0.849610
p -0.104984 -0.695877 -0.791362 -0.833638 -0.962367 -0.998560 -0.811050 -0.962624 -0.998560  ... -0.694467 -0.792306  0.833691  0.964460  1.000000 -0.810918 -0.964722 -0.999978 -0.924358
q  0.024111  0.692130  0.563609  0.962245  0.914810  0.814197  0.999984  0.914684  0.813943  ...  0.690170  0.557946 -0.962446 -0.913111 -0.810918  1.000000  0.912992  0.810699  0.739086
r  0.076398  0.794210  0.782950  0.943679  0.999448  0.965573  0.913074  0.999478  0.965360  ...  0.792683  0.780885 -0.943802 -0.999978 -0.964722  0.912992  1.000000  0.964505  0.850349
s  0.104699  0.695302  0.790680  0.833230  0.962095  0.998509  0.810831  0.962391  0.998550  ...  0.693892  0.791638 -0.833284 -0.964202 -0.999978  0.810699  0.964505  1.000000  0.924946
t  0.044959  0.499856  0.549286  0.704140  0.847423  0.922531  0.739312  0.848093  0.923090  ...  0.498142  0.549867 -0.704031 -0.849610 -0.924358  0.739086  0.850349  0.924946  1.000000

[20 rows x 20 columns]
```

Apparently, many of the features are strongly correlated. A new data frame is created to list strongly correlated variables, using statistics, numpy and panda libraries

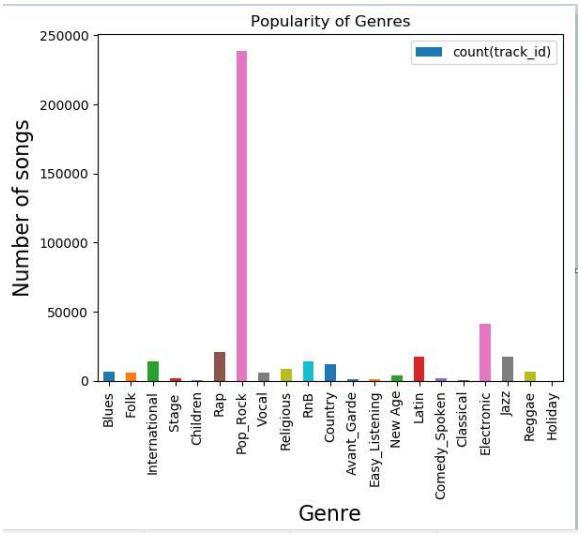
Ancy John
Student ID: 52770710

```
In [112]: dataframe_schema_strong_corr = sqlContext.createDataFrame(strong_corr_result, dataframe_schema)

In [113]: dataframe_schema_strong_corr.show(10,False)
+-----------------------------------------------+-----------------------------------------------+------------------+
|Variable_1                                     |Variable_2                                     |Value             |
+-----------------------------------------------+-----------------------------------------------+------------------+
|Area_Method_of_Moments_Overall_Standard_Deviation_2|Area_Method_of_Moments_Overall_Standard_Deviation_4|0.848546218206615 |
|Area_Method_of_Moments_Overall_Standard_Deviation_2|Area_Method_of_Moments_Overall_Average_2           |0.9995672225853743|
|Area_Method_of_Moments_Overall_Standard_Deviation_3|Area_Method_of_Moments_Overall_Average_3           |0.9944140706910988|
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_5|0.9459890988700506|
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_6|0.8376241688731361|
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_7|0.962425756873749 |
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_8|0.9457013829939959|
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_9|0.8372119298166416|
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Average_2           |0.8465660390164639|
|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Average_7           |0.962245146269548 |
+-----------------------------------------------+-----------------------------------------------+------------------+
only showing top 10 rows


In [114]: dataframe_schema_strong_corr.count()
Out[114]: 56
```

(b) Load the MSD All Music Genre Dataset (MAGD).

Visualize the distribution of genres for the songs that were matched.

```
In [2]: genre_schema = StructType([
   ...:     StructField("track_id", StringType(), True),
   ...:     StructField("genre", StringType(), True)
   ...: ])
   ...:
   ...: genre = (
   ...:     spark.read.format("csv")
   ...:     .option("header", "false")
   ...:     .option("delimiter", "\t")
   ...:     .option("codec", "gzip")
   ...:     .schema(genre_schema)
   ...:     .load("hdfs:///data/msd/genre/msd-topMAGD-genreAssignment.tsv/")
   ...:     .cache()
   ...: )

In [3]: genre.count()
Out[3]: 406427

In [4]: genre.show(5)
+------------------+--------+
|          track_id|   genre|
+------------------+--------+
|TRAAAAK128F9318786|Pop_Rock|
|TRAAAAV128F421A322|Pop_Rock|
|TRAAAAW128F429D538|     Rap|
|TRAAABD128F429CF47|Pop_Rock|
|TRAAACV128F423E09E|Pop_Rock|
+------------------+--------+
only showing top 5 rows
```



Ancy John
Student ID: 52770710

Genres for the songs that were matched is plotted using a user defined function.

(c) Merge the genres dataset and the audio features dataset so that every song has a label.

The features data set is left joined to the genre data set. The quotes on the track_ id variable is removed.

```
In [55]: features = features.withColumn('MSD_TRACKID', F.regexp_replace('MSD_TRACKID', "^'|'$", ''))
    ...: genre_audio = (
    ...:     features
    ...:     .join(genres_matched, features.MSD_TRACKID == genres_matched.track_id, how = 'left')
    ...:     .drop(genres_matched.track_id)
    ...: )

In [56]: genre_audio = genre_audio.where(col("genre").isNotNull())

In [57]: genre_audio.show(1)
+------------------------------------------------------------------------------------------------------------------------
+
|Area_Method_of_Moments_Overall_Standard_Deviation_1|Area_Method_of_Moments_Overall_Standard_Deviation_2|Area_Method_of_Moments_Overall_Standard_Deviation_3|Area_Method_of_Moments_Overall_Stan
dard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_5|Area_Method_of_Moments_Overall_Standard_Deviation_6|Area_Method_of_Moments_Overall_Standard_Deviation_7|Area_Method_of_Mome
nts_Overall_Standard_Deviation_8|Area_Method_of_Moments_Overall_Standard_Deviation_9|Area_Method_of_Moments_Overall_Standard_Deviation_10|Area_Method_of_Moments_Overall_Average_1|Area_Method_o
f_Moments_Overall_Average_2|Area_Method_of_Moments_Overall_Average_3|Area_Method_of_Moments_Overall_Average_4|Area_Method_of_Moments_Overall_Average_5|Area_Method_of_Moments_Overall_Average_6|
Area_Method_of_Moments_Overall_Average_7|Area_Method_of_Moments_Overall_Average_8|Area_Method_of_Moments_Overall_Average_9|Area_Method_of_Moments_Overall_Average_10|       MSD_TRACKID|   genre
|
+------------------------------------------------------------------------------------------------------------------------
+
|                                              1.087|                                             3359.0|                                            20300.0|
        4.005E7|                                            2.396E8|                                            1.433E9|                                           8.817E11|
               2.141E9|                                            1.281E10|                                           1.869E14|                                              2.482|
             5768.0|                                            33690.0|                                            -4.452E7|                                          -2.623E8|                                          -1.545E9|
                    7.736E11|                                           2.335E9|                                           1.377E10|                                       1.603E14|TRAAABD128F429CF47|Pop_Rock
|
+------------------------------------------------------------------------------------------------------------------------
+
only showing top 1 row
```

Q2 First you will develop a binary classification model.

(a) Research and choose three classification algorithms from the spark.ml library.

Justify your choice of algorithms, taking into account considerations such as explainability, interpretability, predictive accuracy, training speed, hyperparameter tuning, dimensionality, and issues with scaling.

Based on the descriptive statistics from Q1 part (a), decide what processing you should apply to the audio features before using them to train each model.

Classification is for dividing items into categories. I have chosen the Linear SVMs, Gradient-boosted tree classifier, and Logistic regression classification models for msd classification problem. Evaluation of the problem using versatile class of algorithms is a suggested step. Nature of each data set is different and suits differently for each base learner.

Logistic regression well appropriates for binary classification model. It is an extension of the linear regression model for classification problems. This is the most popular machine learning algorithm used for classification problems. Logistic is a probabilistic binary linear classifier

Linear support vector machines (Linear SVMs) is a standard method for large scale classification tasks. It is a linear method that supports only binary classification on spark. Linear SVMs are trained with L2 regularization by default, while spark.mlib supports L1 regularization alternatively. Regularizer encourage simple model and avoids overfitting. It has a meaningful stopping criterion that efficiently handles training error. (Kecman). Linear SVMs are non-probabilistic binary linear classifier.

- Linear SVM is linearly scalable (scales linearly with the size of the training data set)
- Efficient in dealing with extra-large data set

- It works for a high dimensional data in both sparse and dense format
- Extremely fast machine learning algorithm
- Superior performance when high accuracy is required
- Simple and easy to implement

Gradient Boosted Trees (GBTs) are ensembles of decision trees in order to minimize a loss function. Spark.mllib supports GBTs for binary classification and regression only. (Spark)

- GBTs do not require feature scaling
- Able to capture non-linearities and feature interactions
- Tuning parameters include loss, numIterations, learningRate, algo etc.
- runWithValidation method to prevent overfitting
- GBTs can be used in the field of learning to rank
- One of the most powerful technique for building predictive models

Data preprocessing

In machine learning, it is critical to determine the optimal features that are used to train the model. Feature engineering involves turning the features (inputs of interest) into n-dimensional feature vectors. (Kabii)

All features in our data are continuous, except the track_id. Since we do not have any categorical predictors, no need to deal with feature extraction. Sparse vectors (large number of zeroes) and the curse of dimensionality (more training data for more features to avoid overfitting) can be dealt with feature reduction.

We need to do category indexing for the response variable anyways and fitted using pipeline.

```
In [66]: from pyspark.ml.feature import StringIndexer
    ...: from pyspark.ml import Pipeline

In [67]: label_stringIdx = StringIndexer(inputCol = 'new_label', outputCol = 'label')
    ...: pipeline = Pipeline(stages=[label_stringIdx])

In [68]: pipelineFit = pipeline.fit(combined_df)
    ...: data = pipelineFit.transform(combined_df)

In [69]: data.show(2)
+--------------------+---------+-----+
|        features_pca|new_label|label|
+--------------------+---------+-----+
|[-122.13907793956...|  not_Rap|  0.0|
|[-100.93140974014...|  not_Rap|  0.0|
+--------------------+---------+-----+
only showing top 2 rows
```

First, we leave out variables that we are not interested in for our model. Thus, we will remove the track_id column from our data. We also remove genre columns with null values. I used na.drop() function to check for any missing values. It returned the same count that matched the previous count (with all values). This means no missing values in the data. Thus we do not have to deal with the imputation of the missing data either.

Ancy John
Student ID: 52770710

From the descriptive statistics in of the audio features data, we see attributes have a wide range of values, that needs to be normalized. Spark ML library makes it easy to do machine learning on big data scalable. We need to separate features from the target variable. We use map () and DenseVector() function to do this. A dense vector is used to store arrays of values for use in pyspark. We create a new data frame out of the input data and re-label the columns as 'label' and 'features'. Here we will pass a list as the second argument (Willems). Now we may use StandardScaler from ml library to standardize our features.

```
In [23]: from pyspark.ml.linalg import DenseVector

In [24]: input_data = df.rdd.map(lambda x: (x[20], DenseVector(x[:19])))
```

```
In [27]: df = spark.createDataFrame(input_data, ["label","features"])

In [28]: df.show(1)
+--------+--------------------+
|   label|            features|
+--------+--------------------+
|Pop_Rock|[1.087,3359.0,203...|
+--------+--------------------+
only showing top 1 row


In [29]: from pyspark.ml.feature import StandardScaler

In [30]: standardScaler = StandardScaler(inputCol="features", outputCol="features_scaled")

In [31]: scaler = standardScaler.fit(df)

In [32]: scaled_df = scaler.transform(df)
```

```
In [41]: scaled_df.show(2)
+--------------+--------------------+--------------------+
|         label|            features|     features_scaled|
+--------------+--------------------+--------------------+
|      Pop_Rock|[1.087,3359.0,203...|[2.05868112693496...|
|Easy_Listening|[0.7689,6721.0,35...|[1.45622807589723...|
+--------------+--------------------+--------------------+
only showing top 2 rows
```

Since we found out strong correlations between the predictor variables, we will apply feature reduction using PCA (k=2).

```
In [34]: from pyspark.ml.clustering import KMeans

In [35]: kmeans = KMeans(k=2, seed=1)

In [36]: from pyspark.ml.feature import PCA

In [37]: pca = PCA(k=2, inputCol='features_scaled', outputCol='features_pca')

In [38]: model = pca.fit(scaled_df)
2020-05-25 17:47:39 WARN  LAPACK:61 - Failed to load implementation from: com.github.fommil.netlib.NativeSystemLAPACK
2020-05-25 17:47:39 WARN  LAPACK:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefLAPACK

In [39]: reduced_df = model.transform(scaled_df).select('features_pca')
```

Ancy John
Student ID: 52770710

```
In [43]: reduced_df = model.transform(scaled_df).select('label','features_pca')

In [44]: reduced_df.show(2)
+--------------+--------------------+
|         label|        features_pca|
+--------------+--------------------+
|      Pop_Rock|[-1.5493311156046...|
|Easy_Listening|[-3.4535607684999...|
+--------------+--------------------+
only showing top 2 rows
```

(b) Convert the genre column into a column representing if the song is ”Rap” or some other genre as a binary label.

What is the class balance of the binary label?

I used a user defined function to generate a new binary label column.

```
In [46]: from pyspark.sql.functions import udf
    ...: def valueToBinary(value):    # user defined function
    ...:
    ...:    if   value == 'Rap': return 'Rap'
    ...:    else: return 'not_Rap'
    ...:

In [47]: udfValueToBinary = udf(valueToBinary, StringType())
```

```
In [49]: binary_df = reduced_df.withColumn("new_label", udfValueToBinary("label"))

In [50]: binary_df.show(2)
+--------------+--------------------+---------+
|         label|        features_pca|new_label|
+--------------+--------------------+---------+
|      Pop_Rock|[-1.5493311156046...|  not_Rap|
|Easy_Listening|[-3.4535607684999...|  not_Rap|
+--------------+--------------------+---------+
only showing top 2 rows


In [51]: binary_df = binary_df.drop('label')
```

If the classes are not distributed equally in the data set, there is class imbalance. Only 4% of the songs belongs to genre Rap and 96% belongs to the other genres. The ratio of the class imbalance is 19. That means for every Rap entry there are 19 not_Rap entries.

```
In [53]: binary_df.count()
Out[53]: 413277

In [54]: major_df = binary_df.filter(col("new_label") == 'not_Rap')
    ...: minor_df = binary_df.filter(col("new_label") == 'Rap')
    ...: ratio = int(major_df.count()/minor_df.count())
    ...: print("ratio: {}".format(ratio))
ratio: 19
```

Ancy John
Student ID: 52770710

c) Split the dataset into training and test sets. Note that you may need to take class balance into account using a sampling method such as stratification, subsampling, or oversampling. Justify your choice of sampling method.

Class imbalance can be addressed in a number of ways such as adding class weights, changing algorithms, random resampling etc.

Random resampling strategies are effective solutions to obtain a more balanced data distribution. I will consider under sampling with such large amount of data. Random deletion of examples in the majority class is called random under sampling. Resampling is not applied to the test dataset.

Dataset is split into train and test datasets. Now we apply under sampling on the train data. We used sample () method in spark to do this. First split the train data according to the class label. Delete rows of the majority class by the value of ratio. And then combined it with the minority class to form the new data frame.

```
In [59]: (train, test) = binary_df.randomSplit([0.7, 0.3], seed=100) # set seed for reproducibility
    ...: major_df = train.filter(col("new_label") == 'not_Rap')
    ...: minor_df = train.filter(col("new_label") == 'Rap')
    ...: ratio = int(major_df.count()/minor_df.count())
    ...: print("ratio: {}".format(ratio))
    ...:
ratio: 19

In [60]: sampled_majority_df = major_df.sample(False, 1/ratio)

In [61]: combined_df = sampled_majority_df.unionAll(minor_df)
```

Under sampling reduces overall size of the training data (Wan). Now ratio between the two class labels is 1.

```
In [63]: combined_df.count()
Out[63]: 28850
```

(d) Train each of the three classification algorithms that you chose in part (a).

Logistic regression model

```
In [83]: lrModel = lr.fit(combined_df)

In [84]: predictions = lrModel.transform(test)

In [85]: predictions.show()
+--------------------+-----+--------------------+--------------------+----------+
|        features_pca|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[-120.29868864720...|  0.0|[-3.1204212384303...|[0.04227271440719...|       1.0|
|[-114.38342544714...|  0.0|[0.14825028265568...|[0.53699483897168...|       0.0|
|[-107.58074185797...|  0.0|[-1.4285774397591...|[0.19332043237002...|       1.0|
|[-107.37747880621...|  0.0|[-2.2173907228807...|[0.09819963051930...|       1.0|
|[-100.64598425708...|  0.0|[0.04369947318506...|[0.51092313007747...|       0.0|
|[-99.323269878139...|  0.0|[-2.2313605994010...|[0.09696943289590...|       1.0|
|[-98.225742603191...|  0.0|[0.63362944068063...|[0.65331197072636...|       0.0|
|[-94.967074678366...|  0.0|[-1.3183075588664...|[0.21110000938939...|       1.0|
|[-91.724922504436...|  1.0|[-3.0479088382804...|[0.04530784081055...|       1.0|
|[-90.838459112737...|  0.0|[-0.8787911821043...|[0.29342833828562...|       1.0|
|[-86.214022467804...|  0.0|[0.60193902947182...|[0.64609980021493...|       0.0|
|[-85.316429605672...|  0.0|[-2.0361688810725...|[0.11545741668891...|       1.0|
|[-84.284792952600...|  1.0|[-1.5898546032070...|[0.16940435422725...|       1.0|
|[-82.065171346809...|  0.0|[-0.2342142750631...|[0.44171264047793...|       1.0|
|[-81.355171571805...|  0.0|[-2.2726525063320...|[0.09341333628736...|       1.0|
|[-80.413191253670...|  0.0|[0.55372451893446...|[0.63499927538395...|       0.0|
|[-78.861842545951...|  0.0|[-2.5726425431715...|[0.07091998845797...|       1.0|
|[-76.923808579755...|  0.0|[-2.2808903437676...|[0.09271802899231...|       1.0|
|[-75.236363268999...|  0.0|[-1.6985195387154...|[0.15465872076253...|       1.0|
|[-74.419379502902...|  0.0|[-0.2093210639392...|[0.44785997274776...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows


In [86]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
    ...: evaluator = BinaryClassificationEvaluator()
    ...: print('Test Area Under ROC', evaluator.evaluate(predictions))
Test Area Under ROC 0.6056965976517351
```

Ancy John
Student ID: 52770710

Area under ROC (AUC) for the logistic regression model is 60.56%

Gradient boosted tree classifier

```
In [90]: from pyspark.ml.classification import GBTClassifier

In [91]: gbt = GBTClassifier(labelCol="label", featuresCol="features_pca", maxIter=10)

In [92]: gbt = GBTClassifier(labelCol="label", featuresCol="features_pca", maxIter=10)
   ...: model = gbt.fit(combined_df)
   ...: predictions = model.transform(test)
   ...: predictions.show(5)
+--------------------+-----+--------------------+--------------------+----------+
|        features_pca|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[-120.29868864720...|  0.0|[-0.0850273265063...|[0.45758849424738...|       1.0|
|[-114.38342544714...|  0.0|[0.36647109480609...|[0.67545058348514...|       0.0|
|[-107.58074185797...|  0.0|[0.36647109480609...|[0.67545058348514...|       0.0|
|[-107.37747880621...|  0.0|[0.36647109480609...|[0.67545058348514...|       0.0|
|[-100.64598425708...|  0.0|[0.36647109480609...|[0.67545058348514...|       0.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 5 rows


In [93]: print('Test Area Under ROC', evaluator.evaluate(predictions))
Test Area Under ROC 0.6387756844115617
```

Area under ROC (AUC) for the Gradient Boosted Tree classifier model is 63.87%

Linear support vector machine

```
In [98]: from pyspark.ml.classification import LinearSVC
   ...: lsvc = LinearSVC(labelCol="label", featuresCol="features_pca", maxIter=10, regParam=0.1)
   ...: lsvcModel = lsvc.fit(combined_df)

In [99]: print("Coefficients: " + str(lsvcModel.coefficients))
   ...: print("Intercept: " + str(lsvcModel.intercept))
   ...:
Coefficients: [-0.029972084086953395,0.45615104729199607]
Intercept: -1.5716774212673112

In [100]: predictions = lsvcModel.transform(test)
    ...: predictions.show(5)
+--------------------+-----+--------------------+----------+
|        features_pca|label|       rawPrediction|prediction|
+--------------------+-----+--------------------+----------+
|[-120.29868864720...|  0.0|[-2.9759092102709...|       1.0|
|[-114.38342544714...|  0.0|[2.33927789803030...|       0.0|
|[-107.58074185797...|  0.0|[-0.4169985640852...|       1.0|
|[-107.37747880621...|  0.0|[-1.7321465010715...|       1.0|
|[-100.64598425708...|  0.0|[1.89207351746101...|       0.0|
+--------------------+-----+--------------------+----------+
only showing top 5 rows


In [101]: print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
    ...: print(lsvcModel)
Test Area Under ROC: 0.6043084124275537
LinearSVC_6c5ab49fba0b
```

Area under ROC (AUC) for the Linear Support Vector Machine model is 60.43%

(e) Use the test set to compute a range of performance metrics for each model, such as precision, accuracy, and recall.

Multiclass Classification Evaluator is imported to get the performance metrics for the models. Binary classification evaluator gives on the metrics, Area under ROC (AUC) and Area under PR.

Ancy John
Student ID: 52770710

Logistic regression

```
In [20]: evaluator3 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedPrecision")
    ...: evaluator4 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedRecall")
    ...: evaluator5 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="accuracy")
    ...: print('Precision', evaluator3.evaluate(predictions))
    ...: print('Recall', evaluator4.evaluate(predictions))
    ...: print('accuracy', evaluator5.evaluate(predictions))
Precision 0.9030994927860002
Recall 0.9503154701392587
accuracy 0.9503154701392587
```

Gradient boosted tree classifier

```
In [27]: from pyspark.ml.classification import GBTClassifier
    ...: gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
    ...: model = gbt.fit(train)
    ...: predictions = model.transform(test)

In [28]: evaluator3 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedPrecision")
    ...: evaluator4 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedRecall")
    ...: evaluator5 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="accuracy")
    ...: print('Precision', evaluator3.evaluate(predictions))
    ...: print('Recall', evaluator4.evaluate(predictions))
    ...: print('accuracy', evaluator5.evaluate(predictions))
Precision 0.9258627329363212
Recall 0.5047441545238902
accuracy 0.5047441545238902
```

Linear support vector machines

```
In [29]: from pyspark.ml.classification import LinearSVC
    ...: lsvc = LinearSVC(labelCol="label", featuresCol="features", maxIter=10, regParam=0.1)
    ...: lsvcModel = lsvc.fit(train)
    ...: predictions = lsvcModel.transform(test)

In [30]: evaluator3 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedPrecision")
    ...: evaluator4 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedRecall")
    ...: evaluator5 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="accuracy")
    ...: print('Precision', evaluator3.evaluate(predictions))
    ...: print('Recall', evaluator4.evaluate(predictions))
    ...: print('accuracy', evaluator5.evaluate(predictions))
Precision 0.9228541629508112
Recall 0.5116022010295138
accuracy 0.5116022010295138
```

(f) Discuss the relative performance of each model and of the classification algorithms overall, taking into account the performance metrics that you computed in part (e).

How does the class balance of the binary label affect the performance of the algorithms?

In terms of accuracy, logistic regression performs the best, with an accuracy of 95%. If we consider area under ROC, all three models performed average. Class imbalance of the model will lead to a biased result. A well-balanced class label can generate an unbiased result and better performance of the model.

Q3 Now you will tune the hyperparameters of your binary classification model.

(a) Look up the hyperparameters for each of your classification algorithms. Try to understand how these hyperparameters affect the performance of each model and if the values you used in Q2 part (d) were sensible.

Ancy John
Student ID: 52770710

Hyperparameters in machine learning is the term used to distinguish from the standard model parameters. A mathematical formula with a number of parameters that is used to learn from the data set is a machine learning model by definition. This is done through the process model training. In this process we fit the model parameters with the training data.

On the other hand, hyperparameters express high-level properties of the model that cannot be directly learned from the regular training process. These parameters are used to generate optimize results from the data. High-level properties imply the complexity, speed of performance, capacity to learn etc. These parameters need to be predefined and are decided by training different models and setting different values to generate the best result.

Logistic regression model:

1) maxIter: maximum number of iterations to run the optimization algorithm, >=0(default =100)
2) regParam: Regularization parameter, >=0 (default = 0)
3) elasticNetParam: A combination of L1 and L2 regularization in range [0,1] (default = 0)
4) aggregationDepth: suggested depth for tree aggregate, >=2 (default = 2)
5) fitIntercept: whether to fit an intercept term (default = True)
6) family: Description of the label distribution to be used in the model (default = auto) (Rai)

In our trained model we used, maxIter 10, regParam 0.3 and elasticNetParam 0.8 which are sensible.

Gradient boosted tree classifier:

1) maxDepth: maximum depth of a tree used to control overfitting
2) maxBins: maximum number of bins created by ordered splits
3) maxIter: maximum number of iterations

The trained model used the maxIter 10.

Linear support vector machine:

1) maxIter: maximum number of iterations to run the optimization algorithm, >=0(default =100)
2) regParam: Regularization parameter, >=0 (default = 0)

The trained model used the maxIter 10 and regParam 0.1.

(b) Use cross-validation to tune some of the hyperparameters of your best performing binary classification model.

How has this changed your performance metrics?

To my surprise my cross-validation model does not show any improvements in the performance. I have tried changing the paramGrid values. But it does not get any better.

To speed up the process, I have enabled parallelism. This argument is set to 8 (number of cores used). The total process took 50ms. 8 jobs in parallell and 10 tasks per each job.

Ancy John
Student ID: 52770710

Metrics from the model summary:

```
In [53]: accuracy = summary.accuracy
    ...: falsePositiveRate = summary.weightedFalsePositiveRate
    ...: truePositiveRate = summary.weightedTruePositiveRate
    ...: fMeasure = summary.weightedFMeasure()
    ...: precision = summary.weightedPrecision
    ...: recall = summary.weightedRecall
    ...: print("Accuracy: %s\nFPR: %s\nTPR: %s\nF-measure: %s\nPrecision: %s\nRecall: %s"
    ...:       % (accuracy, falsePositiveRate, truePositiveRate, fMeasure, precision, recall))
Accuracy: 0.5001734906315058
FPR: 0.49968836217648743
TPR: 0.500173490631506
F-measure: 0.3356343562964504
Precision: 0.5261491087768416
Recall: 0.500173490631506
```

Model predictions:

```
In [57]: # Evaluation of prediction
    ...: predictions = cvModel.transform(test)
    ...: evaluator1 = BinaryClassificationEvaluator(metricName="areaUnderROC")
    ...: evaluator2 = BinaryClassificationEvaluator(metricName="areaUnderPR")
    ...: evaluator3 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedPrecision")
    ...: evaluator4 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="weightedRecall")
    ...: evaluator5 = MulticlassClassificationEvaluator(predictionCol="prediction", metricName="accuracy")
    ...: print('Test Area Under ROC', evaluator1.evaluate(predictions))
    ...: print('Test Area Under PR', evaluator2.evaluate(predictions))
    ...: print('Precision', evaluator3.evaluate(predictions))
    ...: print('Recall', evaluator4.evaluate(predictions))
    ...: print('accuracy', evaluator5.evaluate(predictions))
Test Area Under ROC 0.6004778790443542
Test Area Under PR 0.05991443873520603
Precision 0.9083252181718435
Recall 0.05218569976279227
accuracy 0.052185699762792276
```

Area under ROC (AUC) for the cross validation model is 60%

Hypertune parameters for the best model:

```
In [58]: print('Best Param (regParam): ', bestmodel._java_obj.getRegParam())
    ...: print('Best Param (elasticNetParam): ', bestmodel._java_obj.getElasticNetParam())
    ...: print('Best Param (aggregationDepth): ', bestmodel._java_obj.getAggregationDepth())
    ...: print('Best Param (fitIntercept): ', bestmodel._java_obj.getFitIntercept())
    ...: print('Best Param (maxIter): ', bestmodel._java_obj.getMaxIter())
Best Param (regParam):  0.08
Best Param (elasticNetParam):  0.4
Best Param (aggregationDepth):  2
Best Param (fitIntercept):  False
Best Param (maxIter):  10
```

Q4 Next you will extend your work above to predict across all genres .

(a) Look up and explain the difference between the one vs. one and the one vs. all multiclass classification strategies. Do you expect one of these strategies to perform better than the other in general?

Some classification algorithms support multi-class classification. one-vs-one and one-vs-rest are two different strategies that shares a common approach for the multi-classification problems. A multi-

class classification dataset is split into multiple binary classification dataset and fits a binary classification model on each.

One-vs-one strategy splits a multi-class classification into one binary classification problem per class. This is a suggested method for support vector machines. Where as one-vs-rest strategy splits a multi-class classification into one binary classification problem per each pair of class. Eg. Logistic regression, Perceptron

(b) Choose one of your algorithms from Q2 that is capable of multiclass classification and performed well for binary classification as well.

Spark.ml does not support multi- class classification for the Gradient Boosted Tree (GBT) classifier and Linear SVMs. Logistic regression, that uses the one-vs-rest strategy can be used for multiclass classification.

(c) Convert the genre column into an integer index that encodes each genre consistently. Find a way to do this that requires the least amount of work by hand.

```
In [44]: from pyspark.ml.feature import StringIndexer
    ...: from pyspark.ml import Pipeline
    ...: df = (
    ...:     spark.read.format("parquet")
    ...:     .option("header", "true")
    ...:     .option("inferSchema", "true")
    ...:     .load("hdfs:///user/ajo139/outputs/msd/genre_audio.parquet")
    ...: )
    ...: df = df.drop('MSD_TRACKID')
    ...: label_stringIdx = StringIndexer(inputCol = 'genre', outputCol = 'label')
    ...: pipeline = Pipeline(stages=[label_stringIdx])
    ...: pipelineFit = pipeline.fit(df)
    ...: data = pipelineFit.transform(df)
    ...: data.select('genre', 'label').show(10)
    ...: data = data.drop('genre')
+--------------+-----+
|         genre|label|
+--------------+-----+
|      Pop_Rock|  0.0|
|Easy_Listening| 16.0|
|      Pop_Rock|  0.0|
|      Pop_Rock|  0.0|
|         Blues| 10.0|
|           RnB|  6.0|
|     Religious|  8.0|
|      Pop_Rock|  0.0|
|      Pop_Rock|  0.0|
|      Pop_Rock|  0.0|
+--------------+-----+
only showing top 10 rows
```

StringIndexer from the pyspark.ml library is used for index encoding for the genre column

(d) Split your dataset into training and test sets, train the classification algorithm that you chose in part (b), and compute a range of performance metrics that are relevant to multiclass classification. Make sure you take into account the class balance in your comments.

How has the performance of your model been affected by the inclusion of multiple genres?

First we split the data into train and test using pyspark.sql.window. Class imbalance of a multi-class classification problem can be handled in multiple ways.

Ancy John
Student ID: 52770710

1. Balance the training set
   - Oversample the minority class
   - Under sample the majority class
   - Synthesize new minority classes
2. Throw away minority classes and switch to an anomaly detection framework
3. Different strategies at the algorithm levels
   - Adjust the class weights
   - Adjust the decision threshold
   - Modifying an existing algorithm
   - Construct an entirely new algorithm (Faucett)

In our problem, I would consider under sampling the pop_Rock genre.

Data preprocessed through vectorization, PCA feature reduction (k=5) and string Indexing. All the stages are pipelined. Then, we split the data into train and test using pyspark.sql. window library. We did stratified random sampling that assumes observation independence. This is to ensure the splits partition the outcome with minimal sample variance.

Train data is under-sampled by the ratio. Logistic regression model is fitted with hyperparameter tuning and cross validation. K-fold validation and parallelism is set to 8; the number of cores we used.

```
In [31]: lr = LogisticRegression(family="multinomial", maxIter=100)
    ...: paramGrid = (ParamGridBuilder()
    ...:     .addGrid(lr.regParam, [0.1, 0.3, 0.5])
    ...:     .addGrid(lr.elasticNetParam, [0.6, 0.8, 1.0])
    ...:     .addGrid(lr.aggregationDepth,[2,5,10])
    ...:     .addGrid(lr.fitIntercept,[False, True])
    ...:     .build())
    ...:
    ...: cv = CrossValidator(
    ...:     estimator=lr,
    ...:     estimatorParamMaps=paramGrid,
    ...:     evaluator= evaluator_multiclass,
    ...:     numFolds=8,
    ...:     parallelism=8
    ...:     )

In [32]: cvModel = cv.fit(train)
2020-05-28 11:28:14 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
2020-05-28 11:28:14 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
                                                                                ]
In [33]: predictions = cvModel.transform(test)
    ...: performance_metrics_multiclass(predictions)
Accuracy: 0.09683428695852032
Precision: 0.00937687913076506
Recall: 0.09683428695852032
F1-score: 0.01709807806385555
```

The whole process is done in 55ms. 37 tasks in each of the 8 job ids. Prediction done with the test data. Accuracy of our model is 96.8%. Hyper tuning parameters for the best model are:

```
In [43]: print('Best Param (regParam): ', bestmodel._java_obj.getRegParam())
    ...: print('Best Param (elasticNetParam): ', bestmodel._java_obj.getElasticNetParam())
    ...: print('Best Param (aggregationDepth): ', bestmodel._java_obj.getAggregationDepth())
    ...: print('Best Param (fitIntercept): ', bestmodel._java_obj.getFitIntercept())
Best Param (regParam):  0.1
Best Param (elasticNetParam):  0.6
Best Param (aggregationDepth):  2
Best Param (fitIntercept):  True
```

Inclusion of multiclass (multiple genres) has improved the performance of the model.

Ancy John
Student ID: 52770710

Song recommendations

Q1 First it will be helpful to know more about the properties of the dataset before you being training the collaborative filtering model.

(a) How many unique songs are there in the dataset? How many unique users?

```
In [66]: from pyspark.sql.functions import countDistinct

In [67]: triplets_matched.agg(countDistinct('song_id')).show()
+------------------------+
|count(DISTINCT song_id)|
+------------------------+
|                  378310|
+------------------------+


In [68]: triplets_matched.agg(countDistinct('user_id')).show()
+------------------------+
|count(DISTINCT user_id)|
+------------------------+
|                 1019318|
+------------------------+
```

After removing all mismatches, there are 378310 unique songs and 1019318 unique users are there.

(b) How many different songs has the most active user played?

What is this as a percentage of the total number of unique songs in the dataset?

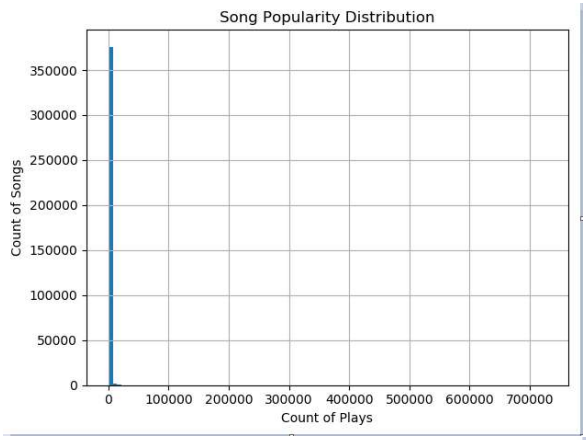
```
In [69]: triplets_matched.groupby("user_id").agg(sum('plays').alias("Sum")).sort(desc("Sum")).show(1,False)
+----------------------------------------+-----+
|user_id                                 |Sum  |
+----------------------------------------+-----+
|093cb74eb3c517c5179ae24caf0ebec51b24d2a2|13074|
+----------------------------------------+-----+
only showing top 1 row


In [70]: triplets_matched.filter('user_id == "093cb74eb3c517c5179ae24caf0ebec51b24d2a2"').agg(countDistinct('song_id')).show()
+----------------------+
|count(DISTINCT song_id)|
+----------------------+
|                   195|
+----------------------+


In [71]: (195/378310)*100
Out[71]: 0.05154502920884989
```

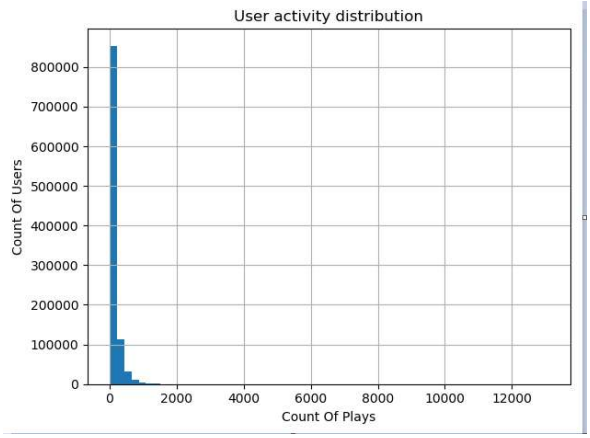
The most active user had played 195 different songs which is 5.15% of the total number of unique songs in the dataset.

(c) Visualize the distribution of song popularity and the distribution of user activity.

What is the shape of these distributions?

Ancy John
Student ID: 52770710





Both distributions are right skewed. Only popular songs are played in a significant amount. Majority of the songs have a smaller number of plays. Similarly, majority of the users exhibits a play count less than 100.

(d) Collaborative filtering determines similar users and songs based on their combined play history. Songs which have been played only a few times and users who have only listened to a few songs will not contribute much information to the overall dataset and are unlikely to be recommended.

Create a clean dataset of user-song plays by removing songs which have been played less than N times and users who have listened to fewer than M songs in total. Choose sensible values for N and M and justify your choices, taking into account (a) and (b).

To avoid data sparsity issue, songs that have been played less than 18 (40th percentile) and users who listen songs less than 26 (20th percentile) are removed and inner joined with triplet dataset.

```
In [25]: song_popularity.approxQuantile("Total_Play_Count", [0.40], 0)
Out[25]: [18.0]

In [26]: user_activity.approxQuantile("Total_Play_Count", [0.40], 0)
Out[26]: [51.0]

In [27]: user_activity.approxQuantile("Total_Play_Count", [0.20], 0)
Out[27]: [26.0]
```

Ancy John
Student ID: 52770710

```
In [28]: song_popularity_filter = song_popularity.filter(F.col("total_play_count") > 18)
    ...: song_popularity_filter.cache()
    ...: song_popularity_filter.count()
Out[28]: 226826

In [29]: user_activity_filter = user_activity.filter(F.col("total_play_count") > 26)
    ...: user_activity_filter.cache()
    ...: user_activity_filter.count()
Out[29]: 815287

In [30]: df = df.select(['user_id', 'song_id', 'plays']).join(song_popularity_filter.select('song_id'),on='song_id',how='inner')

In [31]: df = df.select(['user_id', 'song_id', 'plays']).join(user_activity_filter.select('user_id'),on='user_id',how='inner')

In [32]: df.count()
Out[32]: 42487946
```

(e) Split the user-song plays into training and test sets. Make sure that the test set contains at least 20% of the plays in total.

Note that due to the nature of the collaborative filtering model, you must ensure that every user in the test set has some user-song plays in the training set as well. Explain why this is required and how you have done this while keeping the selection as random as possible.

Preprocessed data is split into test and train. Subtract method is used to find the users in the test data but not in the train. 2 users in the test data are not there in the train data. These users where extracted and added to the train data using unionAll method. This is to avoid nominal behaviors while testing.

```
In [14]: df = test.join(train, on="user_id", how="left_anti")

In [15]: df.show()
+--------------------+------------------+-----+--------+--------+
|             user_id|           song_id|plays|user_int|song_int|
+--------------------+------------------+-----+--------+--------+
|41090c37030771372...|SOVSTIW12A8AE46D72|    4|  815285|  188656|
|8a14299cd2a8e2c71...|SOBRDBW12A81C21AE3|    3|  815282|  206853|
|8a14299cd2a8e2c71...|SOXPZEZ12A8AE48AC9|    1|  815282|    1718|
+--------------------+------------------+-----+--------+--------+


In [16]: test.count()
Out[16]: 12747212

In [17]: test = test.join(drop_users, on="user_id", how="left_anti")

In [18]: test.count()
Out[18]: 12747209

In [19]: train.count()
Out[19]: 29740734

In [20]: train = train.unionAll(df)

In [21]: train.count()
Out[21]: 29740737
```

Q2 Next you will train the collaborative filtering model.

(a) Use the spark.ml library to train an implicit matrix factorization model using Alternating Least

ALS model is trained and fitted on the test data.

Ancy John
Student ID: 52770710

```
In [22]: from pyspark.ml.recommendation import ALS
    ...: from pyspark.ml.evaluation import RegressionEvaluator
    ...: from pyspark.mllib.evaluation import RankingMetrics

In [23]: als = ALS(implicitPrefs=True, userCol="user_int", itemCol="song_int", ratingCol="plays", coldStartStrategy="drop")
    ...: alsModel = als.fit(train)
    ...: predictions = alsModel.transform(test)
    ...: predictions.show()
[Stage 200:>                                    (0 + 10) / 10]2020-05-29 18:24:35 WARN  BLAS:61 - Failed t
s
2020-05-29 18:24:35 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
+--------------------+------------------+-----+--------+--------+----------+
|             user_id|           song_id|plays|user_int|song_int|prediction|
+--------------------+------------------+-----+--------+--------+----------+
|c1255748c06ee3f64...|SOKLRPJ12A8C13C3FE|    7|       3|      12| 0.8773815|
|0c65e8f1e713e01dc...|SOKLRPJ12A8C13C3FE|    2|     530|      12|0.18401666|
|0252acdea2a493da2...|SOKLRPJ12A8C13C3FE|    3|    1024|      12| 0.7736027|
|7dcb1ac4193b55603...|SOKLRPJ12A8C13C3FE|    2|    1357|      12|  0.172444|
|7799610a32b69b348...|SOKLRPJ12A8C13C3FE|    4|    1516|      12|   0.94318|
|9b6d0c10a150a02c2...|SOKLRPJ12A8C13C3FE|    5|    2372|      12| 0.3271227|
|103ee21d642c7dbcd...|SOKLRPJ12A8C13C3FE|    2|    2546|      12| 0.7427625|
|922a4e1a909444aeb...|SOKLRPJ12A8C13C3FE|    1|    3487|      12|0.40288004|
|ed6d7950b4a5aff67...|SOKLRPJ12A8C13C3FE|    1|    3551|      12|0.03347722|
|b3cc48b4cbcf9320e...|SOKLRPJ12A8C13C3FE|    1|    4684|      12|0.15047827|
|106df95462fa708d9...|SOKLRPJ12A8C13C3FE|    1|    4863|      12| 0.7173749|
|4a07e558555d32f86...|SOKLRPJ12A8C13C3FE|    2|    5332|      12|0.98133534|
|a0ac3b84be35f66ad...|SOKLRPJ12A8C13C3FE|    1|    6878|      12| 0.2509492|
|35164e884de868e82...|SOKLRPJ12A8C13C3FE|    6|    7103|      12|0.79634964|
|23cb7ebc9f72af75a...|SOKLRPJ12A8C13C3FE|    2|    7740|      12| 0.7948094|
|dea654e8c0fcf435e...|SOKLRPJ12A8C13C3FE|    1|    8464|      12|0.19647422|
|d7234a231280639e4...|SOKLRPJ12A8C13C3FE|    1|    8643|      12|0.07269969|
|4227bba8d12fe379f...|SOKLRPJ12A8C13C3FE|    1|    8779|      12|  0.547008|
|bee26a889a15e3148...|SOKLRPJ12A8C13C3FE|    1|    9632|      12|0.15753794|
|03a6628a1ead4d62c...|SOKLRPJ12A8C13C3FE|    1|   10371|      12|0.31649387|
+--------------------+------------------+-----+--------+--------+----------+
only showing top 20 rows
```

RMSE value is 7.2

```
In [24]: evaluator = RegressionEvaluator(metricName="rmse", labelCol="plays", predictionCol="prediction")
    ...: rmse = evaluator.evaluate(predictions)
    ...:
    ...: print("Root-mean-square error = " + str(rmse))
Root-mean-square error = 7.2185633735710635
```

(b) Select a few of the users from the test set by hand and use the model to generate some recommendations. Compare these recommendations to the songs the user has actually played. Comment on the effectiveness of the collaborative filtering model.

Subset of users is selected and compared with the ALS model prediction. The user that I have compared had listened 2 songs which are actually recommended for him. Yet most of the predictions differs from what the user had actually played. This shows that the performance of the model is average.

```
In [53]: test.filter('user_int == 21338').show(1,False)
+--------+------------------------------------------------------------------------------------
|user_int|collect_list(song_int)
                                                                  |
+--------+------------------------------------------------------------------------------------
|21338   |[47036, 69349, 5801, 3998, 6467, 28659, 13343, 1207, 70325, 20747, 20628, 15443, 5593, 77214, 11026, 5115, 7497, 263, 22051, 22187, 36989, 8057, 10787, 59673, 641, 273, 152454, 5745,
 723, 9691, 3232, 396, 1022, 20907, 29568, 14769, 36039, 1507, 13456, 1150, 5978, 4466, 22400, 12056, 8109, 16076, 1390, 5559, 71294, 19991, 18172, 22302, 23260, 13673, 2931, 31347, 3383, 3246
0, 254, 2421, 10059, 705, 45717, 1543, 34621, 55196, 17555, 6686]|
+--------+------------------------------------------------------------------------------------

In [54]: temp.filter('user_int == 21338').show(1,False)
+--------+-------------------------------------------------+
|user_int|recommends                                       |
+--------+-------------------------------------------------+
|21338   |[254, 263, 192, 75, 175, 243, 195, 246, 200, 164]|
+--------+-------------------------------------------------+
```

(c) Use the test set of user-song plays and recommendations from the collaborative filtering model to compute the following metrics

• Precision @ 5 • NDCG @ 10 • Mean Average Precision (MAP) Look up these metrics and explain why they are useful in evaluating the collaborate filtering model. Explore the limitations of these metrics in evaluating a recommendation system in general. Suggest an alternative method for comparing two recommendation systems in the real world.

Assuming that you could measure future user-song plays based on your recommendations, what other metrics could be useful?

Ancy John
Student ID: 52770710

```
In [59]: print("Precision@5 :" + str(metrics.precisionAt(5)))
    ...: print("NDCG@10 :" + str(metrics.ndcgAt(10)))
    ...: print("MAP :" + str(metrics.meanAveragePrecision))
Precision@5 :0.047261146496815294
NDCG@10 :0.04773034587718222
MAP :0.01242504531329912
```

Q3 The method used to train the collaborative filtering model above is one of many.

(a) Explain the limitations of using this model to generate recommendations for a streaming music service such as Spotify. In what situations would this model be unable to produce recommendations for a specific user and why?
In what situations would this model produce low quality recommendations and why?

Spotify is a media service provider and music streaming service. The system works on both explicit and implicit feedbacks. ALS model is associated with implicit feedbacks. A case study of matrix factorization with Spotify revealed certain problems.

- Baseline prediction for an unknown rating (cold start problem)
- Users preferences may change
- Movies are more popular at certain times (Time sensitive baseline predictor)
- Users baseline rating may change
- Low quality recommendation on highly sparse data
- Popularity bias

Cold start problem

When the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information, it is a cold start problem. In this situation, the model is unable to produce recommendations for a specific user (wikipedia).

Sparse data and popularity bias

The model produces low quality recommendations, when music recommendation (streaming data) involves, modeling highly sparse data. Popularity bias, when vast majority of ratings are associated with the popular tracks is another situation where the model performance degrades.

(b) Suggest two other recommendation systems that could be used in each of these scenarios respectively.

Collaborative filtering algorithm Singular Value Decomposition (SVD) can handle massive dataset, sparseness of rating matrix, scalability and cold start problem. Content based recommendation system, based on user profiles is an alternative solution to cold start problem.

Item-item collaborative filtering for recommender system is not susceptible to popularity bias and so is a solution to handle this problem. This method is invented and used by Amazone.

(c) Based on the song metadata provided in the million song dataset summary, is there any other business logic that could be applied to the recommendations to improve the real world performance of the system?

Ancy John
Student ID: 52770710

The whole recommendation system is based on the taste profile of the users. The list of similar artists associated with each track can be utilized. The users would get an improved experience by listening to unknown talents. Also, it could be helpful to provide a platform for the new artists.

References

Faucett, Tom. *Learning from Imbalanced Classes*. 25 August 2016. online. 28 May 2020. <http://www.svds.com/learning-imbalanced-classes/>.

Kabii, David. *PySpark Feature Engineering and High Dimensional Data Visualization with Spark SQL in an Hour*. 30 July 2019. online. 25 May 2020. <https://medium.com/@david.kabii/pyspark-feature-engineering-and-high-dimensional-data-visualization-with-spark-sql-in-an-hour-b2fea0de2472>.

Kecman, Te-Ming Huang & Vojislav. *LinearSVM*. 2009. online. 22 May 2020. <http://www.linearsvm.com/>.

ProjectPro. *How Data Partitioning in Spark helps achieve more parallelism?* 07 May 2017. online. 20 May 2020. <https://www.dezyre.com/article/how-data-partitioning-in-spark-helps-achieve-more-parallelism/297>.

Rai, Dhiraj. *Logistic Regression in Spark ML*. 02 November 2018. online. 28 May 2020. <https://medium.com/@dhiraj.p.rai/logistic-regression-in-spark-ml-8a95b5f5434c>.

RAY, SUNIL. *6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R*. 11 September 2017. online. 22 May 2020. <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>.

Spark. *Classification and regression*. n.d. online. 22 May 2020. <https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier>.

Wan, Jun. *Oversampling and Undersampling with PySpark*. n.d. online. 22 May 2020. <https://medium.com/@junwan01/oversampling-and-undersampling-with-pyspark-5dbc25cdf253>.

wikipedia. *Cold start (recommender systems)*. n.d. online. 29 May 2020. <https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)>.

Willems, Karlijn. *Apache Spark Tutorial: ML with PySpark*. 29 July 2017. online. 25 May 2020. <https://www.datacamp.com/community/tutorials/apache-spark-tutorial-machine-learning#preprocess>.