N-R Team

E Navaneet Kumar, Rithvika Paladugu

- **The `ratings.csv` file is prepared for machine learning models and consists of merged data from two different sources.**
- **The first source is `output1.txt`, originating from Parts 1 of Homeworks 5 and 6, which includes user and track identifiers, along with album and artist scores. This file contains 120,000 records.**
- **The second source is `test2_new.txt`, which provides the actual recommendation outcomes (ground truth) for 6,000 user-track pairs that are also present in `output1.txt`.**

Now we will import all the necessary packages

In [ ]:

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz
!tar -xvf spark-3.5.1-bin-hadoop3.tgz
!pip install -q findspark
```

In [ ]:

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.1-bin-hadoop3"
```

In [ ]:

```
import findspark
findspark.init()
```

In [ ]:

```
from pyspark.sql import SparkSession    # main entry point for DataFrame and SQL function
ality
from pyspark.sql.functions import col    # for returning a column based on a given colum
n name
from pyspark.sql.functions import lit    # for adding a new column to PySpark DataFrame
from pyspark.ml.classification import LogisticRegression    # for classification model
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler    # for pre
paring data for classification
from pyspark.ml.evaluation import MulticlassClassificationEvaluator    # for evaluating c
lassification models
from pyspark.ml import Pipeline
import pandas as pd    # for data frames
import numpy as np     # for arrays
import time            # for timing cells
import matplotlib.pyplot as plt # plotting graphs
```

In [ ]:

```
spark = SparkSession.builder.appName('HW9_N-R Team').getOrCreate()
```

In [ ]:

```
spark
```

Out[ ]:

SparkSession - in-memory
SparkContext

[Spark UI](#)

Version

`v3.5.1`

Master

`local[*]`

AppName

`HW 9`

In [ ]:

```
ground_truth_columns = ['userID', 'trackID', 'ground_truth']
```

In [ ]:

```
ground_truth_df = pd.read_csv('test2_new.txt', sep='|', names=ground_truth_columns)
```

In [ ]:

```
ground_truth_df
```

Out[ ]:

	userID	trackID	ground_truth
0	200031	30877	1
1	200031	8244	1
2	200031	130183	0
3	200031	198762	0
4	200031	34503	1
...	...	...	...
5995	212234	137371	0
5996	212234	42375	0
5997	212234	277867	1
5998	212234	83093	1
5999	212234	239143	1

6000 rows × 3 columns

- **We started by specifying the column names for the dataset as 'userID', 'trackID', and 'ground_truth'. These columns correspond to the unique identifiers for users and tracks, and a truth value indicating if a track was liked.**
- **Next, we loaded the data from 'test2_new.txt' into a DataFrame. This text file is structured with each piece of data separated by a pipe ('|'), which then indicated to Pandas using the `sep='|'` parameter.**
- **We named the DataFrame `ground_truth_df` to reflect that it contains the ground truth data for the recommendation system.**
- **Finally, to ensure the data loaded correctly, We displayed the DataFrame which would show the top rows by default, giving me a quick snapshot of the data structure.**

In [ ]:

```
scores_columns = ['userID', 'trackID', 'album_score', 'artist_score']
```

In [ ]:

```
scores_df = pd.read_csv('output1.txt', sep='|', names=scores_columns)
```

In [ ]:

```
scores_df
```

```
scores_df
```

Out[ ]:

	userID	trackID	album_score	artist_score
0	199810	208019	0.0	0.0
1	199810	74139	0.0	0.0
2	199810	9903	0.0	0.0
3	199810	242681	0.0	0.0
4	199810	18515	0.0	70.0
...	...	...	...	...
119995	249010	72192	0.0	0.0
119996	249010	86104	0.0	0.0
119997	249010	186634	90.0	90.0
119998	249010	293818	0.0	0.0
119999	249010	262811	90.0	90.0

120000 rows × 4 columns

In [ ]:

```
ratings_df = ground_truth_df.merge(scores_df, on=['userID', 'trackID']).fillna(0)     # i
nner join by default
```

In [ ]:

```
ratings_df
```

Out[ ]:

	userID	trackID	ground_truth	album_score	artist_score
0	200031	30877	1	90.0	50.0
1	200031	8244	1	90.0	0.0
2	200031	130183	0	0.0	0.0
3	200031	198762	0	0.0	0.0
4	200031	34503	1	90.0	50.0
...	...	...	...	...	...
5995	212234	137371	0	0.0	0.0
5996	212234	42375	0	0.0	0.0
5997	212234	277867	1	90.0	90.0
5998	212234	83093	1	90.0	90.0
5999	212234	239143	1	90.0	90.0

6000 rows × 5 columns

- **Combined the ground truth data with the scores data into a single DataFrame called `ratings_df`. This was done by matching each user and track pair from `ground_truth_df` and `scores_df` on their 'userID' and 'trackID' columns.**
- **The `merge` function performs an inner join by default, which means only user-track pairs present in both DataFrames are included in the resulting `ratings_df`.**
- **After merging, I used the `fillna(0)` method to replace any missing values that might have appeared during the merge with zeros.**
- **Lastly, displayed `ratings_df`.This will give us a 6,000 line DF that contains the scores and ground truths.bold text**

Finally we write this to a csv file.

Next

We will prepare the `ratings.csv` for various machine learning classification models.

- **Initially, we converted the `ratings.csv` into a Spark DataFrame. This transformation is crucial as it allows for the utilization of Spark's powerful distributed data processing capabilities, which are particularly effective for handling machine learning tasks on large datasets.**

In [ ]:

```
ratings_df.to_csv('ratings.csv', index=None)
```

In [ ]:

```
ratings_df = spark.read.csv('ratings.csv', header=True, inferSchema=True)
```

In [ ]:

```
ratings_df
```

Out[ ]:

```
DataFrame[userID: int, trackID: int, ground_truth: int, album_score: double, artist_score
: double]
```

In [ ]:

```
ratings_df.count()
```

Out[ ]:

```
6000
```

In [ ]:

```
ratings_columns = ratings_df.columns
```

In [ ]:

```
pd.DataFrame(ratings_df.take(6000), columns=ratings_columns).groupby('ground_truth').cou
nt()
```

Out[ ]:

ground_truth	userID	trackID	album_score	artist_score
0	3000	3000	3000	3000
1	3000	3000	3000	3000

In [ ]:

```
ratings_df.printSchema()
```

```
root
 |-- userID: integer (nullable = true)
 |-- trackID: integer (nullable = true)
 |-- ground_truth: integer (nullable = true)
 |-- album_score: double (nullable = true)
 |-- artist_score: double (nullable = true)
```

After converting the `ratings.csv` file into a Spark DataFrame, we checked the schema of the DataFrame using the `printSchema()` method. This allowed us to confirm that the DataFrame was structured correctly, with the appropriate data types assigned to each column:

- `userID`: an integer column, representing the unique identifier for users.
- `trackID`: an integer column, representing the unique identifier for tracks.
- `ground_truth`: an integer column, indicating whether the track was liked by the user.
- `album_score`: a double column, representing the score of the album associated with the track.
- `artist_score`: a double column, representing the score of the artist associated with the track.

Each of these columns is set to allow null values (`nullable = true`), which is standard in data schemas to accommodate missing entries.

In [ ]:

```
ratings_df = ratings_df.withColumn('ground_truth', ratings_df['ground_truth'].cast('strin
g'))
```

Converted the `ground_truth` column in the `ratings_df` DataFrame from integers to strings. This step is essential because the `StringIndexer()` method, which I plan to use later, requires the input column to be in string format. This conversion ensures that the DataFrame meets the prerequisites for applying the `StringIndexer()`.

In [ ]:

```
ratings_df.dtypes
```

Out[ ]:

```
[('userID', 'int'),
 ('trackID', 'int'),
 ('ground_truth', 'string'),
 ('album_score', 'double'),
 ('artist_score', 'double')]
```

Then utilized the `VectorAssembler()` function to transform and merge multiple numeric columns into a single vector column. This is a crucial step for preparing the data for machine learning models, as it consolidates the features into a format that the algorithms can process effectively.

In [ ]:

```
feature_columns = ['album_score', 'artist_score']
stages = []
assembler_inputs = feature_columns
assembler = VectorAssembler(inputCols=assembler_inputs, outputCol='features')    # merge
s multiple columns into a vector column
stages += [assembler]
```

In [ ]:

```
label_column = 'ground_truth'
label_string_idx = StringIndexer(inputCol=label_column, outputCol='label')
stages += [label_string_idx]
```

In [ ]:

```
pipeline = Pipeline(stages=stages)                    # initialize the pipeline
pipeline_model = pipeline.fit(ratings_df)             # fit the pipeline model
train_df = pipeline_model.transform(ratings_df)   # transform the input DF with the pipeli
ne model
```

- **Initializing the Pipeline**: We created a `Pipeline` object and specified its `stages`, which includes all the transformations I planned (like `StringIndexer` and `VectorAssembler`). This organizes the steps in a sequence that will be executed in order.
- **Fitting the Pipeline**: Next, we fit the pipeline to the `ratings_df`. This step involves the pipeline learning

from the data, essentially training on the DataFrame to understand the transformations specified in its stages.

- **Transforming the Data**: After fitting, used the trained pipeline model to transform `ratings_df`. This applies all the transformations defined in the pipeline to the data, outputting a new DataFrame, `train_df`, which is now ready for machine learning models with all features properly encoded and assembled.

In [ ]:

```
selected_columns = ['label', 'features'] + ratings_columns
train_df = train_df.select(selected_columns)
train_df.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- userID: integer (nullable = true)
 |-- trackID: integer (nullable = true)
 |-- ground_truth: string (nullable = true)
 |-- album_score: double (nullable = true)
 |-- artist_score: double (nullable = true)
```

We've successfully added two new columns to our DataFrame:

- `label`: This column, of type double, stores the labels that our machine learning models will predict.
- `features`: A vector column that encapsulates all the features needed for modeling.

Next, we'll display the first five rows of this updated DataFrame.

In [ ]:

```
pd.DataFrame(train_df.take(5), columns=train_df.columns).transpose()
```

Out[ ]:

	0	1	2	3	4
label	1.0	1.0	0.0	0.0	1.0
features	[90.0, 50.0]	[90.0, 0.0]	(0.0, 0.0)	(0.0, 0.0)	[90.0, 50.0]
userID	200031	200031	200031	200031	200031
trackID	30877	8244	130183	198762	34503
ground_truth	1	1	0	0	1
album_score	90.0	90.0	0.0	0.0	90.0
artist_score	50.0	0.0	0.0	0.0	50.0

Now we split the data into training data and testing data with a 70:30 split.

In [ ]:

```
train_df, test_df = train_df.randomSplit([0.7, 0.3], seed=2018)
```

In [ ]:

```
print(f'Training Dataset Count: {train_df.count()}')
print(f'Test Dataset Count: {test_df.count()}')
```

```
Training Dataset Count: 4260
Test Dataset Count: 1740
```

Next, we'll load the `output1.txt` file, which contains 120,000 entries that we need to predict using our models. Similar to the earlier steps, w'll set up the pipeline to process this data, ensuring that each entry is formatted correctly with labels and features columns, ready for the prediction phase.

In [ ]:

```
prediction_df = spark.read.csv('output1.txt', sep='|', inferSchema=True)
```

In [ ]:

```
prediction_df.count()
```

Out[ ]:

```
120000
```

In [ ]:

```
prediction_df = prediction_df.withColumnRenamed("_c0", "userID").withColumnRenamed("_c1",
"trackID").withColumnRenamed("_c2", "albumScore").withColumnRenamed("_c3", "artistScore"
)
```

In [ ]:

```
prediction_columns = prediction_df.columns
prediction_columns
```

Out[ ]:

```
['userID', 'trackID', 'albumScore', 'artistScore']
```

In [ ]:

```
prediction_df = prediction_df.withColumn('prediction', lit('0'))
```

In [ ]:

```
pd.DataFrame(prediction_df.take(5), columns=prediction_df.columns).transpose()
```

Out[ ]:

	0	1	2	3	4
userID	199810	199810	199810	199810	199810
trackID	208019	74139	9903	242681	18515
albumScore	0.0	0.0	0.0	0.0	0.0
artistScore	0.0	0.0	0.0	0.0	70.0
prediction	0	0	0	0	0

In [ ]:

```
prediction_df.printSchema()
```

```
root
 |-- userID: integer (nullable = true)
 |-- trackID: integer (nullable = true)
 |-- albumScore: double (nullable = true)
 |-- artistScore: double (nullable = true)
 |-- prediction: string (nullable = false)
```

In [ ]:

```
feature_columns = ['albumScore', 'artistScore']
stages = []
assembler_inputs = feature_columns
assembler = VectorAssembler(inputCols=assembler_inputs, outputCol='features')     # merge
s multiple columns into a vector column
stages += [assembler]
```

In [ ]:

```
label_column = 'prediction'
label_string_idx = StringIndexer(inputCol=label_column, outputCol='label')
stages += [label_string_idx]
```

In [ ]:

```
prediction_pipeline = Pipeline(stages=stages)                              # initialize the pip
eline
prediction_pipeline_model = prediction_pipeline.fit(prediction_df)   # fit the pipeline mo
del
prediction_df = prediction_pipeline_model.transform(prediction_df)   # transform the input
DF with the pipeline model
```

In [ ]:

```
selected_columns = ['label', 'features'] + prediction_columns
prediction_df = prediction_df.select(selected_columns)
prediction_df.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- userID: integer (nullable = true)
 |-- trackID: integer (nullable = true)
 |-- albumScore: double (nullable = true)
 |-- artistScore: double (nullable = true)
```

In [ ]:

```
pd.DataFrame(prediction_df.take(5), columns=prediction_df.columns).transpose()
```

Out[ ]:

	0	1	2	3	4
label	0.0	0.0	0.0	0.0	0.0
features	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	[0.0, 70.0]
userID	199810	199810	199810	199810	199810
trackID	208019	74139	9903	242681	18515
albumScore	0.0	0.0	0.0	0.0	0.0
artistScore	0.0	0.0	0.0	0.0	70.0

Model 1 - Logistic Regression

In [ ]:

```
from pyspark.ml.classification import LogisticRegression
```

In [ ]:

```
start_time = time.time()

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=100)  # in
itialize a logistic regression model
lr_model = lr.fit(train_df)                                                          # f
it the training data with the model

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 4.75 seconds.
```

In [ ]:

```
lr_model.coefficients
```

Out[ ]:

```
DenseVector([0.0452, 0.0326])
```

In [ ]:

```
beta = np.sort(lr_model.coefficients)
```

In [ ]:

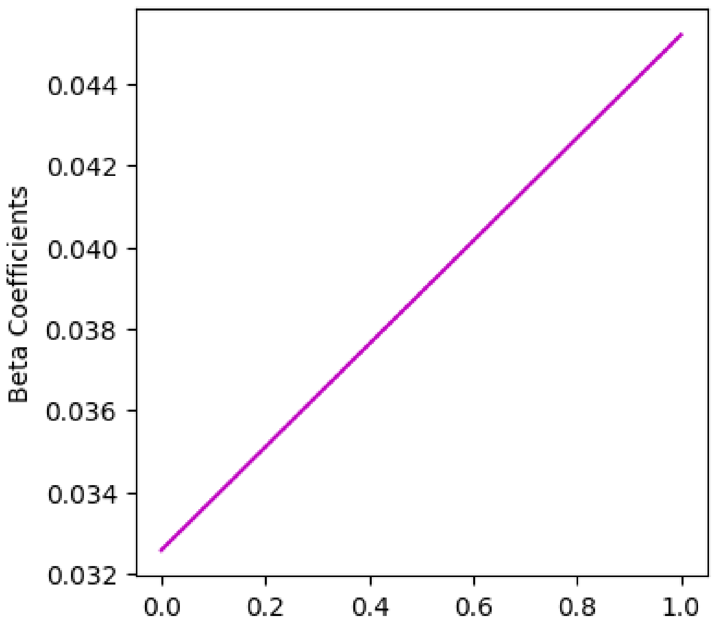
```
beta
```

Out[ ]:

```
array([0.03256793, 0.04519982])
```

In [ ]:

```
plt.figure(figsize=(4, 4))
plt.plot(beta, color="m")
plt.ylabel('Beta Coefficients')
plt.show()
```



In [ ]:

```
training_summary = lr_model.summary
```

In [ ]:

```
roc = training_summary.roc.toPandas()
```

In [ ]:

```
roc
```

Out[ ]:

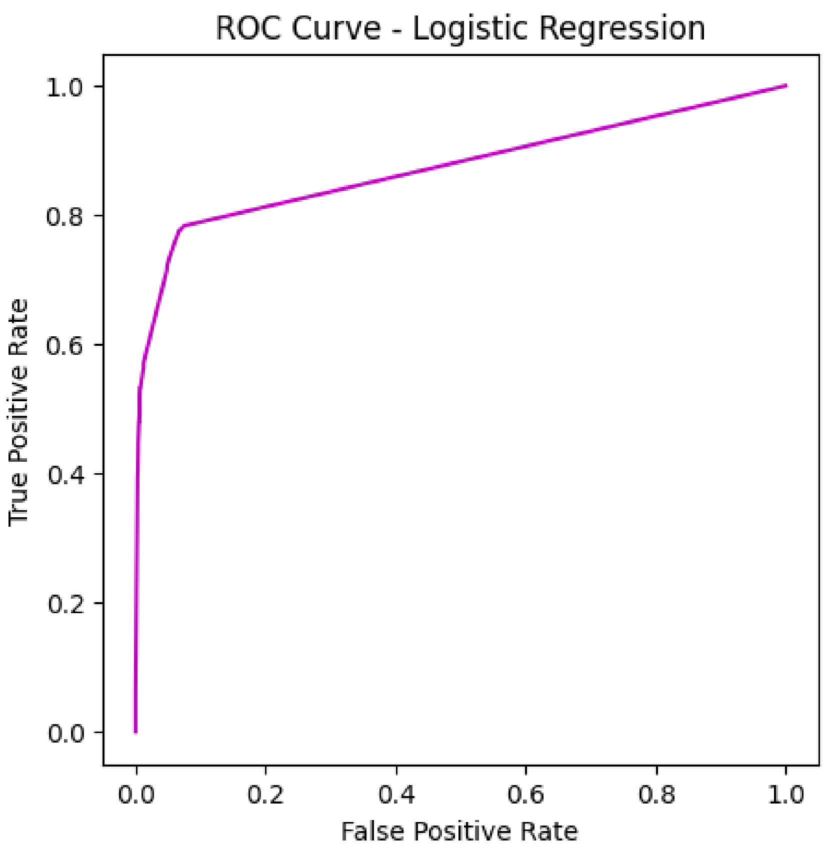
	FPR	TPR
0	0.000000	0.000000
1	0.000000	0.039683
2	0.000000	0.040149
3	0.000000	0.040616
4	0.000000	0.041083
...	...	...

	FPR	TPR
88	0.076015	0.783847
89	0.076487	0.783847
90	0.076959	0.783847
91	1.000000	1.000000
92	1.000000	1.000000

93 rows × 2 columns

In [ ]:

```
plt.figure(figsize=(5, 5))
plt.plot(roc.FPR, roc.TPR, color= "m")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.show()
print(f'Training Set AUC = {training_summary.areaUnderROC}')
```




```
Training Set AUC = 0.8737133978552075
```

We achieved pretty good results with a Training Set AUC of 0.8737133978552075, indicating strong predictive performance of our model.

In [ ]:

```
predictions = lr_model.transform(test_df)
predictions.select('userID', 'trackID', 'label', 'probability', 'rawPrediction', 'predict
ion').show(20)
```

```
+------+-------+-----+--------------------+--------------------+----------+
|userID|trackID|label|         probability|       rawPrediction|prediction|
+------+-------+-----+--------------------+--------------------+----------+
|200031| 130183|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200065| 179571|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200070| 124239|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200070| 271459|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200085| 134106|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200099|  41892|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200106| 152491|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
```

```
|200124| 284066|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200143| 131171|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200143| 187136|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200160| 231680|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200166| 193878|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200168| 226576|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200176| 141029|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200193| 129391|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200263| 132785|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200270| 139707|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200279| 109024|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200308|  55997|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200314| 214898|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
+------+-------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

In [ ]:

```
sort_predictions = predictions.select('userID', 'trackID', 'label', 'probability', 'rawP
rediction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_predictions.show()
```

```
+------+-------+-----+--------------------+--------------------+----------+
|userID|trackID|label|         probability|       rawPrediction|prediction|
+------+-------+-----+--------------------+--------------------+----------+
|200031| 130183|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200031|   8244|  1.0|[0.06488875729218...|[-2.6679911209111...|       1.0|
|200031|  30877|  1.0|[0.01343471003408...|[-4.2963878477543...|       1.0|
|200055|  56695|  1.0|[0.00368752576332...|[-5.5991052292289...|       1.0|
|200065| 179571|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200065| 119451|  1.0|[0.17782970128751...|[-1.5311211916575...|       1.0|
|200065|  26875|  1.0|[0.00368752576332...|[-5.5991052292289...|       1.0|
|200070| 124239|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200070| 271459|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200074| 289311|  1.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200085| 134106|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200085|  49158|  1.0|[0.00704934308743...|[-4.9477465384916...|       1.0|
|200099|  41892|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200099| 173182|  1.0|[0.06488875729218...|[-2.6679911209111...|       1.0|
|200106| 152491|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200106|  72517|  1.0|[0.00368752576332...|[-5.5991052292289...|       1.0|
|200124| 284066|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200124| 112595|  1.0|[0.17782970128751...|[-1.5311211916575...|       1.0|
|200143|   8486|  1.0|[0.80218276453714...|[1.39999291666030...|       0.0|
|200143| 187136|  0.0|[0.80218276453714...|[1.39999291666030...|       0.0|
+------+-------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

In [ ]:

```
logistic_predictions = lr_model.transform(prediction_df)    # transform prediction_df wit
h logistic regression model
logistic_predictions.select('userID', 'trackID', 'probability', 'rawPrediction', 'predict
ion').show(12)
```

```
+------+-------+--------------------+--------------------+----------+
|userID|trackID|         probability|       rawPrediction|prediction|
+------+-------+--------------------+--------------------+----------+
|199810| 208019|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810|  74139|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810|   9903|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810| 242681|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810|  18515|[0.29322699696820...|[-0.8797625009202...|       1.0|
|199810| 105760|[0.17782970128751...|[-1.5311211916575...|       1.0|
|199812| 276940|[0.80218276453714...|[1.39999291666030...|       0.0|
|199812| 142408|[0.00169769820591...|[-6.3767828009944...|       1.0|
|199812| 130023|[0.00169769820591...|[-6.3767828009944...|       1.0|
|199812|  29189|[0.80218276453714...|[1.39999291666030...|       0.0|
|199812| 223706|[0.13507641145138...|[-1.8568005370262...|       1.0|
|199812| 211361|[0.80218276453714...|[1.39999291666030...|       0.0|
```

```
+------+-------+--------------------+--------------------+----------+
only showing top 12 rows
```

In [ ]:

```
sort_logistic_predictions = logistic_predictions.select('userID', 'trackID', 'probability
', 'rawPrediction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_logistic_predictions.show(6)
```

```
+------+-------+--------------------+--------------------+----------+
|userID|trackID|         probability|       rawPrediction|prediction|
+------+-------+--------------------+--------------------+----------+
|199810| 208019|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810|  74139|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810|   9903|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810| 242681|[0.80218276453714...|[1.39999291666030...|       0.0|
|199810|  18515|[0.29322699696820...|[-0.8797625009202...|       1.0|
|199810| 105760|[0.17782970128751...|[-1.5311211916575...|       1.0|
+------+-------+--------------------+--------------------+----------+
only showing top 6 rows
```

In [ ]:

```
pd_sort_logistic_predictions = sort_logistic_predictions.toPandas().fillna(0.0)
```

In [ ]:

```
pd_sort_logistic_predictions
```

Out[ ]:

	userID	trackID	probability	rawPrediction	prediction
0	199810	208019	[0.8021827645371431, 0.1978172354628569]	[1.3999929166603091, -1.3999929166603091]	0.0
1	199810	74139	[0.8021827645371431, 0.1978172354628569]	[1.3999929166603091, -1.3999929166603091]	0.0
2	199810	9903	[0.8021827645371431, 0.1978172354628569]	[1.3999929166603091, -1.3999929166603091]	0.0
3	199810	242681	[0.8021827645371431, 0.1978172354628569]	[1.3999929166603091, -1.3999929166603091]	0.0
4	199810	18515	[0.2932269969682032, 0.7067730030317968]	[-0.8797625009202483, 0.8797625009202483]	1.0
...	...	...	...	...	...
119995	249010	86104	[0.8021827645371431, 0.1978172354628569]	[1.3999929166603091, -1.3999929166603091]	0.0
119996	249010	293818	[0.8021827645371431, 0.1978172354628569]	[1.3999929166603091, -1.3999929166603091]	0.0
119997	249010	110470	[0.003687525763326008, 0.996312474236674]	[-5.599105229228973, 5.599105229228973]	1.0
119998	249010	186634	[0.003687525763326008, 0.996312474236674]	[-5.599105229228973, 5.599105229228973]	1.0
119999	249010	262811	[0.003687525763326008, 0.996312474236674]	[-5.599105229228973, 5.599105229228973]	1.0

120000 rows × 5 columns

In [ ]:

```
columns_to_write = ['userID', 'trackID']
pd_sort_logistic_predictions.to_csv('lr_predictions.csv', index=False, header=None, colum
ns=columns_to_write)
```

In [ ]:

```
f_lr_predictions = open('lr_predictions.csv')
f_lr_final_predictions = open('lr_final_predictions.csv', 'w')
```

In [ ]:

```
f_lr_final_predictions.write('TrackID,Predictor\n')
```

Out[ ]:

```
18
```

In [ ]:

```
last_user_id = -1
track_id_out_vec = [0] * 6
```

In [ ]:

```
start_time = time.time()

# Go through each line of the predictions file
for line in f_lr_predictions:
    arr_out = line.strip().split(',')     # remove any spaces/new lines and create list
    user_id_out = arr_out[0]              # set user
    track_id_out = arr_out[1]             # set track

    if user_id_out != last_user_id:              # if new user reached
        i = 0                                    # reset i

    track_id_out_vec[i] = track_id_out           # add trackID to trackID array

    i = i + 1                      # increment i
    last_user_id = user_id_out     # set last_user_id as current userID

    if i == 6:                                # if last entry for current user reached
        # Here we set the predictions
        predictions = np.ones(shape=(6)) # initialize numpy array for predictions
        for index in range(0, 3):
            predictions[index] = 0           # set first 3 values in array to 0 (other 3
are 1)

        # Here we write to the final predictions file for the 6 track predictions for the
current user
        for ii in range(0, 6):
            out_str = str(user_id_out) + '_' + str(track_id_out_vec[ii]) + ',' + str(int
(predictions[ii]))
            f_lr_final_predictions.write(out_str + '\n')


end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 0.27 seconds.
```

In [ ]:

```
f_lr_predictions.close()
f_lr_final_predictions.close()
```

Model 2 - Gradient-Boosted Tree Classifier

In [ ]:

```
from pyspark.ml.classification import GBTClassifier
```

In [ ]:

```
start_time = time.time()
```

```
gbt = GBTClassifier(maxIter=100)
gbt_model = gbt.fit(train_df)

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 37.10 seconds.
```

In [ ]:

```
predictions_gbt = gbt_model.transform(test_df)
```

In [ ]:

```
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction
', metricName='accuracy')     # initialize an Evaluator for Multiclass Classification
accuracy = evaluator.evaluate(predictions_gbt)     # evaluate random forest model on predi
ctions
print(f'Test Error = {1.0 - accuracy:.2%}')
```

```
Test Error = 14.48%
```

In [ ]:

```
sort_predictions_gbt = predictions_gbt.select('userID', 'trackID', 'label', 'probability'
, 'rawPrediction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_predictions_gbt.show(5)
```

```
+------+-------+-----+--------------------+--------------------+----------+
|userID|trackID|label|         probability|       rawPrediction|prediction|
+------+-------+-----+--------------------+--------------------+----------+
|200031| 130183|  0.0|[0.80853359039673...|[0.72025491912707...|       0.0|
|200031|  30877|  1.0|[0.01165681655269...|[-2.2200694341913...|       1.0|
|200031|   8244|  1.0|[0.01165681655269...|[-2.2200694341913...|       1.0|
|200055|  56695|  1.0|[0.01632026531261...|[-2.0494463824457...|       1.0|
|200065| 179571|  0.0|[0.80853359039673...|[0.72025491912707...|       0.0|
+------+-------+-----+--------------------+--------------------+----------+
only showing top 5 rows
```

In [ ]:

```
gbt_predictions = gbt_model.transform(prediction_df)     # transform prediction_df with gr
adient-boosted tree model
gbt_predictions.select('userID', 'trackID', 'probability', 'rawPrediction', 'prediction')
.show(10)
```

```
+------+-------+--------------------+--------------------+----------+
|userID|trackID|         probability|       rawPrediction|prediction|
+------+-------+--------------------+--------------------+----------+
|199810| 208019|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810|  74139|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810|   9903|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810| 242681|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810|  18515|[0.24633418901301...|[-0.5591299711568...|       1.0|
|199810| 105760|[0.19777096337535...|[-0.7001422692047...|       1.0|
|199812| 276940|[0.80853359039673...|[0.72025491912707...|       0.0|
|199812| 142408|[0.01161609118736...|[-2.2218399428711...|       1.0|
|199812| 130023|[0.01161609118736...|[-2.2218399428711...|       1.0|
|199812|  29189|[0.80853359039673...|[0.72025491912707...|       0.0|
+------+-------+--------------------+--------------------+----------+
only showing top 10 rows
```

In [ ]:

```
sort_gbt_predictions = gbt_predictions.select('userID', 'trackID', 'probability', 'rawPre
diction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_gbt_predictions.show(5)
```

```
+------+-------+--------------------+--------------------+----------+
|userID|trackID|         probability|       rawPrediction|prediction|
+------+-------+--------------------+--------------------+----------+
|199810| 208019|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810|  74139|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810|   9903|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810| 242681|[0.80853359039673...|[0.72025491912707...|       0.0|
|199810|  18515|[0.24633418901301...|[-0.5591299711568...|       1.0|
+------+-------+--------------------+--------------------+----------+
only showing top 5 rows
```

In [ ]:

```python
pd_sort_gbt_predictions = sort_gbt_predictions.toPandas().fillna(0.0)
```

In [ ]:

```python
pd_sort_gbt_predictions
```

Out[ ]:

	userID	trackID	probability	rawPrediction	prediction
0	199810	208019	[0.8085335903967381, 0.1914664096032619]	[0.720254919127077 5, -0.7202549191270775]	0.0
1	199810	74139	[0.8085335903967381, 0.1914664096032619]	[0.7202549191270775, -0.7202549191270775]	0.0
2	199810	9903	[0.8085335903967381, 0.1914664096032619]	[0.7202549191270775, -0.7202549191270775]	0.0
3	199810	242681	[0.8085335903967381, 0.1914664096032619]	[0.7202549191270775, -0.7202549191270775]	0.0
4	199810	18515	[0.24633418901301743, 0.7536658109869826]	[-0.5591299711568518, 0.5591299711568518]	1.0
...	...	...	...	...	...
119995	249010	86104	[0.8085335903967381, 0.1914664096032619]	[0.7202549191270775, -0.7202549191270775]	0.0
119996	249010	293818	[0.8085335903967381, 0.1914664096032619]	[0.7202549191270775, -0.7202549191270775]	0.0
119997	249010	110470	[0.0163202653126165, 0.9836797346873835]	[-2.0494463824457227, 2.0494463824457227]	1.0
119998	249010	186634	[0.0163202653126165, 0.9836797346873835]	[-2.0494463824457227, 2.0494463824457227]	1.0
119999	249010	262811	[0.0163202653126165, 0.9836797346873835]	[-2.0494463824457227, 2.0494463824457227]	1.0

120000 rows × 5 columns

In [ ]:

```python
columns_to_write = ['userID', 'trackID']
pd_sort_gbt_predictions.to_csv('gbt_predictions.csv', index=False, header=None, columns=
columns_to_write)
```

In [ ]:

```python
f_gbt_predictions = open('gbt_predictions.csv')
f_gbt_final_predictions = open('gbt_final_predictions.csv', 'w')
```

In [ ]:

```python
f_gbt_final_predictions.write('TrackID,Predictor\n')
```

Out[ ]:

```
18
```

In [ ]:

```
# Initialize some values
last_user_id = -1
track_id_out_vec = [0] * 6
```

In [ ]:

```
start_time = time.time()

# Go through each line of the predictions file
for line in f_gbt_predictions:
    arr_out = line.strip().split(',')     # remove any spaces/new lines and create list
    user_id_out = arr_out[0]              # set user
    track_id_out = arr_out[1]             # set track

    if user_id_out != last_user_id:              # if new user reached
        i = 0                                    # reset i

    track_id_out_vec[i] = track_id_out           # add trackID to trackID array

    i = i + 1                    # increment i
    last_user_id = user_id_out   # set last_user_id as current userID

    if i == 6:                                # if last entry for current user reached
        # Here we set the predictions
        predictions = np.ones(shape=(6)) # initialize numpy array for predictions
        for index in range(0, 3):
            predictions[index] = 0            # set first 3 values in array to 0 (other 3
are 1)

        # Here we write to the final predictions file for the 6 track predictions for the
current user
        for ii in range(0, 6):
            out_str = str(user_id_out) + '_' + str(track_id_out_vec[ii]) + ',' + str(int
(predictions[ii]))
            f_gbt_final_predictions.write(out_str + '\n')


end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

Done! Time elapsed - 0.29 seconds.

In [ ]:

```
f_gbt_predictions.close()
f_gbt_final_predictions.close()
```

Model 3 - Decision Tree Classifier

In [ ]:

```
from pyspark.ml.classification import DecisionTreeClassifier
```

In [ ]:

```
start_time = time.time()

dt = DecisionTreeClassifier(featuresCol='features', labelCol='label', maxDepth=3)
dt_model = dt.fit(train_df)

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

Done! Time elapsed - 0.74 seconds.

In [ ]:

```
predictions_dt = dt_model.transform(test_df)
```

In [ ]:

```
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction
', metricName='accuracy')     # initialize an Evaluator for Multiclass Classification
accuracy = evaluator.evaluate(predictions_dt)     # evaluate decision tree model on predic
tions
print(f'Test Error = {1.0 - accuracy:.2%}')
```

```
Test Error = 14.48%
```

In [ ]:

```
sort_predictions_dt = predictions_dt.select('userID', 'trackID', 'label', 'probability',
'rawPrediction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_predictions_dt.show(5)
```

```
+------+-------+-----+--------------------+--------------+----------+
|userID|trackID|label|         probability| rawPrediction|prediction|
+------+-------+-----+--------------------+--------------+----------+
|200031| 130183|  0.0|[0.80858085808580...|[1960.0,464.0]|       0.0|
|200031|  30877|  1.0|[0.08880090497737...|[157.0,1611.0]|       1.0|
|200031|   8244|  1.0|[0.01470588235294...|    [1.0,67.0]|       1.0|
|200055|  56695|  1.0|[0.08880090497737...|[157.0,1611.0]|       1.0|
|200065| 179571|  0.0|[0.80858085808580...|[1960.0,464.0]|       0.0|
+------+-------+-----+--------------------+--------------+----------+
only showing top 5 rows
```

In [ ]:

```
dt_predictions = dt_model.transform(prediction_df)     # transform prediction_df with deci
sion tree model
dt_predictions.select('userID', 'trackID', 'probability', 'rawPrediction', 'prediction').
show(10)
```

```
+------+-------+--------------------+--------------+----------+
|userID|trackID|         probability| rawPrediction|prediction|
+------+-------+--------------------+--------------+----------+
|199810| 208019|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  74139|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|   9903|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810| 242681|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  18515|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199810| 105760|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812| 276940|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199812| 142408|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812| 130023|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812|  29189|[0.80858085808580...|[1960.0,464.0]|       0.0|
+------+-------+--------------------+--------------+----------+
only showing top 10 rows
```

In [ ]:

```
sort_dt_predictions = dt_predictions.select('userID', 'trackID', 'probability', 'rawPredi
ction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_dt_predictions.show(5)
```

```
+------+-------+--------------------+--------------+----------+
|userID|trackID|         probability| rawPrediction|prediction|
+------+-------+--------------------+--------------+----------+
|199810| 208019|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  74139|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|   9903|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810| 242681|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  18515|[0.08880090497737...|[157.0,1611.0]|       1.0|
+------+-------+--------------------+--------------+----------+
only showing top 5 rows
```

In [ ]:

```
pd_sort_dt_predictions = sort_dt_predictions.toPandas().fillna(0.0)
```

In [ ]:

```
pd_sort_dt_predictions
```

Out[ ]:

	userID	trackID	probability	rawPrediction	prediction
0	199810	208019	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
1	199810	74139	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
2	199810	9903	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
3	199810	242681	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
4	199810	18515	[0.08880090497737557, 0.9111990950226244]	[157.0, 1611.0]	1.0
...	...	...	...	...	...
119995	249010	86104	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
119996	249010	293818	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
119997	249010	110470	[0.08880090497737557, 0.9111990950226244]	[157.0, 1611.0]	1.0
119998	249010	186634	[0.08880090497737557, 0.9111990950226244]	[157.0, 1611.0]	1.0
119999	249010	262811	[0.08880090497737557, 0.9111990950226244]	[157.0, 1611.0]	1.0

120000 rows × 5 columns

In [ ]:

```
columns_to_write = ['userID', 'trackID']
pd_sort_dt_predictions.to_csv('dt_predictions.csv', index=False, header=None, columns=co
lumns_to_write)
```

In [ ]:

```
f_dt_predictions = open('dt_predictions.csv')
f_dt_final_predictions = open('dt_final_predictions.csv', 'w')
```

In [ ]:

```
f_dt_final_predictions.write('TrackID,Predictor\n')
```

Out[ ]:

18

In [ ]:

```
# Initialize some values
last_user_id = -1
track_id_out_vec = [0] * 6
```

In [ ]:

```
start_time = time.time()

# Go through each line of the predictions file
for line in f_dt_predictions:
    arr_out = line.strip().split(',')     # remove any spaces/new lines and create list
    user_id_out = arr_out[0]              # set user
    track_id_out = arr_out[1]             # set track

    if user_id_out != last_user_id:              # if new user reached
        i = 0                                    # reset i

    track_id_out_vec[i] = track_id_out           # add trackID to trackID array
```

```
    i = i + 1                           # increment i
    last_user_id = user_id_out    # set last_user_id as current userID

    if i == 6:                                  # if last entry for current user reached
        # Here we set the predictions
        predictions = np.ones(shape=(6)) # initialize numpy array for predictions
        for index in range(0, 3):
            predictions[index] = 0            # set first 3 values in array to 0 (other 3 
are 1)

        # Here we write to the final predictions file for the 6 track predictions for the 
current user
        for ii in range(0, 6):
            out_str = str(user_id_out) + '_' + str(track_id_out_vec[ii]) + ',' + str(int
(predictions[ii]))
            f_dt_final_predictions.write(out_str + '\n')


end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 0.27 seconds.
```

In [ ]:

```
f_dt_predictions.close()
f_dt_final_predictions.close()
```

Model 4 - Random Forest Classifier

In [ ]:

```
from pyspark.ml.classification import RandomForestClassifier
```

In [ ]:

```
start_time = time.time()

rf = RandomForestClassifier(featuresCol='features', labelCol='label')
rf_model = rf.fit(train_df)

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 2.20 seconds.
```

In [ ]:

```
predictions_rf = rf_model.transform(test_df)
```

In [ ]:

```
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction
', metricName='accuracy')     # initialize an Evaluator for Multiclass Classification
accuracy = evaluator.evaluate(predictions_rf)     # evaluate random forest model on predic
tions
print(f'Test Error = {1.0 - accuracy:.2%}')
```

```
Test Error = 14.48%
```

In [ ]:

```
sort_predictions_rf = predictions_rf.select('userID', 'trackID', 'label', 'probability', 
'rawPrediction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_predictions_rf.show(6)
```

```
+------+-------+-----+--------------------+--------------------+----------+
```

```
|userID|trackID|label|         probability|       rawPrediction|prediction|
+------+-------+-----+--------------------+--------------------+----------+
|200031| 130183|  0.0|[0.80809388591375...|[16.1618777182750...|       0.0|
|200031|  30877|  1.0|[0.02845040162389...|[0.56900803247793...|       1.0|
|200031|   8244|  1.0|[0.00847380746267...|[0.16947614925344...|       1.0|
|200055|  56695|  1.0|[0.01412590057019...|[0.28251801140389...|       1.0|
|200065| 179571|  0.0|[0.80809388591375...|[16.1618777182750...|       0.0|
|200065| 119451|  1.0|[0.19067347143701...|[3.81346942874023...|       1.0|
+------+-------+-----+--------------------+--------------------+----------+
only showing top 6 rows
```

In [ ]:

```
rf_predictions = dt_model.transform(prediction_df)     # transform prediction_df with rand
om forest model
rf_predictions.select('userID', 'trackID', 'probability', 'rawPrediction', 'prediction').
show(12)
```

```
+------+-------+--------------------+-------------+----------+
|userID|trackID|         probability| rawPrediction|prediction|
+------+-------+--------------------+-------------+----------+
|199810| 208019|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  74139|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|   9903|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810| 242681|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  18515|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199810| 105760|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812| 276940|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199812| 142408|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812| 130023|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812|  29189|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199812| 223706|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199812| 211361|[0.80858085808580...|[1960.0,464.0]|       0.0|
+------+-------+--------------------+-------------+----------+
only showing top 12 rows
```

In [ ]:

```
sort_rf_predictions = rf_predictions.select('userID', 'trackID', 'probability', 'rawPredi
ction', 'prediction').sort(col('userID').asc(), col('probability').desc())
sort_rf_predictions.show(6)
```

```
+------+-------+--------------------+-------------+----------+
|userID|trackID|         probability| rawPrediction|prediction|
+------+-------+--------------------+-------------+----------+
|199810| 208019|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  74139|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|   9903|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810| 242681|[0.80858085808580...|[1960.0,464.0]|       0.0|
|199810|  18515|[0.08880090497737...|[157.0,1611.0]|       1.0|
|199810| 105760|[0.08880090497737...|[157.0,1611.0]|       1.0|
+------+-------+--------------------+-------------+----------+
only showing top 6 rows
```

In [ ]:

```
pd_sort_rf_predictions = sort_rf_predictions.toPandas().fillna(0.0)
```

In [ ]:

```
pd_sort_rf_predictions
```

Out[ ]:

	userID	trackID	probability	rawPrediction	prediction
0	199810	208019	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
1	199810	74139	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0

	userID	trackID	probability	rawPrediction	prediction
2	199810	9903	[0.8085808580858086, 0.1914191419141943]	[1960.0, 464.0]	0.0
3	199810	242681	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
4	199810	18515	[0.0888009049773757, 0.9111990950226244]	[157.0, 1611.0]	1.0
...	...	...	...	...	...
119995	249010	86104	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
119996	249010	293818	[0.8085808580858086, 0.19141914191419143]	[1960.0, 464.0]	0.0
119997	249010	110470	[0.0888009049773757, 0.9111990950226244]	[157.0, 1611.0]	1.0
119998	249010	186634	[0.0888009049773757, 0.9111990950226244]	[157.0, 1611.0]	1.0
119999	249010	262811	[0.0888009049773757, 0.9111990950226244]	[157.0, 1611.0]	1.0

120000 rows × 5 columns

In [ ]:

```
columns_to_write = ['userID', 'trackID']
pd_sort_rf_predictions.to_csv('rf_predictions.csv', index=False, header=None, columns=co
lumns_to_write)
```

In [ ]:

```
f_rf_predictions = open('rf_predictions.csv')
f_rf_final_predictions = open('rf_final_predictions.csv', 'w')
```

In [ ]:

```
f_rf_final_predictions.write('TrackID,Predictor\n')
```

Out[ ]:

```
18
```

In [ ]:

```
# Initialize some values
last_user_id = -1
track_id_out_vec = [0] * 6
```

In [ ]:

```
start_time = time.time()

# Go through each line of the predictions file
for line in f_rf_predictions:
    arr_out = line.strip().split(',')      # remove any spaces/new lines and create list
    user_id_out = arr_out[0]               # set user
    track_id_out = arr_out[1]              # set track

    if user_id_out != last_user_id:               # if new user reached
        i = 0                                     # reset i

    track_id_out_vec[i] = track_id_out            # add trackID to trackID array

    i = i + 1                     # increment i
    last_user_id = user_id_out    # set last_user_id as current userID

    if i == 6:                                # if last entry for current user reached
        # Here we set the predictions
        predictions = np.ones(shape=(6)) # initialize numpy array for predictions
        for index in range(0, 3):
            predictions[index] = 0            # set first 3 values in array to 0 (other 3
are 1)

        # Here we write to the final predictions file for the 6 track predictions for the
current user
        for ii in range(0, 6):
```

```
            out_str = str(user_id_out) + '_' + str(track_id_out_vec[ii]) + ',' + str(int
(predictions[ii]))
            f_rf_final_predictions.write(out_str + '\n')


end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 0.27 seconds.
```

In [ ]:

```
f_dt_predictions.close()
f_dt_final_predictions.close()
```

Summary

Based on the Kaggle results we submitted, here's a breakdown of how each of the four machine learning classification models we implemented performed:

- **Logistic Regression: This model provided the highest score among the classifiers, with an accuracy of 0.845. This suggests that the logistic regression model was quite effective at capturing the linear relationships between the features and the target variable.**
- **Decision Tree: The decision tree model achieved an accuracy of 0.823. While it performed decently, it was slightly less effective compared to logistic regression. Decision trees are typically good at handling complex datasets with non-linear relationships, but they might overfit if not properly tuned.**
- **Random Forest: This model also scored 0.823, tying with the decision tree. Random forests, an ensemble method that uses multiple decision trees, usually provide better generalization compared to a single decision tree. The identical score to the decision tree suggests that in this particular case, the ensemble effect did not significantly enhance the predictive accuracy, which could be due to the nature of the data or model parameters.**
- **Gradient-Boosted Tree: Nearly matching logistic regression, this model scored 0.844. Gradient boosting is another ensemble technique that builds trees sequentially, each new tree correcting errors made by previously trained trees. Its performance here indicates a strong capability, nearly matching the simpler logistic regression, which emphasizes its efficiency in handling different types of data distributions and complexities.**