Saavn Analytics Case Study

NOTES:

1. Please read readme file provided.

2. I have used below reference links in order to convert array of features obtained from ALS into a vector so can be used as input to clustering algorithm. (First link followed by second link)

   https://stackoverflow.com/questions/53165864/data-type-arraytypefloattype-false-not-supported-when-passed-to-vectorassemble

   https://stackoverflow.com/questions/52927303/convert-array-to-densevector-in-spark-dataframe-using-java/52943953

3. I have used this link to understand ALS and how to use it:

   https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html

4. I was getting below warnings during execution of ALS algorithm:

   WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
   WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS

   I tried a lot to resolve these warnings. In order to resolve these, I kept on following different links one after another:

   a. Used below to add dependency in pom.xml

      https://github.com/fommil/netlib-java

      ```
      <dependency>
        <groupId>com.github.fommil.netlib</groupId>
        <artifactId>all</artifactId>
        <version>1.1.2</version>
        <type>pom</type>
      </dependency>
      ```

   b. Used below link to install libgfortran (sudo yum install libgfortran.x86_64)

      https://github.com/jblas-project/jblas/wiki/Missing-Libraries

   c. Used below link to install openblas library (sudo yum install openblas-devel)

      https://gist.github.com/bmmalone/1b5f9ff72754c7d4b313c0b044c42684

   d. Used below link to install other libraries (sudo yum install atlas atlas-devel lapack lapack-devel blas blas-devel)

https://www.linuxquestions.org/questions/linux-software-2/instruction-for-installing-lapack-and-lapack-devel-on-centos-6-5-a-4175509481/

e. Used below to add more dependency in pom.xml:

https://oweissbarth.de/running-sparkmlib-with-native-lapack-and-blas/

```
<dependency>
 <groupId>net.sourceforge.f2j</groupId>
 <artifactId>arpack_combined_all</artifactId>
 <version>0.1</version>
</dependency>
```

f. Even after doing all above, I could not get rid of those warnings. Only option left was to rebuild spark with netlib as mentioned on this link:

https://stackoverflow.com/questions/46064099/failed-to-load-implementation-nativesystemblas-hibench

But this one was for windows and didn't work on Linux.

g. This is where I stopped and decided to ignore those warnings. It is mentioned by TA as well in this discussion forum link that we can ignore these warnings:

https://learn.upgrad.com/v/course/119/question/122505

5. My Cloudera VM has 26 GB RAM which is hosted on a 32 GB machine. I changed below parameters in yarn configuration for Cloudera VM during course of execution of this project:

   Set ApplicationMaster Java Maximum heap size to 10 GB
   Set Client Java heap size in Bytes to 12 GB
   Set Java heap size of NodeManager in Bytes to 12 GB
   Set Java heap size of Resource Manager in Bytes to 12 GB
   Set Container Memory (yarn.nodemanager.resource.memory-mb) to 24 GB
   Set Container Memory Maximum (yarn.scheduler.maximum-allocation-mb) to 24 GB
   Set Container Memory Minimum (yarn.scheduler.minimum-allocation-mb) to 1 GB
   Set Container Virtual CPU Cores (yarn.nodemanager.resource.cpu-vcores) to 4
   Set Container Virtual CPU Cores Maximum (yarn.scheduler.maximum-allocation-vcores) to 3

6. I used Cloudera VM in order to save cost associated with use of EC2 instance. **From Cloudera VM, it takes too much time in reading files from S3 so I had to download files and had copied onto Hadoop file system in Cloudera VM**.

7. In order to maximize CTR, I had to increase number of clusters due to which K-Means algorithm started taking bit more time but I still managed whole program within reasonable amount of time for such massive data and getting good overlap. In-fact saving data to files was also quite time consuming. If I would have just saved data at the end only once then my program would have got over in just 30-35 minutes. That's how I had tested and tuned my algorithm. At the end, I added steps to save more data in files at different intervals which added more time in program execution.

8. Approach to the problem and code screenshots followed by results having NotificationID and CTR for notifications pushed by the program, are shown in next section in this document.

Saavn Analytics Approach

We have been provided with 4 datasets:

a. Click-Stream Activity Data
b. Meta Data
c. Notification Clicks Data
d. Notification Artists Data

1. Program first checks if all 6 input parameters are passed or not:
    i. Checkpoint directory path on hadoop hdfs: (/user/cloudera/checkpoint_dir)
    ii. Input path to activity data: (s3a://bigdataanalyticsupgrad/activity/sample100mb.csv)
    iii. Input path to new metadata: (s3a://bigdataanalyticsupgrad/newmetadata/*)
    iv. Input path to notification clicks: (s3a://bigdataanalyticsupgrad/notification_clicks/*)
    v. Input path to notification artists: (s3a://bigdataanalyticsupgrad/notification_actor/*)
    vi. Output directory path on hadoop hdfs: (/user/cloudera/saavnanalytics)

2. If all required input parameters are passed to program, then program moves ahead. Logging is to set to ERROR. Spark configuration is set. Spark Context is created. Check point directory is set. Spark session is created, followed by creation of SQL context. Program then prints start time for execution of the program.

```
public static void main(String[] args) {

    /*
     * Check for input paramters. There should be 6 input parameters to the jar file:
     * 1. Checkpoint directory path on hadoop hdfs : (/user/ec2-user/checkpoint_dir)
     * 2. Input path to activity data : (s3a://bigdataanalyticsupgrad/activity/sample100mb.csv)
     * 3. Input path to new metadata : (s3a://bigdataanalyticsupgrad/newmetadata/*)
     * 4. Input path to notification clicks : (s3a://bigdataanalyticsupgrad/notification_clicks/*)
     * 5. Input path to notification artists : (s3a://bigdataanalyticsupgrad/notification_actor/*)
     * 6. Output directory path on hadoop hdfs : (/user/ec2-user/saavnanalytics)
     */
    if (args.length < 6) {
        System.out.println("Please specify all 6 parameters");
        System.out.println(
                "Provide checkpoint dir path, input path to activity data, metadata, notification clicks data "
                + "and notification artists data, output directory path");
        return;
    } else {

        // Setup logging to error only
        Logger.getLogger("org").setLevel(Level.ERROR);
        Logger.getLogger("akka").setLevel(Level.ERROR);

        // Setup SparkConf object
        SparkConf conf = new SparkConf().setAppName("SaavnAnalyticsCaseStudy");

        // Setup SparkContext
        SparkContext context = new SparkContext(conf);

        // Setup checkpoint directory
        context.setCheckpointDir(args[0]);

        // Create the spark session
        SparkSession spark = SparkSession.builder().appName("SaavnAnalyticsCaseStudy").getOrCreate();

        // Setup SQL context
        SQLContext sc = spark.sqlContext();

        // Print Start time
        System.out
                .println("\nStart Time : " + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()) + "\n");
```

3. Program then loads all input datasets one by one in different data frames and drop records where value is not present in the columns that we are interested in. And creates temporary views on those data frames so can be used in Spark SQL.

```
// Load User Activity Data into dataframe
System.out.println("\nLoading user activity data....\n");
Dataset<Row> userActivityData = spark.read().option("header", "false").csv(args[1])
        .select(col("_c0").as("UserID"), col("_c2").as("SongID")).na().drop();

// Create temporary view on userActivityData
userActivityData.createOrReplaceTempView("useractivitydata");

// Load New Meta Data into dataframe
System.out.println("\nLoading new metadata....\n");
Dataset<Row> newMetaData = spark.read().option("header", "false").csv(args[2])
        .select(col("_c0").as("SongID"), col("_c1").as("ArtistID")).na().drop();

// Create temporary view on newMetaData
newMetaData.createOrReplaceTempView("newmetadata");

// Load Notification Clicks Data into dataframe
System.out.println("\nLoading notification clicks data....\n");
Dataset<Row> notificationClickData = spark.read().option("header", "false").csv(args[3])
        .select(col("_c0").as("NotificationID"), col("_c1").as("UserID")).na().drop();

// Create temporary view on notificationClickData
notificationClickData.createOrReplaceTempView("notificationclickdata");

// Load Notification Artists Data into dataframe
System.out.println("\nLoading notification artists data....\n");
Dataset<Row> notificationArtistData = spark.read().option("header", "false").csv(args[4])
        .select(col("_c0").as("NotificationID"), col("_c1").as("ArtistID")).na().drop();

// Create temporary view on notificationArtistData
notificationArtistData.createOrReplaceTempView("notificationartistdata");
```

4. Program then calculates frequency of a song listened by a user. Select UserID, SongID and Count as frequency from useractivitydata group by UserID and SongID. Since UserID and SongID are in alphanumeric format, these will be converted into numeric format using StringIndexer before passing to ALS algorithm.

```
// Calculate frequency of a song listened by a user
Dataset<Row> activityDataFrequency = sc
        .sql("select UserID, SongID, count(*) as Frequency from useractivitydata group by UserID, SongID");

// This will convert the String values of UserID to numeric
StringIndexer userIndexer = new StringIndexer().setInputCol("UserID").setOutputCol("userIndex");

System.out.println("Changing UserID to numeric...\n");
Dataset<Row> userIndexed = userIndexer.fit(activityDataFrequency).transform(activityDataFrequency);

// This will convert the String values of SongID to numeric
StringIndexer songIndexer = new StringIndexer().setInputCol("SongID").setOutputCol("songIndex");

System.out.println("Changing SongID to numeric...\n");
Dataset<Row> indexedFinal = songIndexer.fit(userIndexed).transform(userIndexed);
```

5. I have used ALS algorithm to determine features from implicit learning. Parameters used for ALS algorithm, are shown in screenshot. We first setup ALS object and then input dataset is fit to get ALSModel. Then features are obtained from it using userFactors() method.

```
// Use ALS algorithm for implicit learning and obtain features from it
System.out.println("Starting the ALS algorithm to get features from implicit learning....\n");

ALS als = new ALS().setRank(10).setMaxIter(10).setImplicitPrefs(true).setUserCol("userIndex")
        .setItemCol("songIndex").setRatingCol("Frequency").setSeed(46L);

ALSModel model = als.fit(indexedFinal);

Dataset<Row> userImplicitFactors = model.userFactors();
```

6. Features obtained from ALSModel are inform of array. We need to convert this array of features into a Vector so can be passed to clustering algorithm. I have created a UDF to convert array of features into a Vector and registered it. Then called it on the output from ALSModel and obtained features inform of a Vector.

```
/*
 * Received the implicit factors from ALS. However, it is in array format. This
 * needs to be changed into vector. Use toVector UDF for the same.
 */
System.out.println(
        "\nRecieved implicit factors from ALS. However, it is in array format. "
        + "Working to change it into Vector now...\n");

spark.udf().register("toVector", convertToVector, new VectorUDT());

Dataset<Row> userFactorsAsVectors = userImplicitFactors.withColumn("features",
        functions.callUDF("toVector", userImplicitFactors.col("features")));
```

```
/*
 * UDF to convert array of features obtained from ALS into a Vector
 */
public static UDF1<Seq<Float>, Vector> convertToVector = new UDF1<Seq<Float>, Vector>() {

    private static final long serialVersionUID = 1L;

    public Vector call(Seq<Float> t1) throws Exception {

        List<Float> L = scala.collection.JavaConversions.seqAsJavaList(t1);
        double[] doubleArray = new double[t1.length()];
        for (int i = 0; i < L.size(); i++) {
            doubleArray[i] = L.get(i);
        }
        return Vectors.dense(doubleArray);
    }
};
```

7. I have used K-Means clustering algorithm to form clusters based on input features. Parameters used for K-Means algorithm, are shown in screenshot. We first setup K-Means object and then fit features vector and obtain K-Means model. K-Means model is then transformed to obtain clusters of users. I have setup ClusteringEvaluator to evaluate user clusters and displaying silhouette value. **In order to maximize CTR and ensure all notification ids specified in problem statement are also pushed, I had to increase number of clusters due to which execution time for the algorithm also increased a bit.**

```
// Use K-means algorithm to form clusters for users
System.out.println("Starting K-means algorithm to form clusters....\n");

KMeans KM = new KMeans().setK(20000).setMaxIter(10).setSeed(46L);

KMeansModel KMmodel = KM.fit(userFactorsAsVectors);

Dataset<Row> userClusters = KMmodel.transform(userFactorsAsVectors);

// Evaluate clustering by computing Silhouette score
ClusteringEvaluator evaluator = new ClusteringEvaluator();

double silhouette = evaluator.evaluate(userClusters);
System.out.println("Silhouette with squared euclidean distance = " + silhouette + "\n");
```

8. Select id and prediction columns from the user clusters obtained and cast id as double type and store as userClusters. Create temporary view on user clusters and input dataset to ALS algorithm. Join input dataframe with userClusters on userIndex and id in the inner query and obtain UserID, SongID and prediction as ClusterID. Join this output with newmetadata on SongID and obtain UserID, ClusterID, SongID and ArtistID and store this as clusteredData. Create temporary view on it.

```
// Cast id as DoubleType
userClusters = userClusters.select(col("id").cast(DataTypes.DoubleType), col("prediction"));

// Create temporary view on userClusters and input dataframe
userClusters.createOrReplaceTempView("userclusters");
indexedFinal.createOrReplaceTempView("indexedfinal");

/*
 * Join input dataframe with userClusters on userIndex and id in the inner query
 * and obtain UserID, SongID and prediction as ClusterID. Join this output with
 * newmetadata on SongID and obtain UserID, ClusterID, SongID and ArtistID.
 */
Dataset<Row> clusteredData = sc
        .sql("select cluster.UserID, cluster.ClusterID, cluster.SongID, n.ArtistID from"
                + " (select i.UserID, i.SongID, u.prediction as ClusterID from"
                + " indexedfinal i, userclusters u where i.userIndex = u.id) cluster,"
                + " newmetadata n where cluster.SongID = n.SongID");

// Create temporary view on clusteredData
clusteredData.createOrReplaceTempView("clustereddata");
```

9. Join clustereddata and notificationartistdata on ArtistID to get distinct UserID, ClusterID, ArtistID and NotificationID for given notification ids as per problem statement. This is our NotificationNumber1 data which has original ClusterID and ArtistID for given NotificationIDs along with UserID and NotificationID. Save NotificationNumber1 data in hdfs.

```
/*
 * Join clustereddata and notificationartistdata on ArtistID to get
 * distinct UserID, ClusterID, ArtistID and NotificationID for given
 * notification ids as per problem statement. This is our
 * NotificationNumber1 data which has original ClusterID and
 * ArtistID for given NotificationIDs along with UserID and
 * NotificationID.
 */
Dataset<Row> notificationNumber1 = sc.sql(
        "select distinct a.UserID, a.ClusterID, a.ArtistID, b.NotificationID from clustereddata a,"
        + " notificationartistdata b where a.ArtistID = b.ArtistID and"
        + " b.NotificationID in (9553, 9660, 9690, 9703, 9551)");

// Save NotificationNumber1 data to a file
System.out.println(
        "\nSaving NotificationNumber1 data for specific notification ids as per problem statement, to a file\n");

notificationNumber1.write().mode(SaveMode.Overwrite).format("csv").option("header", "true")
        .save(args[5] + "/" + "NotificationNumber1");
```

10. Select ClusterID and ArtistID from clustereddata and compute count by grouping on ClusterID and ArtitID as cnt_artists in innermost query. Then in immediate outer query, select ClusterID, ArtistID and use analytic function row_number() over partition by clause to partition data on ClusterID order by cnt_artists in descending order so ArtistID with highest count gets 1 as row number. Then in immediate outer query, select ClusterID, ArtistID and use analytic function first_value on ClusterID over partition by clause to partition on ArtistID order by ClusterID as CommonClusterID, from inner query output where row number = 1. Finally select ClusterID, form new ClusterID using CommnClusterID suffixed with '_common', ArtistID as PopularArtistID and store as popularArtistWithNewCluster. Create temporary view on popularArtistWithNewCluster.

    After this step, we will have formed new common clusters with popular artists for each cluster and 1 artist is associated with only 1 new common cluster.

```
/*
 * Select ClusterID and ArtistID from clustereddata and compute count by
 * grouping on ClusterID and ArtitID as cnt_artists in innermost query. Then in
 * immediate outer query, select ClusterID, ArtistID and use analytic function
 * row_number() over partition by clause to partition data on ClusterID order by
 * cnt_artists in descending order so ArtistID with highest count gets 1 as row
 * number. Then in immediate outer query, select ClusterID, ArtistID and use
 * analytic function first_value on ClusterID over partition by clause to
 * partition on ArtistID order by ClusterID as CommonClusterID, from inner query
 * output where row number = 1. Finally select ClusterID, form new ClusterID
 * using CommnClusterID suffixed with '_common', ArtistID as PopularArtistID.
 *
 * After this step, we will have formed new common clusters with popular artists
 * for each cluster and 1 artist is associated with only 1 new common cluster.
 */
Dataset<Row> popularArtistWithNewCluster = sc.sql(
        "select ClusterID, CommonClusterID||'_common' as NewCommonClusterID, ArtistID as PopularArtistID from"
                + " (select ClusterID, ArtistID, first_value(ClusterID) over"
                + " (partition by ArtistID order by ClusterID) as CommonClusterID from"
                + " (select ClusterID, ArtistID, row_number() over"
                + " (partition by ClusterID order by cnt_artists desc) rn"
                + " from (select ClusterID, ArtistID, count(*) as cnt_artists from clustereddata"
                + " group by ClusterID, ArtistID)) where rn = 1)");

// Create temporary view on popularArtistWithNewCluster
popularArtistWithNewCluster.createOrReplaceTempView("popularartistwithnewcluster");
```

11. Join popularartistwithnewcluster with clustereddata on ClusterID to get distinct UserID, NewCommonCluserID and PopularArtistID. This is our UserClusterArtist data. Save UserClusterArtist data in hdfs. Create temporary view on UserClusterArtist.

```
/*
 * Join popularartistwithnewcluster with clustereddata on ClusterID
 * to get distinct UserID, NewCommonCluserID and PopularArtistID.
 * This is our UserClusterArtist data.
 */
Dataset<Row> UserClusterArtist = sc.sql(
        "select distinct a.UserID, b.NewCommonClusterID, b.PopularArtistID from clustereddata a,"
        + " popularartistwithnewcluster b where a.ClusterID = b.ClusterID");

// Save UserClusterArtist data to a file
System.out.println("\nSaving UserClusterArtist data to a file\n");

UserClusterArtist.write().mode(SaveMode.Overwrite).format("csv").option("header", "true")
        .save(args[5] + "/" + "UserClusterArtist");

// Create temporary view on UserClusterArtist
UserClusterArtist.createOrReplaceTempView("userclusterartist");
```

12. Join popularartistwithnewcluster with notificationartistdata on PopularArtistID and ArtistID to get distinct NewCommonCluserID, PopularArtistID and NotificaitonID. Store it as popularArtistNotification. Create temporary view on popularArtistNotification.

```
/*
 * Join popularartistwithnewcluster with notificationartistdata on
 * PopularArtistID and ArtistID to get distinct NewCommonCluserID,
 * PopularArtistID and NotificaitonID.
 */
Dataset<Row> popularArtistNotification = sc.sql(
        "select distinct a.NewCommonClusterID, a.PopularArtistID, b.NotificationID from"
        + " popularartistwithnewcluster a, notificationartistdata b"
        + " where a.PopularArtistID = b.ArtistID");

// Create temporary view on popularArtistNotification
popularArtistNotification.createOrReplaceTempView("popularartistnotification");
```

13. Join userclusterartist with popularartistnotification on PopularArtistID and NewCommonClusterID to get distinct UserID, NewCommonClusterID, PopularArtistID and NotificationID for given notification ids as per problem statement. This is our NotificationNumber2 data for given NotificationIDs. Save NotificationNumber2 data in hdfs.

```
/*
 * Join userclusterartist with popularartistnotification on
 * PopularArtistID and NewCommonClusterID to get distinct UserID,
 * NewCommonClusterID, PopularArtistID and NotificationID for given
 * notification ids as per problem statement. This is our
 * NotificationNumber2 data for given NotificationIDs.
 */
Dataset<Row> notificationNumber2 = sc.sql(
        "select distinct a.UserID, a.NewCommonClusterID, a.PopularArtistID, b.NotificationID from userclusterartist a,"
        + " popularartistnotification b where a.NewCommonClusterID = b.NewCommonClusterID and "
        + " a.PopularArtistID = b.PopularArtistID and b.NotificationID in (9553, 9660, 9690, 9703, 9551)");

// Save NotificationNumber2 data to a file
System.out.println(
        "\nSaving NotificationNumber2 data for specific notification ids as per problem statement, to a file\n");

notificationNumber2.write().mode(SaveMode.Overwrite).format("csv").option("header", "true")
        .save(args[5] + "/" + "NotificationNumber2");
```

14. Push notifications now. Get distinct UserID and NotificationID by joining userclusterartist and popularartistnotification on PopularArtistID and NewCommonClusterID. **By doing so, if a notification is for multiple artists and those artists are popular in different clusters, then all users for all such clusters will be picked up for that notification and we will be able to report common CTR for that notification**. Store it as userNotificationData and create temporary view on userNotificationData.

```
/*
 * Push notifications now. Get distinct UserID and NotificationID by
 * joining userclusterartist and popularartistnotification on
 * PopularArtistID and NewCommonClusterID. By doing so, if a
 * notification is for multiple artists and those artists are
 * popular in different clusters, then all users for all such
 * clusters will be picked up for that notification and we will be
 * able to report common CTR for that notification.
 */
Dataset<Row> userNotificationData = sc.sql(
        "select distinct a.UserID, b.NotificationID from userclusterartist a, popularartistnotification b"
                + " where a.PopularArtistID = b.PopularArtistID and a.NewCommonClusterID = b.NewCommonClusterID");

// Create temporary view on userNotificationData
userNotificationData.createOrReplaceTempView("usernotificationdata");
```

15. Get NotificationID and count from usernotificationdata data group by NotificationID. This will give us count of users where notifications were pushed. Store it as pushedUserCount and create temporary view on pushedUserCount.

```
/*
 * Get NotificationID and count from usernotificationdata data group by
 * NotificationID. This will give us count of users where notifications were
 * pushed.
 */
Dataset<Row> pushedUserCount = sc.sql(
        "select NotificationID, count(*) as pushedCount from usernotificationdata group by NotificationID");

// Create temporary view on pushedUserCount
pushedUserCount.createOrReplaceTempView("pushedusercount");
```

16. Join usernotificationdata with notificationclickdata on UserID and NotificationID and get count of NotificationIDs group by NotitificationID. This will give us count of users who clicked pushed notifications. Store it as clickedUserCount and create temporary view on clickedUserCount.

```
/*
 * Join usernotificationdata with notificationclickdata on UserID and
 * NotificationID and get count of NotificationIDs group by NotitificationID.
 * This will give us count of users who clicked pushed notifications.
 */
Dataset<Row> clickedUserCount = sc.sql(
        "select NotificationID, count(*) as clickedCount from (select distinct b.NotificationID, b.UserID from "
                + " usernotificationdata a, notificationclickdata b"
                + " where a.UserID = b.UserID and a.NotificationId = b.NotificationID) group by NotificationID");

// Create temporary view on clickedUserCount
clickedUserCount.createOrReplaceTempView("clickedusercount");
```

17. Compute CTR - (count of users who clicked pushed notifications / count of users receiving pushed notifications) multiply by 100, by joining pushedusercount and clickedusercount on NotificationID. Store it as ctrData. Save CTR data in hdfs. Create temporary view on ctrData.

```
/*
 * Compute CTR - (count of users who clicked pushed notifications / count of
 * users receiving pushed notifications) multiply by 100, by joining
 * pushedusercount and clickedusercount on NotificationID.
 */
Dataset<Row> ctrData = sc.sql(
        "select a.NotificationID, (b.clickedCount/a.pushedCount) * 100 as CTR from pushedusercount a,"
        + " clickedusercount b where a.NotificationID = b.NotificationID");

// Save CTR data in a file
System.out.println("\nSaving CTR data to a file\n");

ctrData.coalesce(1).write().mode(SaveMode.Overwrite).format("csv").option("header", "true")
        .save(args[5] + "/" + "CTRData");

// Create temporary view on ctrData
ctrData.createOrReplaceTempView("ctrdata");
```

18. Get CTR for specific notification ids mentioned in the problem statement. Store it as ctrDataSpecific. Save CTR data for specific notification ids in hdfs.

```
// Get CTR for specific notification ids mentioned in the problem statement
Dataset<Row> ctrDataSpecific = sc.sql(
        "select NotificationID, CTR from ctrdata where NotificationID in (9553, 9660, 9690, 9703, 9551)");

// Save CTR data for specific notification ids in a file
System.out.println("\nSaving CTR data for specific notification ids as per problem statement, to a file\n");

ctrDataSpecific.coalesce(1).write().mode(SaveMode.Overwrite).format("csv").option("header", "true")
        .save(args[5] + "/" + "CTRSpecificNotificationData");
```

19. Drop all temporary views created so far and print end time.

```
// Drop all temporary views created so far
spark.catalog().dropTempView("useractivitydata");
spark.catalog().dropTempView("newmetadata");
spark.catalog().dropTempView("notificationclickdata");
spark.catalog().dropTempView("notificationartistdata");
spark.catalog().dropTempView("userclusters");
spark.catalog().dropTempView("indexedfinal");
spark.catalog().dropTempView("clustereddata");
spark.catalog().dropTempView("popularartistwithnewcluster");
spark.catalog().dropTempView("userclusterartist");
spark.catalog().dropTempView("popularartistnotification");
spark.catalog().dropTempView("usernotificationdata");
spark.catalog().dropTempView("pushedusercount");
spark.catalog().dropTempView("clickedusercount");
spark.catalog().dropTempView("ctrdata");

// Print End time
System.out.println("\nEnd Time : " + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()) + "\n");
```

20. Output of program has more than 3% overlap with Saavn's data as shown below:

NotificationID	CTR
9586	100.000000
9692	8.042803
9537	7.496823
9553	25.000000
9661	9.077368
9624	18.181818
9667	9.419795
9713	7.788516
9722	4.065041
9551	3.043478
9700	3.693476
9642	14.285714
9690	10.000000
9629	7.586207
9673	9.835323
9659	3.549246
9563	9.626199
9621	0.697618
9717	8.041505
9612	3.038524
9607	3.881567
9703	9.112150
9660	7.759841
9559	12.328767
9707	6.465727

==========THE END==========