Programming for Big Data

Project Report: YouTube Video Analysis

Group Members

Aakar Jinwala: adj329
Sagar Patel: sap590
Sanket Nawle: skn288

Project Idea:

Introduction:

YouTube is one of the most popular web sites on the internet today. The reason being millions of creators coming together to share their content. This content ranges from high budget and amazing movie trailers to silly cat videos. Majority of the YouTube videos are free to the public to view. Also, anyone can upload their own video being an individual or an organization. As a result, the dataset of YouTube is massive and immensely interesting for analysts like us.

Motivation:

YouTube contains a section called as the 'Trending videos'. In this section, the videos which are currently popular and liked by majority of the YouTubers are displayed. This section contains videos from all the categories; be it music or documentaries. However, it may be of little use to the user who wants to search for the videos of his liking. What if the user wants to search the trending videos of 'photography'? Unfortunately, YouTube doesn't provide any such functionality. This seemed like an interesting and challenging problem to be solved. Hence, in this project, we have made an effort to provide this functionality of displaying the trending videos of different categories.

Problem Statement:

1. We can see the categories in Trending section in YouTube. But the way it works is that YouTube categorizes the videos which are in the Trending section. But it doesn't provide the trending videos for each category, i.e., you can see the categories of the videos which are currently trending, but you cannot see the trending videos of different categories. Our project works on providing the latter.
2. We are also doing analysis on hot tags of trending videos which uploaders can use while describing their videos to receive more hits.
3. We have also tried to create a model to predict if the new obtained video will feature in our Categorically Trending Videos or not.

Dataset:

We have used the YouTube Developer API to fetch the video dataset and also other important statistics related to those videos.

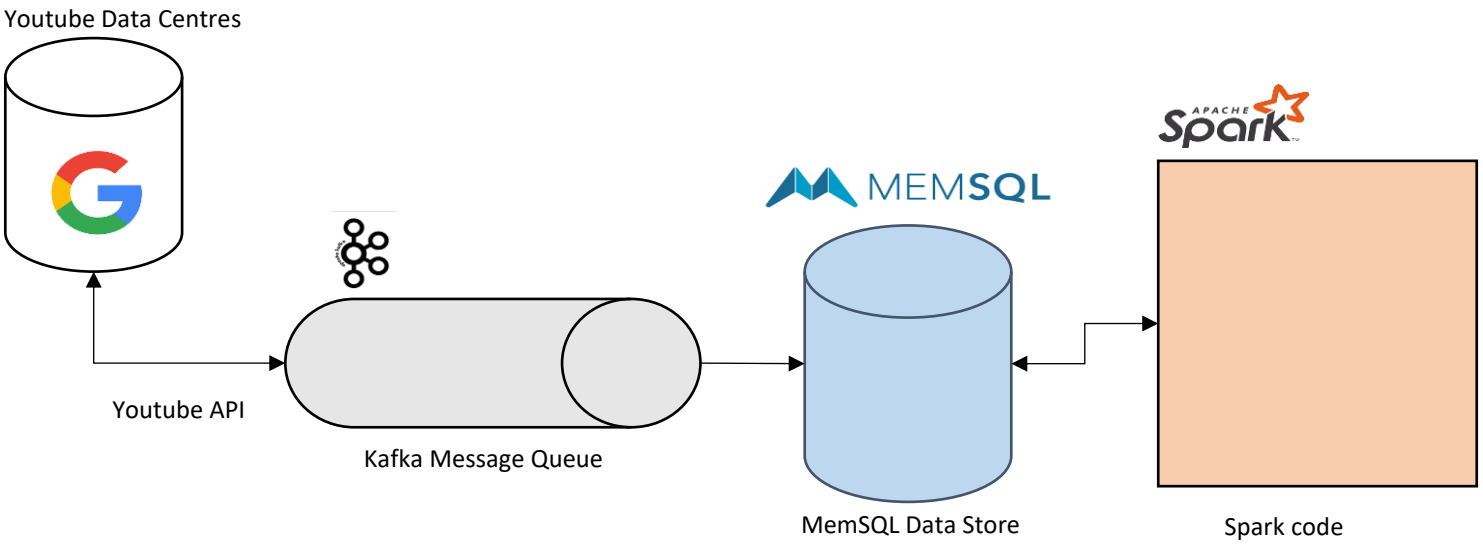
Link providing all the usage of the API is https://developers.google.com/youtube

Data includes the 'videoID', 'url', 'published time', 'channelID', 'category', 'viewCount', 'likeCount', 'dislikeCount' and so on.

Our dataset is about 100MB in size combined.

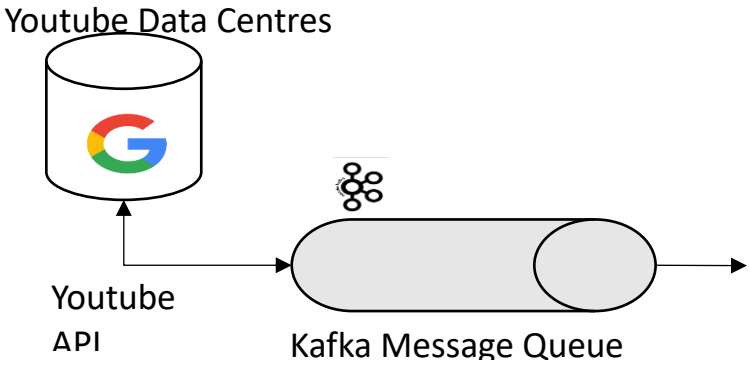
Technology:

Architecture Diagram



Component Break-Down

1. Kafka Message Queue:



Description:

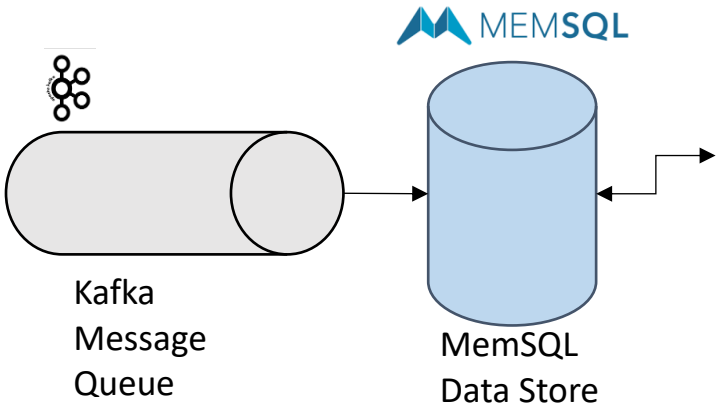
- Kafka Message Bus. It enables us to stream real-time data to our spark code to perform the analysis.
- There are two topics corresponding to 2 types of data which we receive from the API requests.
- The topics currently have 1 partition only. But, we can leverage this parameter to further partition the topics based on categories and so on. This will enable us to parallelize the analysis step further.
- Kafka follows a Producer/Consumer Model.
- We have created 2 topics in Kafka with 1 partition namely ‘videos_topic' and ‘statistics_topic’

Some code snippets:

```
//Producer Config
producer = KafkaProducer(bootstrap_servers= ['localhost:9092']
//Produce to ‘statistics_topic’
```

```
producer.send('statistics_topic', json.dumps(d))
//Produce to ‘videos_topic’
producer.send('videos_topic', json.dumps(data))
```

2. MemSQL:



Description:

- MemSQL is a real-time data warehouse for cloud and on-premises that delivers immediate insights across live and historical data.
- MemSQL is like an in-memory persistent Data Store which has two tables namely Video and Statistics which consume data from Kafka and maintain a persistent storage for the data.
- We have used this component since in our use case we could not perform analysis on live streaming data and required a batch or a set of data to be available.
- MemSQL pipelines consume data from Kafka and insert it into the Tables.

Creating Tables in MemSQL:

Table Video:

```
CREATE TABLE Video(
message JSON NOT NULL,
videoId AS message::$videoId PERSISTED varchar(20),
publishedAt AS message::$publishedAt PERSISTED varchar(20),
channelId AS message::$channelId PERSISTED varchar(100),
category AS message::$category PERSISTED varchar(20),
title AS message::$title PERSISTED VARCHAR(1000),
description AS message::$description PERSISTED varchar(5000),
url AS message::$url PERSISTED varchar(1000),
channelTitle AS message::$channelTitle PERSISTED varchar(100),
PRIMARY KEY(videoId),
SHARD KEY(videoId)
);
```

Table Statistics:

```
CREATE TABLE Statistics(
message JSON NOT NULL,
```

```
videoId AS message::$videoId PERSISTED varchar(20),
viewCount AS message::$viewCount PERSISTED BIGINT,
likeCount AS message::$likeCount PERSISTED BIGINT,
dislikeCount AS message::$dislikeCount PERSISTED BIGINT,
favoriteCount AS message::$favoriteCount PERSISTED BIGINT,
commentCount AS message::$commentCount PERSISTED BIGINT,
PRIMARY KEY(videoId),
SHARD KEY(videoId)
);
```

Creating Pipelines to Connect to Kafka Topics:

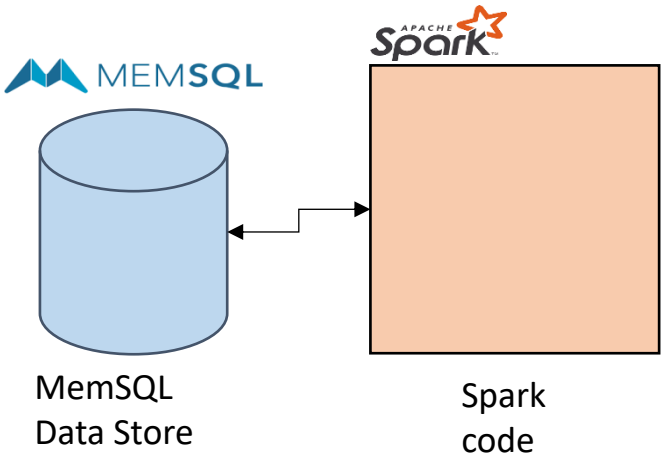
Pipeline for Video Table:

```
CREATE PIPELINE IF NOT EXISTS TEST_Video
AS LOAD DATA KAFKA 'localhost:9092/videos_topic'
BATCH_INTERVAL 2500
SKIP ALL ERRORS
REPLACE
INTO TABLE Video
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY ''
LINES TERMINATED BY '\n' STARTING BY ''
(`message`);
```

Pipeline for Statistics Table:

```
CREATE PIPELINE IF NOT EXISTS TEST_Statistics
AS LOAD DATA KAFKA 'localhost:9092/statistics_topic'
BATCH_INTERVAL 2500
SKIP ALL ERRORS
REPLACE
INTO TABLE Statistics
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY ''
LINES TERMINATED BY '\n' STARTING BY ''
(`message`);
```

3. Spark



Description:

- Spark as we know, one the best technologies to work with Big Data. Even though currently we do not have such huge amount of data, but with YouTube Data we can be safe to assume that we will have to deal with Big Data in practical purposes.

- We have leveraged the Spark SQL abilities to perform various types of analysis on the data we received out of YouTube.
- Spark constantly queries the MemSQL tables to fetch data and perform analysis.
- We can also write out results back into some result table in MemSQL.
- We have used scala to write spark code.

Work Accomplished

Analysis:

1. To find the top trending videos in each category.

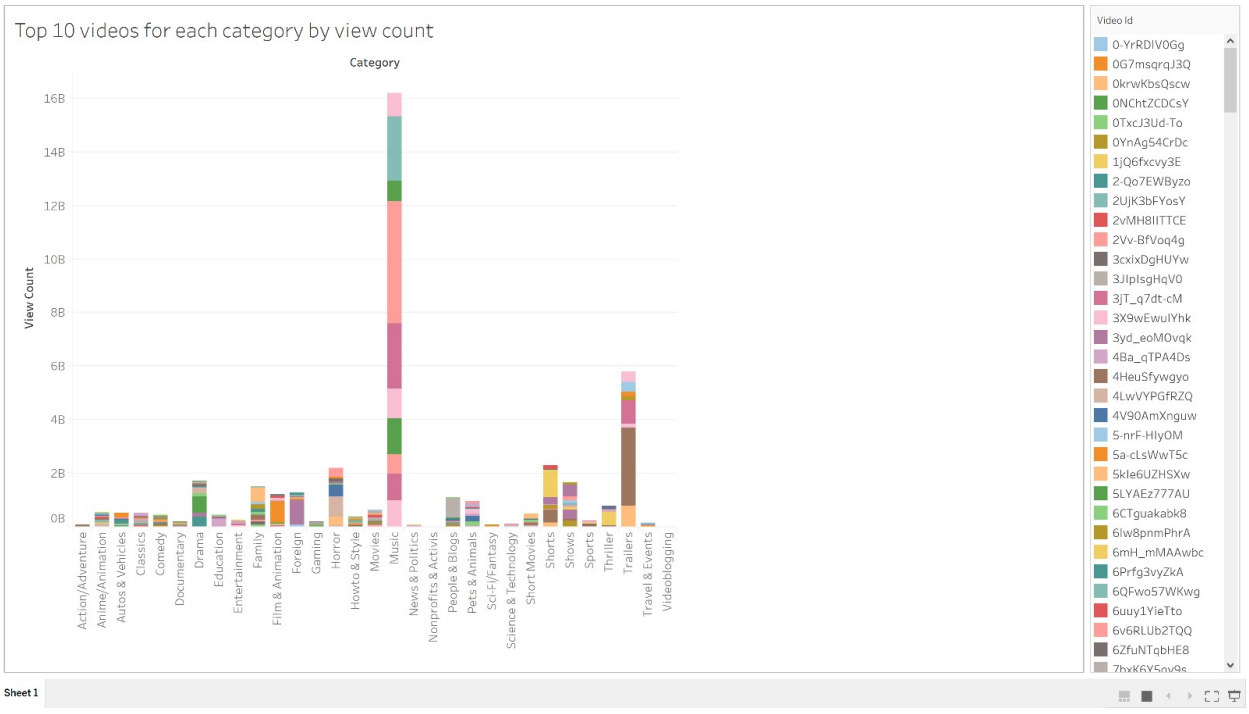
Code snippet (based on View Count):

```
// Part 1 -- Getting the top videos in each and every category on the basis of view counts
val topViewStats =
  """ SELECT videoId, category, viewCount, likeCount
    |  FROM ( SELECT viewCount, videoId, category, likeCount, row_number()
    |         over (PARTITION BY category ORDER BY viewCount DESC) as r
    |         FROM CategoryData as T) as T
    |  WHERE T.r<=10
  """.stripMargin

val topViewStats_DF = spark.sql(topViewStats)

topViewStats_DF.createOrReplaceTempView("viewStats")

topViewStats_DF.coalesce(1).write.format("com.databricks.spark.csv")
  .option("header", "true")
  .save("./Data/topViewStats.csv")
```

Here we get top 10 videos in each category based on View_Count

Output:

Output file : “Data/topViewStats.csv”



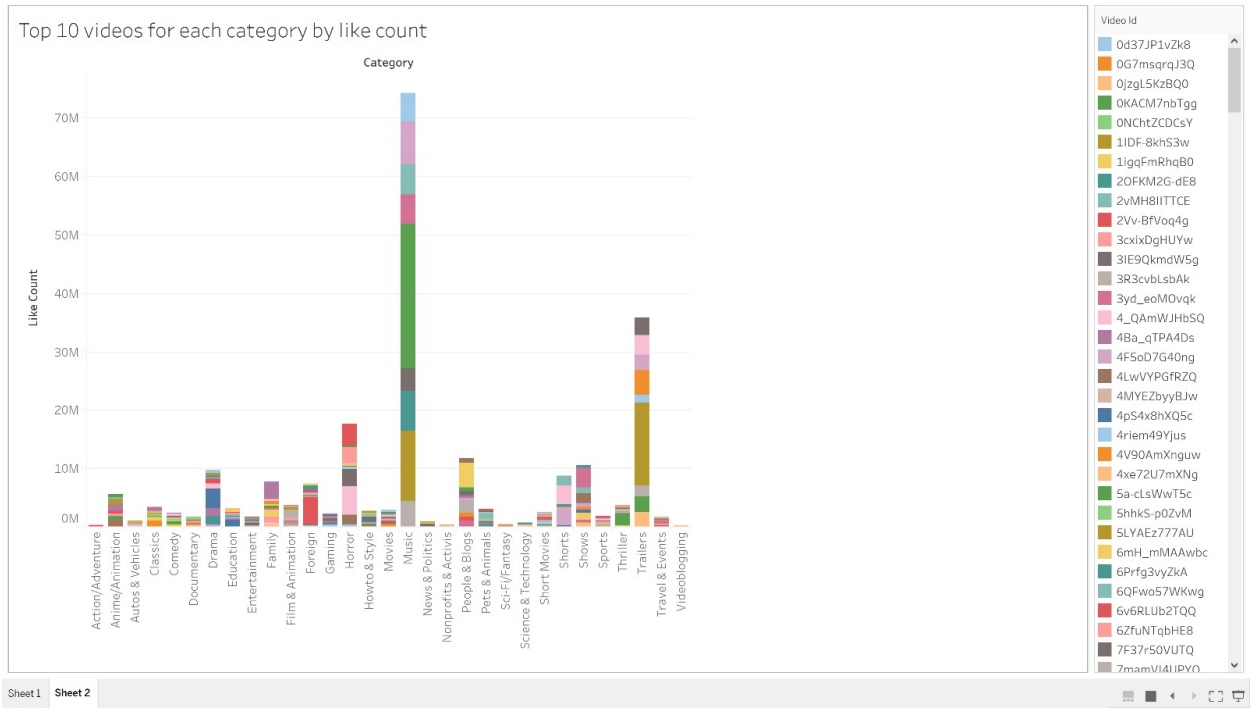
Code snippet (based on Like Count):

```
// Part 2 -- Getting the top videos in each and every category on the basis of Like counts
val topLikeStats =
  """ SELECT videoId, category, viewCount, likeCount
    |  FROM ( SELECT likeCount, videoId, category, viewCount, row_number()
    |         over (PARTITION BY category ORDER BY likeCount DESC) as r
    |         FROM CategoryData as T) as T
    |  WHERE T.r<=10
  """.stripMargin

val topLikeStats_DF = spark.sql(topLikeStats)

topViewStats_DF.createOrReplaceTempView("viewStats")

topLikeStats_DF.coalesce(1).write.format("com.databricks.spark.csv")
  .option("header", "true")
  .save("./Data/topLikeStats.csv")
```

Here we get top 10 videos in each category based on Like_Count

Output:

Output file : “Data/topLikeStats.csv”



2. To find the trending category based on views of videos.

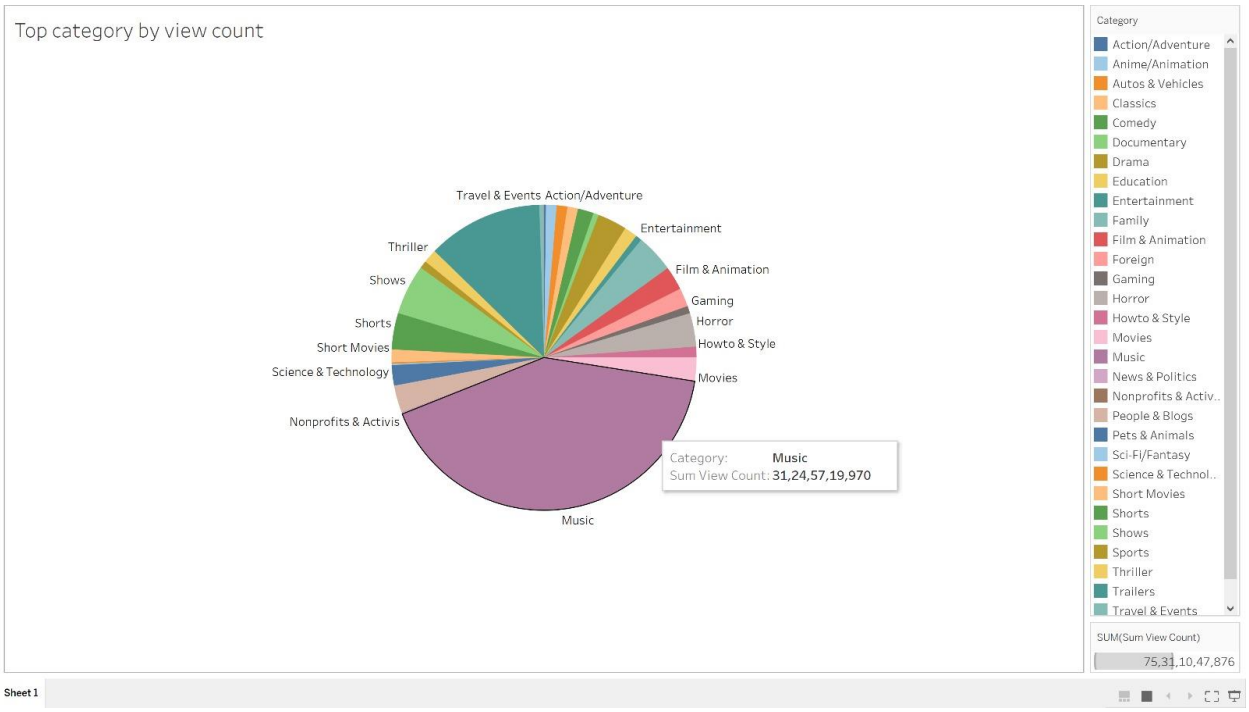
Code snippet:

```
// Part 1 -- Getting the top videos in each and every category on the basis of view counts
val topViewStats =
  """ SELECT category, SUM(viewCount) as sumViewCount
    |  FROM ( SELECT viewCount, videoId, category, likeCount, row_number()
    |         over (PARTITION BY category ORDER BY viewCount DESC) as r
    |         FROM CategoryData as T) as T
    |  WHERE T.r<=100
    |  GROUP BY category
    |  ORDER BY sumViewCount DESC
  """.stripMargin

val topViewStats_DF = spark.sql(topViewStats)

topViewStats_DF.coalesce(1).write.format("com.databricks.spark.csv")
  .option("header", "true")
  .save("./Data/sumTopViewStats.csv")
```

Here we get top category based on the sum of ViewCount of top 100 videos in each category.

Output:

Output file : “Data/sumTopViewStats.csv”



3. Analysis of categories.

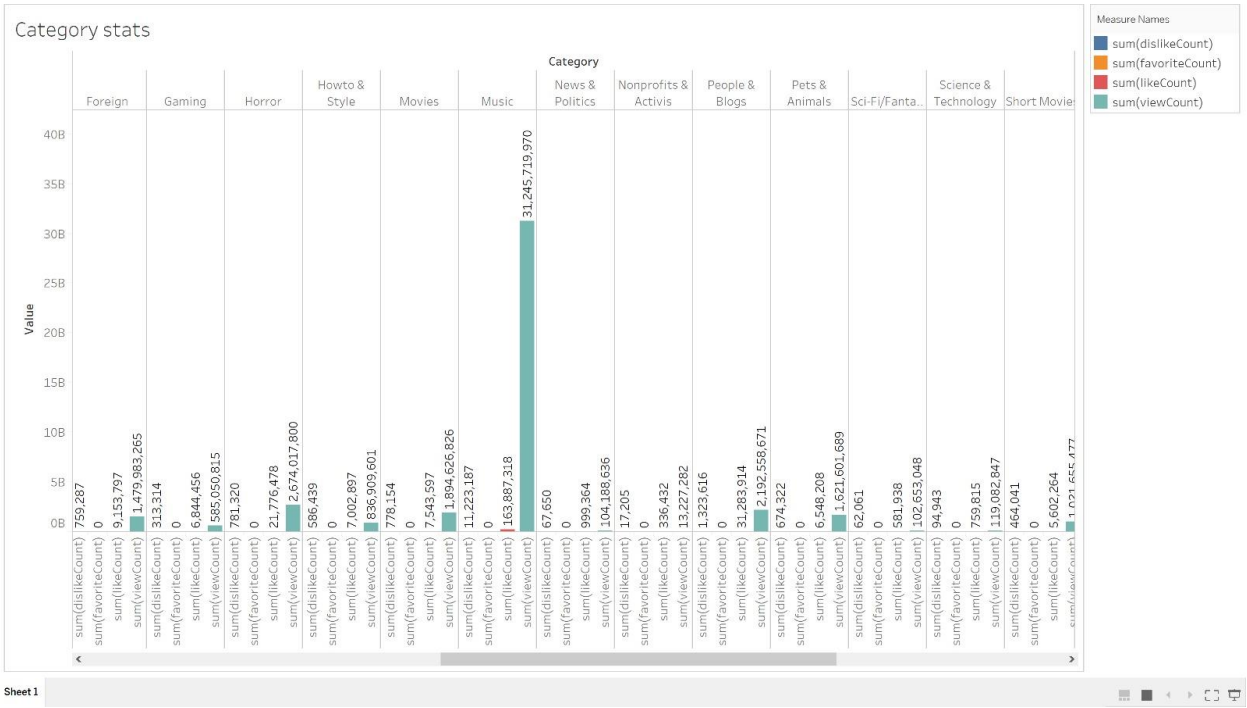
Code snippet:

```
// Part 1 -- Getting the top videos in each and every category on the basis of view counts
val trendStats =
  """ SELECT videoId, category, viewCount, likeCount, favoriteCount, dislikeCount
    |  FROM ( SELECT viewCount, videoId, category, likeCount, favoriteCount, dislikeCount, row_number()
    |         over (PARTITION BY category ORDER BY viewCount DESC) as r
    |         FROM CategoryData as T) as T
    |  WHERE T.r<=100
  """.stripMargin

val trendStats_DF = spark.sql(trendStats)

trendStats_DF.createOrReplaceTempView("trendStats")

val trendSummary =
  """ SELECT category, SUM(viewCount), SUM(likeCount), SUM(dislikeCount), SUM(favoriteCount)
    | FROM trendStats
    | GROUP BY category
  """.stripMargin

val trendSummary_DF = spark.sql(trendSummary)

trendSummary_DF.coalesce(1).write.format("com.databricks.spark.csv")
  .option("header", "true")
  .save("./Data/trendStats.csv")
```

Here we analyze the statistics of each category based on the sum of ViewCount of top 100 videos in each category.

Output:

Output file : “Data/trendStats.csv”



4. Average number of Views and average number of Likes for each category.

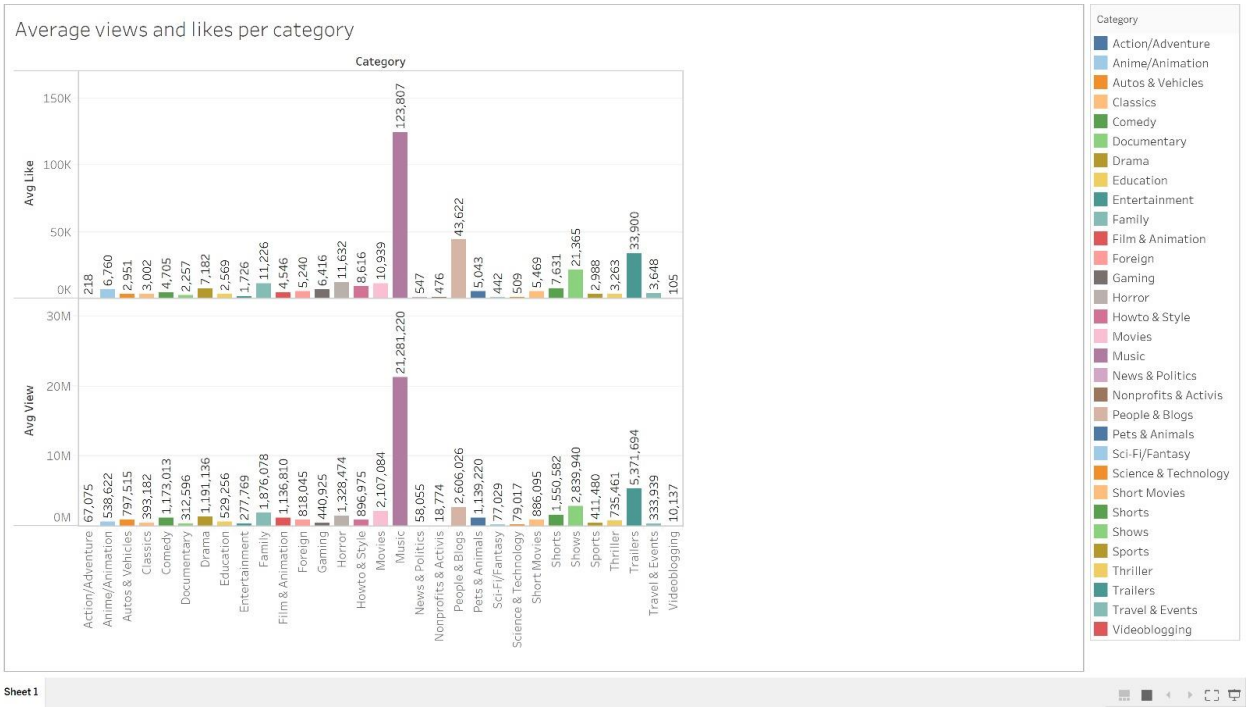
Code snippet:

```
// Part 1 -- Getting the top videos in each and every category on the basis of view counts
val avgLikeViewStats =
  """ SELECT category, ROUND(AVG(viewCount),2) as avgView, ROUND(AVG(likeCount),2) as avgLike
    |FROM CategoryData
    |Group BY category
    |ORDER BY avgView DESC, avgLike DESC """.stripMargin

val avgLikeView_DF = spark.sql(avgLikeViewStats)

avgLikeView_DF.createOrReplaceTempView("avgLikeViewStats")

// Do not truncate for showing the data
avgLikeView_DF.show(100,false)

// Writting the data into a csv file
avgLikeView_DF.coalesce(1).write.format("com.databricks.spark.csv")
  .option("header", "true")
  .save("./Data/avgLikeView.csv")
```

Here we get the avg of Views and Likes of each category.

Output:

Output file : “Data/avgLikeView.csv”



5. Identifying Hot Tags for each Category.

Code snippet:

```
val textfile = r2.groupBy("category").agg(concat_ws(",", collect_list("tags")) as "tags")


for(cat <- 0 to 30){

  val s = textfile.filter(col("category").like(categoryList(cat).toString)).select("tags")
  val p = s.first().getString(0)

  val stringRdd = sc.parallelize(List(p))

  val counts = stringRdd.flatMap(line => line.split(","))
    .map(word => (word, 1))
    .reduceByKey(+)
    .map(item => item.swap) // interchanges position of entries in each tuple
    .sortByKey(false, 1)

  counts.toDF().show()
```

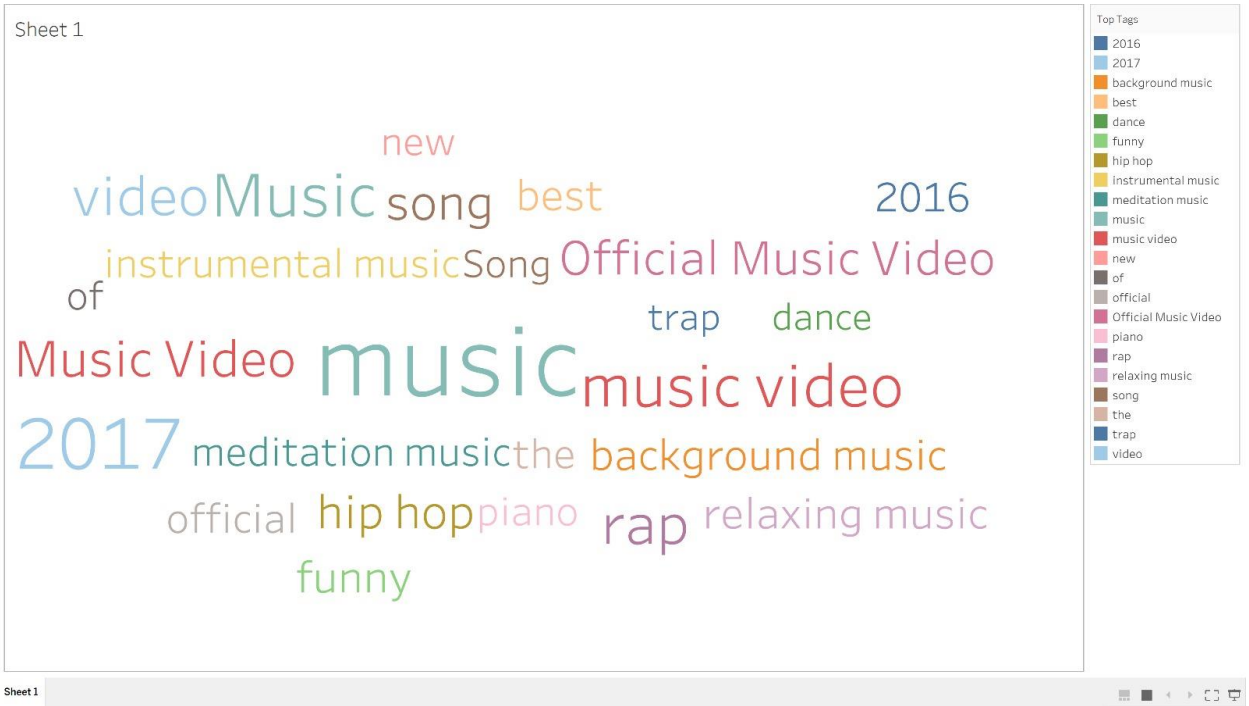
Here we have preformed word count using map-reduce for each category. This gives us the list of hot tags in each category which the uploader could use in his video description helping him get more hits.

This was executed on NYU HPC to leverage the high computing power required to perform this task.

Output:

Output file : “Data/avgLikeView.csv”

Music Category Tags



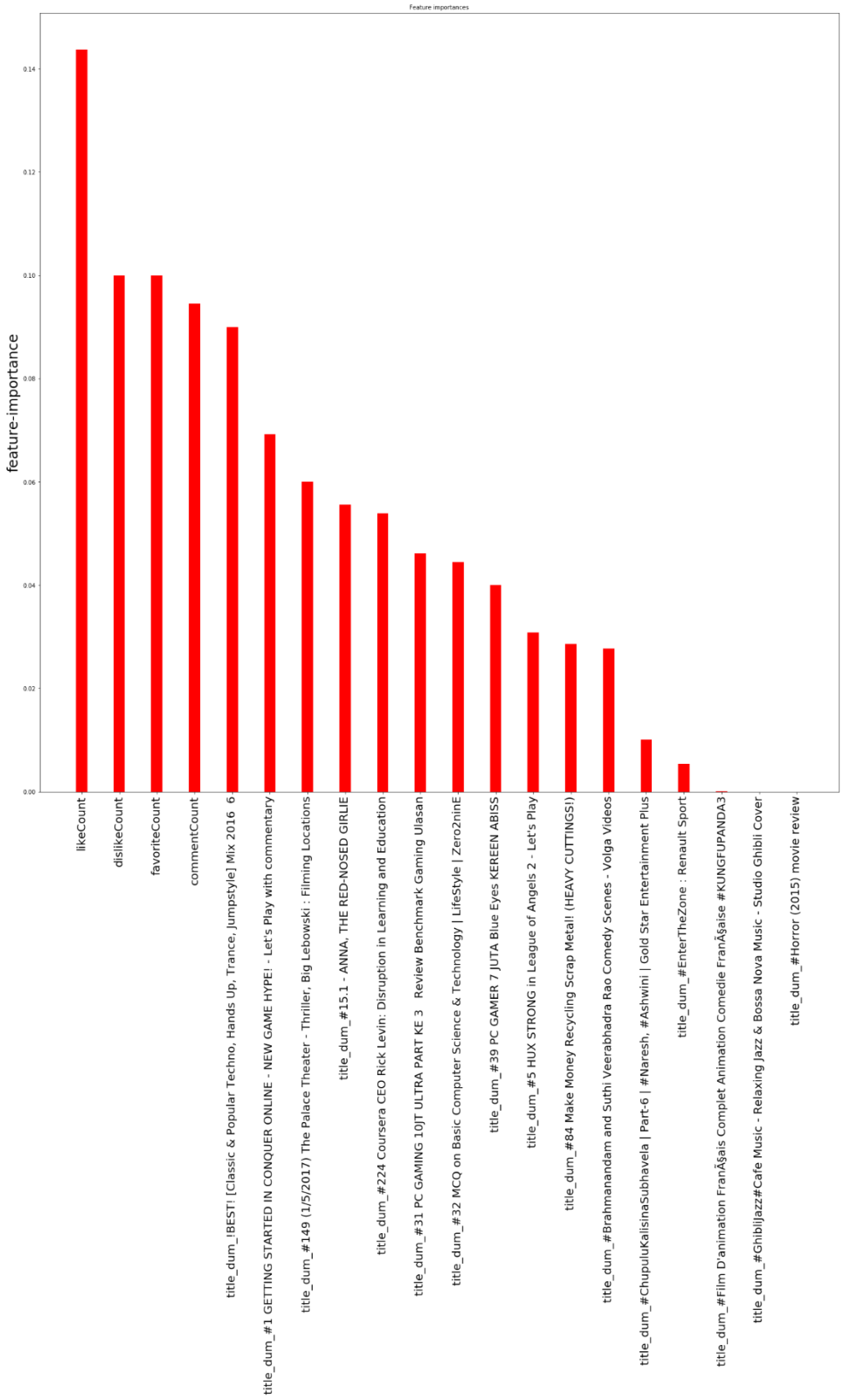
Education Category Tags

Sci-Fi Category Tags

6. Predicting if the new obtained video can feature in the trending video list

In this section, we have attempted to predict whether a newly obtained video will feature in any of the categories in our trending videos. In order to do this, we have used the Jupyter (Ipython) notebook and performed the following steps:

- Imported libraries:
    - csv
    - pandas as pd
    - numpy as np
    - sys
    - from scipy: stats
    - matplotlib.pyplot as plt
    - from sklearn.model_selection: train_test_split
    - from sklearn.ensemble: RandomForestClassifier
    - from sklearn.metrics: accuracy_score
    - from sklearn.metrics: precision_recall_fscore_support
- Read the data containing information about the video such as 'category', 'title', 'like_count', etc (The data read was actually formed by merging of two data sets; data containing descriptive information about the videos and data containing statistical information about the videos)
- Cleaning:
    - Since there were irregularities in the data, we had to drop a few rows to maintain data integrity
    - Converted each attribute to its appropriate data type
- Merging:
    - Merged the data which was present in our final trending videos for each category based on matching the video ids
- Preprocessing:
    - We decided to use the random forest model for our classification problem
    - Unfortunately, the model in scikit learn library doesn't handle categorical (string) attributes implicitly, i.e., the attributes such as 'category' cannot be used directly with the model.
    - As a solution, we resorted to use the dummies library which creates dummy attributes and assigns a value based on whether the video belongs to that category or not, i.e., it creates a column for each category (for example, music, photography, education, etc.) and it assigns the value '1' to that column if the video belongs to that category
- Modeling:
    - In order to do the modeling, we made a train-test split with 75% training data and 25% testing data which is the general norm
    - After trial and error using different combinations of the hyper parameters, we decided to keep the max_depth parameter of the random forest model to be 2 and random_state to be zero. We kept the other parameters as default since the change in those parameters was very trivial
    - After performing the fitting, we have also looked at the feature_importances so that we can understand which attributes gives the most information gain. It turns out that the attributes like 'like_count', 'comment_count' are significant enough while modeling our data. Note: We have dropped the 'view_count' attribute since we have picked our trending videos based on that attribute and it would be unfair and unethical to use that attribute for our classification (we can predict whether the newly obtained video will feature in our trending videos based on this attribute alone)



- In order to understand the performance of our model, we have used the most popular performance metric; 'accuracy'. We have obtained an accuracy of around 99% with both the training and testing set when we used 10,000 samples. The reason of getting such a high accuracy is that most of the times the value for the target variable is 'False' on both cases

Accuracy

```
In [21]: from sklearn.metrics import accuracy_score
         predictions = clf.predict(X_data_train)
         accuracyTrain = accuracy_score(predictions,Y_data_train)
         print("Accuracy from the training data set prediction is",accuracyTrain*100,"%")

         Accuracy from the training data set prediction is 99.8631261976 %

In [22]: predictions = clf.predict(X_data_test)
         accuracyTest = accuracy_score(predictions,Y_data_test)
         print("Accuracy from the testing data set prediction is",accuracyTest*100,"%")

         Accuracy from the testing data set prediction is 99.8768472906 %
```

- Thus, we don't obtain the true score for understanding how our model has performed. In our case, interestingly, it would be important to consider the recall, i.e., understanding how many samples have been misclassified where the value for the target variable is True, but it is classified as False by our model (i.e., measuring the false negative rate). In order to do this, we have used the 'precision_recall_fscore_support' library from the sklearn.metrics library. We managed to obtain a recall of around 99% for our model when we used 10,000 samples which means that our model is quite successful

Precision Recall score

```
from sklearn.metrics import precision_recall_fscore_support
precision_micro,recall_micro,f_beta_score_micro,fscore_micro= precision_recall_fscore_support(Y_data_test, predictions, average='
precision_weighted,recall_weighted,f_beta_score_weighted,fscore_weighted= precision_recall_fscore_support(Y_data_test, prediction

print("Recall micro: {}".format(recall_micro))
print("Recall weighted: {}".format(recall_weighted))
```

```
Recall micro: 0.9987684729064039
Recall weighted: 0.9987684729064039
```