

Music View Predictor (MVP)

- Project Report

Team Members:

Vidit Bhargava, Nandhitha Raghuram, Nishchitha Prasad, Daksh Kumar

Business Understanding:

The US music industry's revenue rose 12% to a total of 9.8 Billion in the past year mainly due to a boost in paid subscription services. Streaming accounted for 75% of the total industry revenue. We separately know that 47% of consumers expect a website to load within two seconds and 40% of users abandon the website if it takes more than 3 seconds to load. The goal of this project is to help streaming services better their content delivery speeds and thus increase user retention. We gather two-day historical data from YouTube and aggregate it with data from different content providers (such as Spotify) and social media platforms to predict the change in the Youtube View Count of each individual video the next day. Each instance is a song at a particular point in time. This allows the Content Delivery Network (CDN) to focus on accurately distributing the videos to local servers which are closer to the users, thus decreasing buffering lags. The model will be used on the state-level and based on the predicted number of views, the CDN can choose which video to send to what depth of locality. For example, a video that is predicted to have a million views can be distributed over local caching servers instead of just keeping it on the main server; whereas a video with only a hundred predicted views will only be kept on the main server. If a music company is using an external CDN service, our project can help them reduce storage costs as well as lower network traffic congestion because each packet has to travel less distance.

Other Use Cases of the project:

- Help record labels identify upcoming artists or efficiently choose which songs to advertise.
- Provide artists an easy way to identify factors that influence the popularity of their songs and allowing them to stay ahead of the curve by experimenting

Data Collection:

To address the business problem we collected data from four different sources and used MongoDB Atlas to store the entire dataset in the Cloud for easy access.

- 1) [Spotify Track DB](#) - This Kaggle Dataset consists of information about songs divided based on Genre. There are over 26 genres with around 10,000 songs belonging to each one. Each song has a unique identifier along with audio features such as danceability,

energy, tempo, mode, etc. However, this dataset is old and did not contain all the details we were looking for. Hence, this served as a base because all the songs were released in the same time period and were well distributed over different genres and artists.

- 2) [Spotify API](#) - To fetch the updated audio analysis, song metadata as well as indexed artist information for each song, we used the Spotify Web API, merging over several functionalities. After generating a Developer Key, we used the unique identifier to query Spotify and update each record in the database with the corresponding information. Spotify allows querying at max 50 songs in each request so we had to batch all the records.
- 3) [YouTube Search API](#) - This API accepts a single string query, which we constructed by combining the song name with the artist's information. It returns a list of songs associated with that query, similar to how the search functionality works on the Youtube website. However, we only need the official Youtube Video associated with the song because the other videos are usually bootlegged. In order to ensure that we picked the right song, we wrote a script to find the song with the most relevance to our query using cosine distance. The main roadblock is that each Youtube API key has a limit on the number of queries it can perform in 24 hours. To bypass this, we generated around 30 keys and implemented an automated script to cycle through these iteratively once each one reaches its limit.
- 4) [Youtube Data API](#) - We used this API to fetch YouTube statistics for each song. Using the Youtube Ids we collected using the Search API, we batch queried the Data API to get the count of corresponding views, likes, dislikes, and comments. However, the API does not return the statistics in the same order as the query, it can also return NULL. This API was run at the same time every 24 hours to get the daily increase in statistics and allow the API limit to reset.

We used the APIs listed above to build the Videos collection. To further illustrate our dataset, we found all the unique artists present in the Videos collection and fetched the number of followers and corresponding popularity score for each artist. This information is maintained separately in the Artists collection.

Future Additions for Data Sources - Through limited trials not included in this report, we also identified that we can enrich our data by collecting information from Twitter about the song and the artists. Twitter provides tweets, queried by hashtags, and sorted based on relevancy, which can be analyzed using NLP techniques or Sentiment analysis to find out if the song is trending/gaining popularity. Recognition on social networking platforms tends to transfer onto media platforms. However, this requires monetary investment.

Data Understanding and Preparation:

The data format in MongoDB is optimized for optimal reads and updates but can not directly be fed into the model. The raw data consisted of NULL and illogical values because some Spotify songs do not have videos associated with them. This section lists out all the steps taken to flatten the nested structures and clean the data.

First, information was retrieved from the Videos collection and the nested dictionary structures were flattened using `json_normalize`. Example:

```
"views" : {  
  "05/04/2020" : {  
    "viewCount" : "2990",  
    "likeCount" : "14",  
    "dislikeCount" : "1",  
    "favoriteCount" : "0",  
    "commentCount" : "0"  
  }  
}
```

Was converted to `views.05/04/2020.viewCount`, `views.05/04/2020.likeCount`, `views.05/04/2020.dislikeCount` and `views.05/04/2020.commentCount`. Flattening the collection results in variable column names for the statistics and since historical data of only the past two days is considered to predict the change in view count, we created a function that used a window size of three days to make each instance and rename all columns with consistency. For example, consider a song that has views from 05/04/2020 to 12/04/2020 (format is dd/mm/yyyy). Instead of having one row for each song, each row would have `day0.viewCount` `day0.likeCount`, `day0.dislikeCount`. Where `day0` would indicate 05/04/2020. This is done for 3 consecutive dates, therefore the row would have `day0.viewCount`, `day1.viewCount`, `day2.viewCount`, where `day0` is 05/04/2020, `day1` is 06/04/2020, `day2` is 07/04/2020. The next row would have the same song and features but `day0`, `day1` and `day2` would correspond to 06/04/2020, 07/04/2020 and 08/04/2020. Similarly, this is done for the other statistics. Now we have values that we can use directly for our analysis instead of values that are in a list of dictionaries.

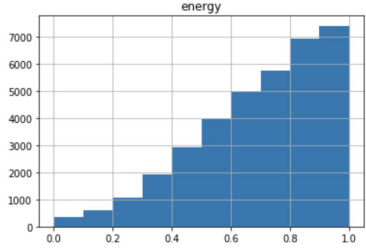
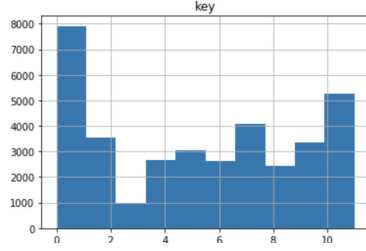
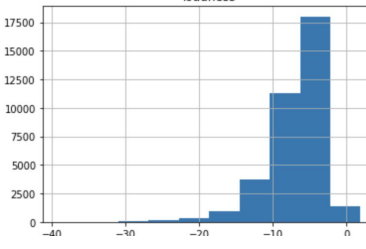
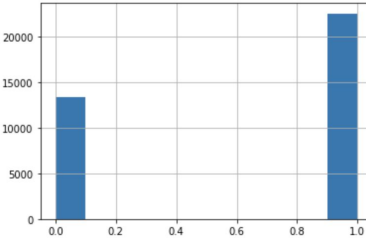
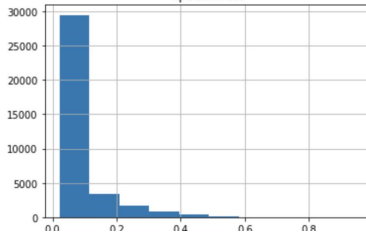
Second, the entire Artists collection was fetched and the two tables were joined using common ArtistIds. Different songs have different numbers of artists, hence all artists' followers and popularity coalesced into two arrays. Third, the columns that are not required are filtered out. Table 1, shows all the fields in consideration of what each one corresponds to. After checking multiple combinations of fields for indicativeness of the target variable, we landed on the percentage change in views, likes, and dislikes. This resulted in the finding that Youtube can sometimes give a lesser number of views for `day1` than `day0`, this issue is caused by the Eventual

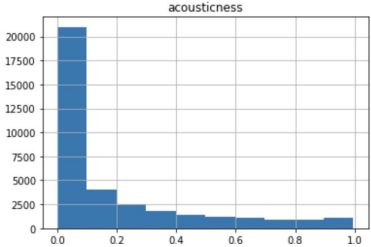
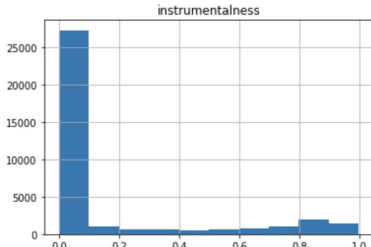
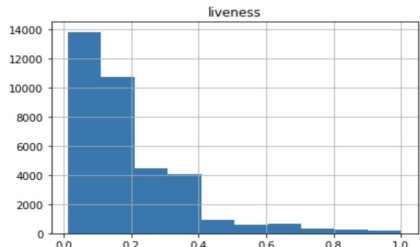
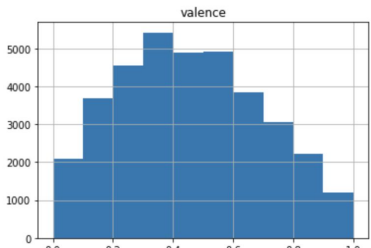
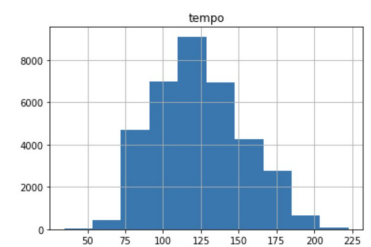
Consistency. Hence, we removed rows where any of the statistics is 0 (resulting in np.inf values) or if the statistics count decreases from one day to the next. Further, to get comparable values we take the percentage change in the values for statistics - the percentage change between day0 and day1, for view count, likes, dislikes. Similarly, we take the percentage change between day1 and day2 for the statistics and add these as features in the table.

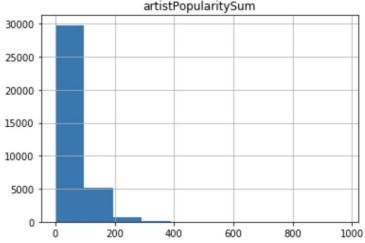
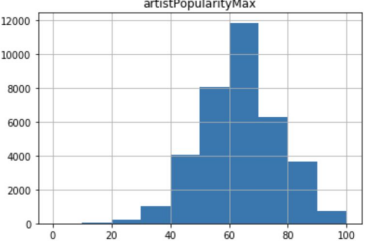
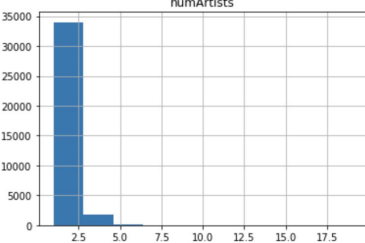
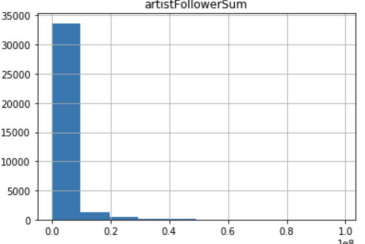
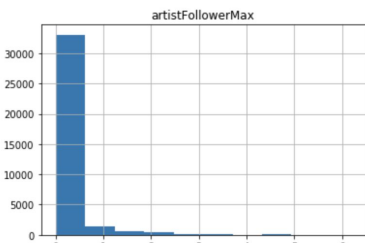
Realizing there was leakage because the models had very high accuracy, we discovered that day2 statistics were seeping in while creating the model and will not be present at deployment, hence we just use it to calculate our target percentage change (which is our target variable) and then we drop it to avoid using it later.

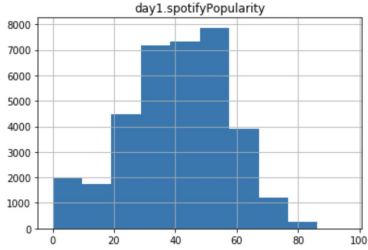
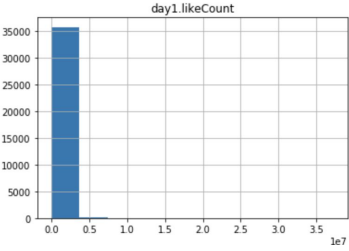
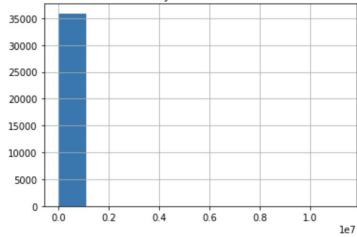
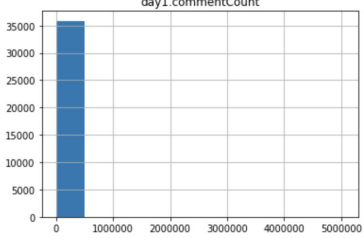
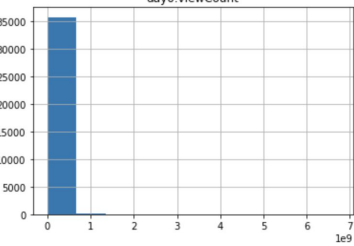
Table 1 - Videos (Fields with similar/related distributions have been combined)

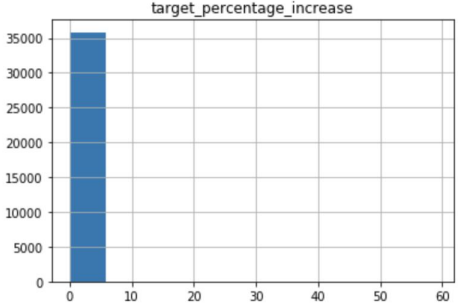
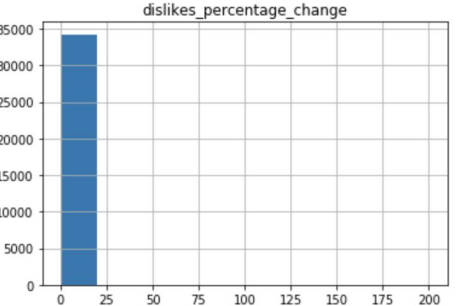
Field Name	Data Type; Explanation	Distribution (if used as a feature)
_id	String; Unique identifier	
name	String; name of the track	
artists	An array of Objects; Dictionary of Artist Id and Name	
albumReleaseDate	String; When the album the song belongs to was released	
youtubeId	String; Youtube Id for the track	
duration_ms	Int; The duration of the track in milliseconds	
artistsPopularity	An array of Int; Contains the popularity of each artist in an array.	
artistFollowers	An array of Int; Contains the number of followers of each artist in an array.	
views	Object; Dictionary with the key as the date the statistics were collected on and the value as a dictionary of statistics	

energy	Double; Represents a perceptual measure of intensity and activity.	
key	Int; Estimated overall key of the track. Integers map to pitches using standard Pitch Class notation	
loudness	Double; The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track.;	
mode	Int; Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived	
speechiness	Double; The presence of spoken words in the track, a value above 0.66 indicates that the song is mostly spoken words.	

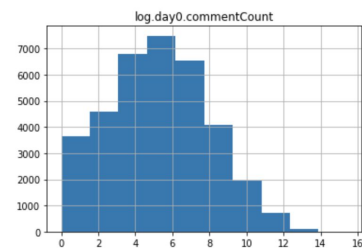
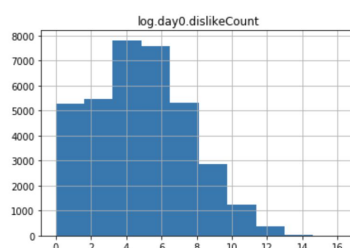
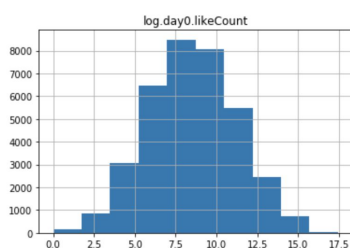
acousticness	Double; The confidence level of whether the song is acoustics	
instrumentalness	Double; Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context.	
liveness	Double; Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live.	
valance	Double; Describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric)	
tempo	Double; The overall estimated tempo of a track in beats per minute (BPM)	

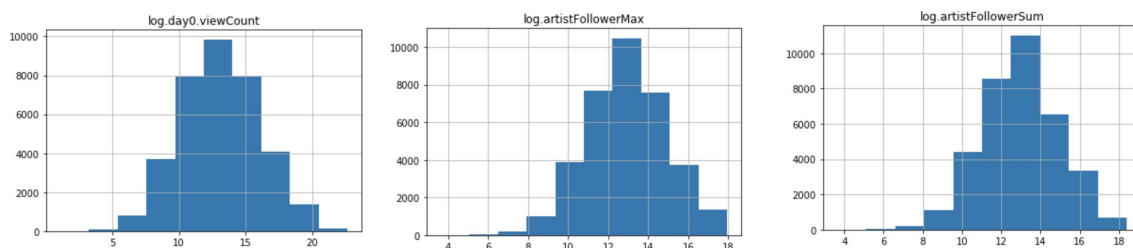
artistPopularitySum	Int; The sum of the array of artist popularity. This was calculated by the artistPopularity field	 <p>A histogram titled 'artistPopularitySum' showing the frequency of popularity sums. The x-axis ranges from 0 to 1000 with increments of 200. The y-axis ranges from 0 to 30,000 with increments of 5,000. The distribution is highly right-skewed, with a peak frequency of approximately 30,000 for the first bin (0-100) and a long tail extending towards 1000.</p>
artistPopularityMax	Int; The maximum popularity, from the array of artist popularity. This was calculated by the artistPopularity field	 <p>A histogram titled 'artistPopularityMax' showing the frequency of maximum popularity values. The x-axis ranges from 0 to 100 with increments of 20. The y-axis ranges from 0 to 12,000 with increments of 2,000. The distribution is roughly bell-shaped, peaking at approximately 11,500 for the bin between 60 and 70.</p>
numArtists	Int; The number of artists associated with the song.	 <p>A histogram titled 'numArtists' showing the frequency of the number of artists associated with a song. The x-axis ranges from 0 to 17.5 with increments of 2.5. The y-axis ranges from 0 to 35,000 with increments of 5,000. The distribution is highly right-skewed, with a peak frequency of approximately 34,000 for the first bin (0-2.5) and a long tail extending towards 17.5.</p>
artistFollowerSum	Int; The sum of the array of artists' followers. This was calculated by the artistFollowers field	 <p>A histogram titled 'artistFollowerSum' showing the frequency of the sum of artists' followers. The x-axis ranges from 0.0 to 1.0 with increments of 0.2, and is scaled by 1e8. The y-axis ranges from 0 to 35,000 with increments of 5,000. The distribution is highly right-skewed, with a peak frequency of approximately 34,000 for the first bin (0.0-0.1e8) and a long tail extending towards 1.0e8.</p>
artistFollowerMax	Int; The maximum number of followers of an artist from the array of artist followers. This was calculated by the artistFollowers field	 <p>A histogram titled 'artistFollowerMax' showing the frequency of the maximum number of followers of an artist. The x-axis ranges from 0 to 6 with increments of 1, and is scaled by 1e7. The y-axis ranges from 0 to 30,000 with increments of 5,000. The distribution is highly right-skewed, with a peak frequency of approximately 32,000 for the first bin (0-1e7) and a long tail extending towards 6e7.</p>

day0.spotifyPopularity, day1.spotifyPopularity	Int; The popularity of a track (value between 0 to 100)	
day0.likeCount, day1.likeCount	Int; The number of likes received by a youtube video.	
day0.dislikeCount, day1.dislikeCount	Int; The number of dislikes received by the youtube video.	
day0.commentCount, day1.commentCount	Int; The number of comments for a youtube video	
day0.viewCount, day1.viewCount, day2.viewCount	Int; The number of views for a youtube video.	

First_percentage_change, target_percentage_change	Double; The difference in number of views on day0 and number of views on day1	
Likes_percentage_change, dislikes_percentage_change	Double; The difference in the number of likes/dislikes on day0 and day1	

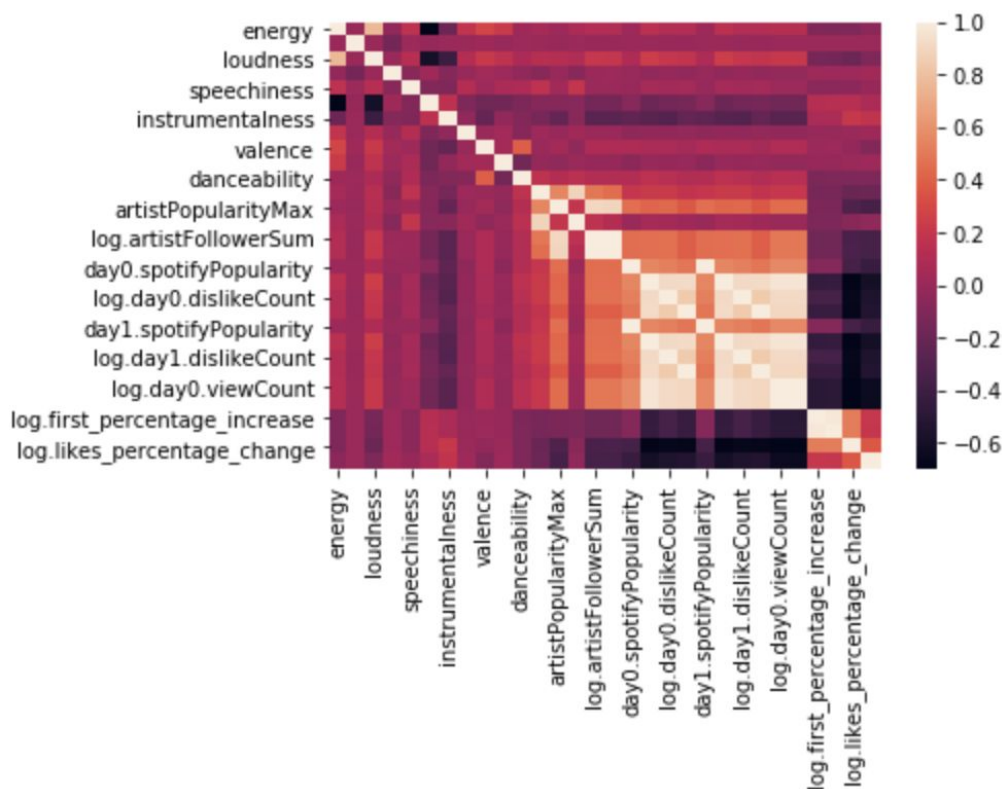
As shown by the distribution graphs given in the table, some columns have highly varied ranges with values from zero to over hundred million or zero to negative forty. High range inversely affects regression models putting extra weightage on high values thus the data had to be standardized. Another issue was that the data was highly skewed in some columns, especially the ones with larger ranges. Intuitively, there would be more videos with lesser number of views than higher number of views with only a few going in hundred of millions. After trying several nonlinear transformations like PowerTransformer (caused distorted correlations and distances within and across features) to manipulate the data, we finally landed on taking the log value of these columns.





After cleaning and reorganizing the data, we now try to form an understanding of how each feature is related to others and if we can derive important information from them.

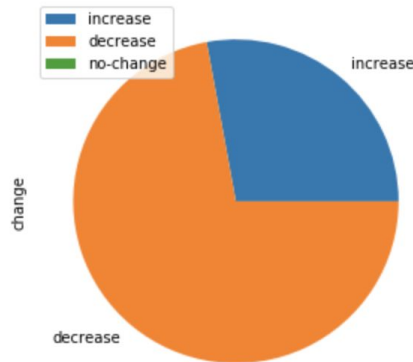
We first start by understanding how each feature is correlated with the other. The following shows a Pearson correlation matrix. A Pearson correlation is a number between -1 and 1 that indicates the extent to which two variables are linearly related.



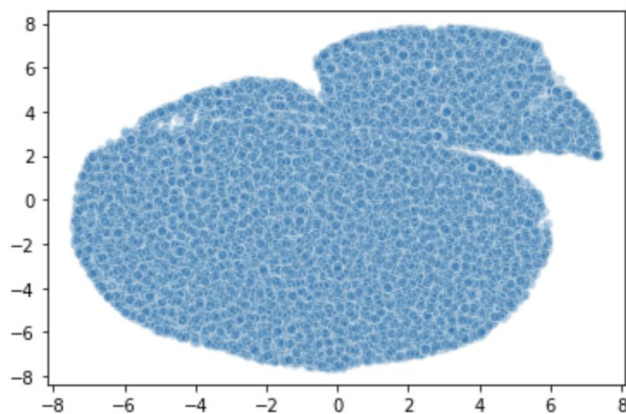
The above graph shows that there is a correlation between day0 and day1 statistics, namely likesCount, dislikeCount, viewCount, and commentCount and that these are indicative of predicting the target variable. The darker squares appear because we use percentage increase which is divided using the corresponding count. It can also be observed that some Spotify audio features are more correlated to the Like count (instead of View count) than others namely, danceability, loudness, and energy. While several Spotify features are related to each other like

Valence and Energy, Speechiness, and Popularity. The maximum and sum of Artist Followers on Spotify also relates to the view count on Youtube

It was also discovered that most tracks see a decrease in percent change as days progress.



t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data. We used the technique to reduce the number of dimensions to two and plot the result on a 2D graph.



The graph above, besides looking like a duck, shows some semblance of clusters in the data. We cross-referenced random instances with their analogous t-SNE values to derive meaningful insights. The upper right of the graph (the beak) shows the songs which already have a high number of views/engagement. The major oval consists of two distinct clusters based on the increasing Spotify track and artist popularity.

Modeling and Evaluation:

In order to predict the percentage increase in views of a particular song, its previous day's views and other data are used along with the other Spotify attributes. Since the values of views range from a few hundred to millions, the data is normalized based on the following 4 types of normalization:

1. **Robust normalization:** Since the data might contain many outliers, scaling using the mean and standard deviation of the data is likely to not work very well. Robust scaling is suitable as it removes the median and scales the data according to the quantile range in unit variance.
2. **Min-Max normalization:** Min-Max rescales the data set such that all feature values are in the range $[0, 1]$. Due to various factors, there might be an outburst in the number of views, likes, dislikes, and comments. This is squelched in Robust normalization. A comparison needs to be made with respect to the behavior of the outliers. Since Min-Max is sensitive to outliers, we can understand the behavior of outliers.
3. **Z-Score normalization:** Z-Score compares results to a "normal" population and gives an idea of how far from the mean a data point is. It is calculated by subtracting the data with the mean and dividing it by standard deviation. This method did not yield satisfactory results as the viewCount, likeCount, dislikeCount, commentCount varies from few tens to millions.
4. **MaxAbs normalization:** Similar to Min-Max scalar, MaxAbs rescales the data set in the range $[-1, 1]$, which implies that this is more sensitive to outliers and might yield prejudiced results.

Different types of models are used to predict the `log.target_percentage_increase`

The following set of models were implemented:

1. **Decision Regression Tree:** Decision tree regression applies the principles of regression on decision trees where it observes the feature variables and trains the model in the structure of a tree to predict and produce meaningful continuous output which is appropriate to predict the `target_percentage_increase`. The main disadvantage here is that each split in a tree leads to a reduced dataset under consideration. And, hence the model created at the split will potentially be biased. Also, the result of the greedy strategy applied by the decision tree's variance in finding the right starting point of the tree can greatly impact the final result.
2. **K Nearest Neighbor:** K nearest neighbors stores all available cases and classifies new cases based on a similarity measure. KNN has been used in statistical estimation and pattern recognition which aids in understanding the data better and new data can be added seamlessly. The downside of this though is that KNN does not perform well for large datasets and datasets with outliers which leads to overfitting of the data.

3. **Ridge Regression:** Ridge Regression is best used for analyzing multiple regression data that suffer from the existence of near-linear relationships among the independent variables. When this multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. Though this might work well in most situations, Ridge regression uses L2 regularization which squishes the features.
4. **Lasso Regression:** Lasso regression is a type of linear regression that uses shrinkage where data values are shrunk towards the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters) and is well-suited for showing high levels of multicollinearity. This also uses L1 regularization which does not penalize much for outliers.
5. **Linear Regression:** Linear regression is a linear approach to modeling the relationship between feature variables and the target variable. This is simple to implement but the errors and outliers are not penalized enough.

Based on Grid Search Score, the best values for min-samples-leaf of the Decision Regression Tree, the number of neighbors in the Nearest Neighbor model, and the alpha values of Ridge and Lasso regression is determined. Using these values, training and testing scores and root-squared-mean errors are determined for each of the models. The best model is chosen for prediction.

The values for these are given in the table below:

Model and best GridSearched parameter value	Train Score (R2)	Test score (R2)	Test (RMSE)
Robust normalization (RMSE Base Rate = 0.02906, R-Squared Base Rate = 0.94484)			
Decision Regression Tree (min-samples-leaf: 100)	0.9517	0.9464	0.02840
Nearest Neighbor (Neighbors: 100)	1.0	0.6857	0.1677
Ridge Regression (alpha: 0.1)	0.9489	0.9467	0.02745
Lasso Regression (alpha: 0.001)	0.9487	0.9464	0.0275
Linear Regression	0.9489	0.9466	0.0280

Min-Max normalization (RMSE Base Rate = 0.01057, R-Squared BaseRate = -0.2726)			
Decision Regression Tree (max-depth: 100)	0.9517	0.9464	0.0004
Nearest Neighbor (Weight: distance Neighbors: 100)	1.0	0.5723	0.00359
Ridge Regression (alpha: 0.001)	0.9489	0.9466	0.00043
Lasso Regression (alpha: 0.001)	0.9387	0.93801	0.00051
Linear Regression	0.9489	0.9466	0.336

Root Mean Square Error (RMSE) can be interpreted as the standard deviation of the unexplained variance and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate a better fit. The root mean squared error is more sensitive than other measures to the occasional large error. R-squared is simply the fraction of the response variance that is captured by the model. Comparing both, R-squared is a relative measure of fit, RMSE is an absolute measure of fit. For our purpose, we need a scoring methodology that does not disregard outliers, in fact, penalize heavily if the predictions are way off because they are essential to our business problem.

While observing the scores of the model, the following observations can be derived:

1. The Nearest Neighbor model is overfitting the training data yielding a training score of 1.0 across normalizations. This shows that the model will not be able to predict the target variable appropriately for new values.
2. Though Lasso and Ridge regression models are performing well, they do not perform well with outliers, which is a huge possibility in this case
3. Linear regression yields an RMSE value closer to the base rate or is higher than the base rate. Hence, the model is not very good at predicting the target value.
4. Decision Tree Regression yields an RMSE value closer to the base rate in Robust scaling but performs well with Min-Max scaling.

Hence, it is better to choose the Decision Tree Regression model with Min-Max scaling.

The return on investment is hard to calculate because the model predictions have to be combined with Live data to actually see the repercussions. We would eventually want to

evaluate the increase in customer satisfaction and retention. We can also evaluate by A/B testing the deployment of the model and evaluating the difference in buffering speeds when the model is actually in use. In the meantime, we can also test the model by evaluating the amount by which it actually predicts the delta views instead of the log of percentage increase. This is explained more in the Deployment Section

Deployment:

The model will be deployed on different levels of geographical servers with relative thresholds. These thresholds depend on the number of active users at each level. It will be used in conjunction with live analytics through which we get the statistics at the corresponding level. The model predicts the log of percentage increase from the current to the next day. Each prediction is converted by taking the inverse log and using that with the current day views to calculate the exact increase in the number of views. Lesser frequently listened to tracks trickle up the server hierarchy. We assume that going up the hierarchy, it is cheaper to cache songs.

For example,

1. Neighborhood-level servers - If the view increase is more than 50% of the number of active users in the neighborhood, we cache the track at this level
2. State-level servers - If the view increase is more than 40% of the number of active users in the state and it hasn't been cached in all neighborhoods, we cache the track at this level
3. Region-level servers - If the view increase is more than 35% of the number of active users in the region and it hasn't been cached in all states, we cache the track at this level.
4. Country-level servers - If the view increase is more than 31% of the number of active users in the country and it hasn't been cached in all regions, we cache the track at this level.

The listening patterns are subject to change, thus the model has to be retrained periodically and deployed at each level. The instances we have used were released one year back and the model could be trained using more recent data without significant investment. The model does not perform well on songs that are released within five days of the current date. This problem can be mitigated by constructing a different model for those kinds of tracks. Also, the threshold levels need to be experimented with, in the real world and readjusted accordingly as they are currently set based on the distribution. If the model is doing well and we do see a significant decrease in latency/increase in buffering speeds, we can look into increasing the number of features by looking into other social media platforms and Google Search hits.