Evan Magee
Professor Janghoon Yang
CmpSc 445 - 001
10/25/2025

1. Project Description

The goal of this project was to test the difference in data collection methods for a youtube video to see how well that data can be used to determine the potential category or predict a given numerical attribute of a video given certain parameters. In a world where online content is more prevalent than ever. From content creators to advertisers, they've built their entire brands on engagement so there is an increasing need to try and figure out what increases this engagement.

2. How to use

- Project Programs
  - DataCollection.py
    - Uses YouTube API provided by google to collect 3000 videos
  - WebScraper.py
    - Uses Selenium and BeautifulSoup to scrape video links from YouTube search results
  - VideoScraper.py
    - Uses Selenium and BeautifulSoup to scrape different attributes from video links collected by WebScraper.py
  - Preprocessing.py
    - Preprocesses data collected by VideoScraper.py & DataCollection.py and also executes feature engineering for both.
    - Creates csv files for each as well as a csv with data that is shared between both.
  - ModelDevelopment.py
    - Develops an XGBoost model for predicting a given numerical attribute and a Logistic Regressor model for predicting category for both models.
    - Tests their performance on the data inside the matching database to better compare performances.

- Training
  - The model training contains many different dependencies, many of which come from the sklearn package.
    - Pandas is important for reading the csv files which contain all the datasets with all the video attributes in them.

        - Numpy is used for many of the preprocessing sets that take place inside of ModelDevelopment.py.
        - Seaborn is used for the visualization of the models
        - XGBoost is used as the training model for predicting numerical models
        - Matplotlib is used for visualization
        - Sklearn provides many different functions from data preprocessing to measuring model performance.

- Inference
    - Inputs
        - The programs don't require any manual inputs but the files do rely on each other. VideoScraper.py uses a csv file created by WebScraper.py to collect different video attributes. Preprocessor.py requires 2 csv files to run, one from VideoScraper and one from DataCollection.py. Finally ModelDevelopment.py requires 3 csv files, all of which are created by Preprocessor to run.
    - Outputs
        - Each model outputs 5 visualization to help interpret performance and feature importance
            1. XGBoost: Actual vs Predicted Scatter Plot
            2. XGboost Feature Importance Plot
            3. Logistic Regression Confusion Matrix
            4. Logistic Regression Feature Importance Plot
            5. Engagement by Category Bar Plot
        - After both models have finished training and testing a Model Performance Comparison Bar Plot is output to help compare the performances of the database models.

3. Data Collection

For the data collection in this project, I used a large variety of different tools and libraries that were at my disposal. I used BeautifulSoup and re for scrapping through the html to find different attributes. Selenium is used for loading and traversing YouTube pages to then be loaded by BeautifulSoup, it is also used to scroll the pages to retrieve things like videos in a search bar. Pandas is used to turn retrieved data into a csv file to later be used for preprocessing. All of the found videos from the search results, were then added into a large dataframe which was outputted as a txt file of different youtube

links. This allowed the task of searching for videos and grabbing data from said videos to be separated into separate programs.

Youtube API, provided by google, was used for collecting YouTube video data directly without needing to use webscraping. It performed way faster than the web scraping function which incorporated video loading, and took about 3 hours to create a csv file. The problem with this API however is that it is very limited by google, allowing for only 10,000 scrapes per day. This is due to it being the free version of the API so while it is annoying for this project, in a real world setting. So, for a company that heavily uses this and owns paid licenses to this API, it shouldn't be as limited.

The csv file for web scraping contained 3005 entries and the csv file for the API contained exactly 3000 entries. One thing of note I found interesting about the web scraping entries was that it would collect around 10000 youtube results but would only accept about ¼ of them. This is due to most of the youtube results being shorts instead of videos which contain a different html layout then a normal youtube video.

Example Rows

- Turned into columns to fit in documents

Example	1	2
views	76642269.0	17430015.0
likes	1546159.0	37107.0
duration	239	183
category	6	6
has_questions	0	0
has_exclamations	0	0
has_numbers	0	0
has_emoji	0	0
title_caps_ratio	0.0	0.4285708163274053
emoji_count	0	2
title_word_count	9	6
title_char_count	54	37
clickbait_count	0	0
clickbait_ratio	0.0	0.0
has_clickbait	0	0

des_caps_ratio	0.0	0.06611569701523165
des_word_count	61	107
des_char_count	756	1085
des_avg_word_len	11.114752276270117	8.89719543016865
channel_word_count	2	2
channel_char_count	12	10
channel_freq	1	1
tags_count	9	10
days_since_upload	16	82
upload_month	10	7
upload_hour	23	16
is_weekend	1	0
weekday_0	0	0
weekday_1	0	0
weekday_2	0	0
weekday_3	0	1
weekday_4	0	0
weekday_5	0	0
weekday_6	1	0
category_encoded	6	6

Collected Attributes:

- Video Id
  - The value at the end of a youtube link after ‘/watch?v=’
- Title
  - The title of a given youtube video
- Description
  - The description of a given youtube video
- Channel Title
  - The name of the channel the video is posted on
- Tags
  - Tags linked to a given youtube video
- Views

    - The number of views on a given youtube video
- Likes
    - The number of likes on a given youtube video.
- Upload Date
    - The date at which a given youtube video was uploaded.
- Search Prompt or “Category”
    - The search term used to collect a given youtube video with.

4. Data Preprocessing

1. Cleaning

➢ All data with missing values or NaN were dropped as filling those values would’ve given cases where attributes like “like” for example would be greater than the number of “views”. Missing data also indicated the videos were not loading properly during scraping or were instances of livestreams which handle metrics differently.

➢ Any video that contained more likes than views was dropped as it indicated bot activity on and is disingenuous. Same goes for videos with a lot of views but no likes, especially if they were very recently uploaded.

2. Normalization

➢ For the categorical models, the data was put through the StandardScaler so that all the data like views, which can be a very large number, don’t overshadow smaller numbers like say of the week when training the model.

➢ For the numerical models, the data was log-transformed using numpy (np.log1p()) to help stabilize the variance before its used for training.

3. Conversions

➢ Category was converted to numerical presentation using LabelEncoder to be compatible with model input requirements.

➢ The upload day was converted to the day of the week, using one hot encoding so that the day of the week a video was uploaded could have an effect on the model.

4. Filtering

➢ Overlapping entries were removed from both models and placed into its own dataset. This dataset would then be used as testing data for both models to better compare their results together.

➢ Videos with a length over 6 hours were removed as they are very uncommon and can cause distorted feature scaling for duration

5. Feature Engineering

1. Calculated Attributes:
    - ➢ **has_questions**
        - If title contains a question mark or not
        - 1 or 0
    - ➢ **has_exclamations**
        - If title contains an exclamation mark or not
        - 1 or 0
    - ➢ **has_numbers**
        - If title contains numbers
        - 1 or 0
    - ➢ **has_emoji**
        - If title contains any emojis
        - 1 or 0
    - ➢ **title_caps_ratio**
        - Ratio of all-uppercase words to total number of words in title
    - ➢ **emoji_count**
        - Total amount of emojis in both title and description
    - ➢ **title_word_count**
        - Number of words in title using split()
    - ➢ **title_char_count**
        - Number of characters in the title
    - ➢ **clickbait_count**
        - Number of clickbait keywords in title
        - The clickbait words are provided by a list in the preprocessor program
    - ➢ **clickbait_ratio**
        - Proportion of clickbait words to total words
    - ➢ **has_clickbait**
        - If title contains clickbait
        - 1 or 0
    - ➢ **des_caps_ratio**
        - Ratio of all-uppercase words to total number of words in description
    - ➢ **des_word_count**
        - Number of words in description using split()
    - ➢ **des_char_count**

    - Number of characters in the description
- ➢ **des_avg_word_len**
    - Average word length in description
- ➢ **channel_word_count**
    - Number of word in channel name
- ➢ **channel_char_count**
    - Number of characters in channel name
- ➢ **channel_freq**
    - How many times channel appears in dataset
    - Identify channel popularity
- ➢ **tags_count**
    - Number of tags used
- ➢ **duration**
    - Total seconds of video
- ➢ **days_since_upload**
    - How long ago video was posted (from point of data collection which was 10/22/2025)
- ➢ **upload_month, upload_day, upload_hour**
    - Time components
- ➢ **weekday_0 to weekday_6**
    - Day of week of upload
    - One-hot Encoding
- ➢ **is_weekend**
    - If uploaded day was from Friday to Sunday (weekday 4 to 6)
- ➢ **category**
    - Label encoded search prompt at which video was found.

Note:

- Since I was testing for views, I did not make an engagement attribute as that uses the views attribute for its calculation. Engagement is only calculated later to show engagement trends based on category.

Examples:

- Before Processing

	video_id	title	description	channel_title	tags	views	likes
2	ko70cExuzZM	Taylor Swift - The Fate of Ophelia (Official	The official music video for "The Fate of Ophelia" Stream/download 'The	Taylor Swift	['Taylor swift', 'taylor', 'swift', 'taylor swift TLOAS', 'TS12',	76642269	1546159
3	6wmDlk-7aMk	YLL LIMANI - LARG (Official Music Video)	YLL LIMANI - LARG (Official Music Video) Subscribe to my YouTube	Yll Limani	['Yll Limani', 'Ylli', 'Ylli Limani', 'ylli limani', 'yll',	17430015	37107
4	sr_qh33LsKQ	Marino - Lust (feat. Alexandria)	Marino, Alexandria - Lust (Official Lyric Video) https://broke.ffm.	Marino	['marino', 'marino music', 'devil in disguise', 'devil in	6876836	106261

duration	upload_date	category
PT3M59S	2025-10-05T23:00:06Z	Music
PT3M3S	2025-07-31T16:00:07Z	Music
PT1M56S	2025-10-14T04:00:07Z	Music

- After Processing

	video_id	views	likes	duration	categ	has_questions	has_exclamations	has_numbers	has_emoji	title_caps_ratio	emoji_count	title_word_count	title_char_count
2	ko70cExuzZM	76642269.0	1546159.0	239	6	0	0	0	0	0.0	0	9	54
3	6wmDlk-7aMk	17430015.0	37107.0	183	6	0	0	0	0	0.4285708163274053	2	6	37
4	sr_qh33LsKQ	6876836.0	106261.0	116	6	0	0	0	0	0.0	0	4	28

clickbait_count	clickbait_ratio	has_clickbait	des_caps_ratio	des_word_count	des_char_count	des_avg_word_len	channel_word_count	channel_char_count	channel_freq	tags_count
0	0.0	0	0.0	61	756	11.114752276270117	2	12	1	9
0	0.0	0	0.06611569701523165	107	1085	8.89719543016865	2	10	1	10
0	0.0	0	0.0	17	259	14.058815259520436	1	6	1	17

days_since_upload	upload_month	upload_day	upload_hour	is_weekend	weekday_0	weekday_1	weekday_2	weekday_3	weekday_4	weekday_5	weekday_6	category_label
16	10	5	23	1	0	0	0	0	0	0	1	Music
82	7	31	16	0	0	0	0	1	0	0	0	Music
7	10	14	4	0	0	1	0	0	0	0	0	Music

Note:

- video_id is removed in the ModelDevelopment.py and category_label is encoded and removed later and is used for matching the numeric category value with the label on the visualizations.

6. Model Development and Evaluation

When training these algorithms, I wanted to make sure the data they both received was unique to themselves. This would help to better compare performance between both models. So using the video_id attribute, I found all the shared videos between both the algorithms and placed them into another dataset to be used for testing. This came to about 291 videos which was close to the testing data amount I would get from using the train split for the API database which had 1532 entries. This is roughly a 0.189 training split, which is close to the 0.2 commonly used for such. I removed the 291 matching videos from both of the databases so when it would be used for testing, the data would be new to the models to better test them.

- Two types of models used
  - XGBoost Regression for regression
  - Logistic Regression for classification

1. Web Scraping Models
   - ➢ View Prediction Model
     - XGBoost Regression

```
xg_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    colsample_bytree=1.0,
    learning_rate=0.01,
    max_depth=5,
    n_estimators=500,
    subsample=0.7,
    random_state=42
)
```

     - Trained to predict the logarithmic view count of videos give its attributes
     - XGBoost was selected because:
       - Can handle complex, mixed-type datasets efficiently
       - Capture nonlinear feature relationships
       - Maintains strong generalization performance

- Input
  - Top 25 Features according to RFECV were input into the model for training
    - 'likes', 'duration', 'category', 'has_numbers', 'has_emoji', 'title_caps_ratio', 'emoji_count', 'title_word_count', 'title_char_count', 'clickbait_count', 'clickbait_ratio', 'des_caps_ratio', 'des_word_count', 'des_char_count', 'des_avg_word_len', 'channel_word_count', 'channel_char_count', 'channel_freq', 'tags_count', 'days_since_upload', 'upload_month', 'weekday_1', 'weekday_3', 'weekday_4', 'weekday_6'

- Training Data Size
  - 2106 Entries

- Performance with Training Data
  - Mean-Squared Error = 0.341
  - R^2 = 0.967

- Performance with Testing Data
  - Mean-Squared Error = 0.813
  - R^2 = 0.841

➢ Category Prediction

- Logistic Regression

```
model = LogisticRegression(
    class_weight='balanced',
    penalty='l2',
    solver='lbfgs',
    C=1000,
    max_iter=2500,
    random_state=42
)
```

  - Trained to classify which category a video is given its attributes
  - Logistic Regression was selected because:

- Well suited for classification tasks with categorical data
- Known to perform effectively
- Generalizes well and avoids overfitting

- Input
  - Top 22 Features according to RFECV were input into the model for training
    - 'views', 'likes', 'duration', 'has_questions', 'has_numbers', 'has_emoji', 'title_caps_ratio', 'emoji_count', 'title_word_count', 'title_char_count', 'clickbait_count', 'clickbait_ratio', 'has_clickbait', 'des_word_count', 'des_char_count', 'des_avg_word_len', 'channel_word_count', 'channel_char_count', 'channel_freq', 'tags_count', 'days_since_upload', 'upload_month'

- Training Data Size
  - 2106 Entries

- Performance with Training Data
  - Accuracy = 0.557

- Performance with Testing Data
  - Accuracy = 0.272

2. Youtube API Models

➢ View Prediction Model

- XGBoost Regression
  - Uses same parameters as above
- Input
  - Top 7 Features according to RFECV were input into the model for training
    - ['likes', 'duration', 'category', 'des_caps_ratio', 'channel_freq', 'days_since_upload', 'weekday_3']

- Training Data Size
  - 1532 Entries

- Performance with Training Data
  - Mean-Squared Error = 0.212
  - R^2 = 0.979

- Performance with Testing Data
  - Mean-Squared Error = 1.01
  - R^2 = 0.803

➢ Category Prediction

- Logistic Regression
  - Uses same parameters as above

- Input
  - Top 29 Features according to RFECV were input into the model for training
    - ['views', 'likes', 'duration', 'has_questions', 'has_exclamations', 'has_numbers', 'has_emoji', 'title_caps_ratio', 'emoji_count', 'title_word_count', 'title_char_count', 'clickbait_count', 'clickbait_ratio', 'has_clickbait', 'des_caps_ratio', 'des_word_count', 'des_char_count', 'des_avg_word_len', 'channel_word_count', 'channel_char_count', 'channel_freq', 'tags_count', 'days_since_upload', 'upload_month', 'upload_hour', 'weekday_1', 'weekday_3', 'weekday_4', 'weekday_6']

- Training Data Size
  - 1532 Entries

- Performance with Training Data
  - Accuracy = 0.615

- Performance with Testing Data
  - Accuracy = 0.302

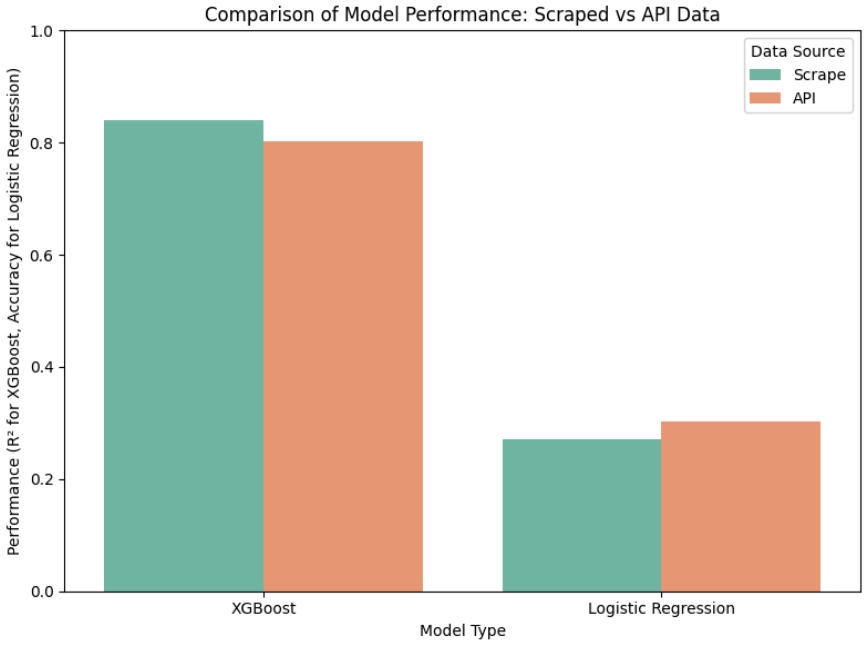
When looking at the performance results for the view count models, the results can give somewhat conflicting results. Scraping with the training data gave an R^2 = 0.967 while API gave an R^2 = 0.979, showing the API performed slightly better when both are using their own respective data. But when we look at the testing data, which are videos neither have seen, web scraping gave an R^2 = 0.841 while API gave an R^2 = 0.803, showing that the web scraping data provided a better model for unseen data.

For the category prediction models, API came out on top with both performance results. API had an accuracy of 0.302 for the testing data and an accuracy of 0.615 on its training data while web scraping had an accuracy of 0.272 for the testing data and an accuracy of 0.557 for its training data. This means that API provides a better model for predicting the category of a video.

- Feature Importance
    - The feature importance technique used in the models was RFECV or Recursive Feature Elimination with Cross-Validation. This method automatically picks the most important features for a model while avoiding overfitting. It starts by training the provided model with all features and measuring its performance, it will then remove the least important feature (which is scored by the model) and repeat this step until only one feature remains. It will then choose the set of features that give the best cross-validation score and use those for the model.

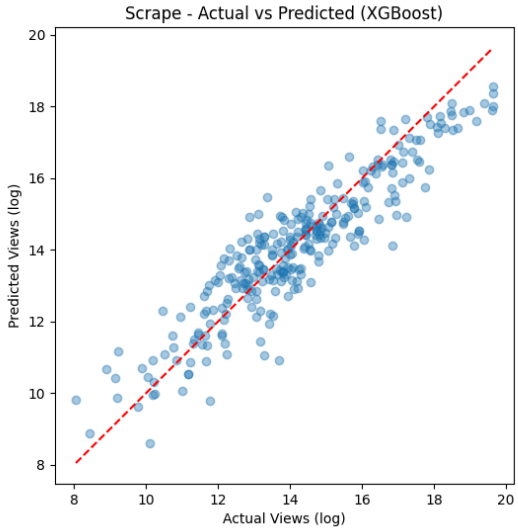
7. Visualization

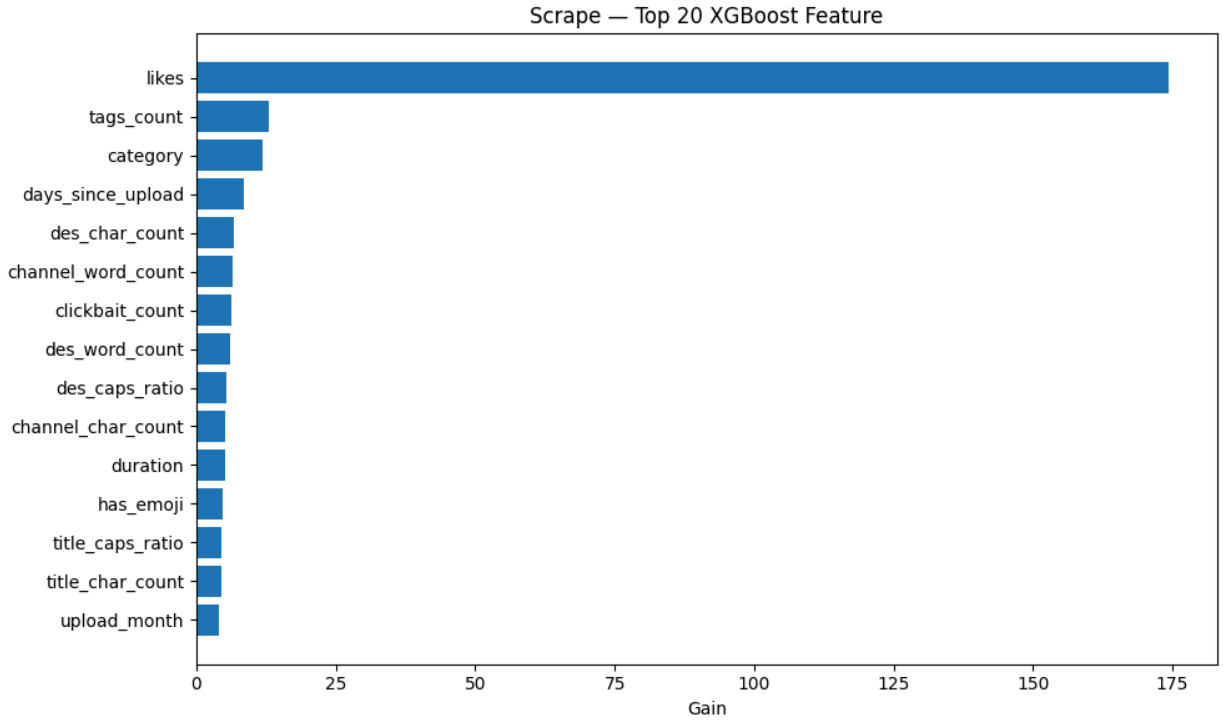
1. Comparing Models

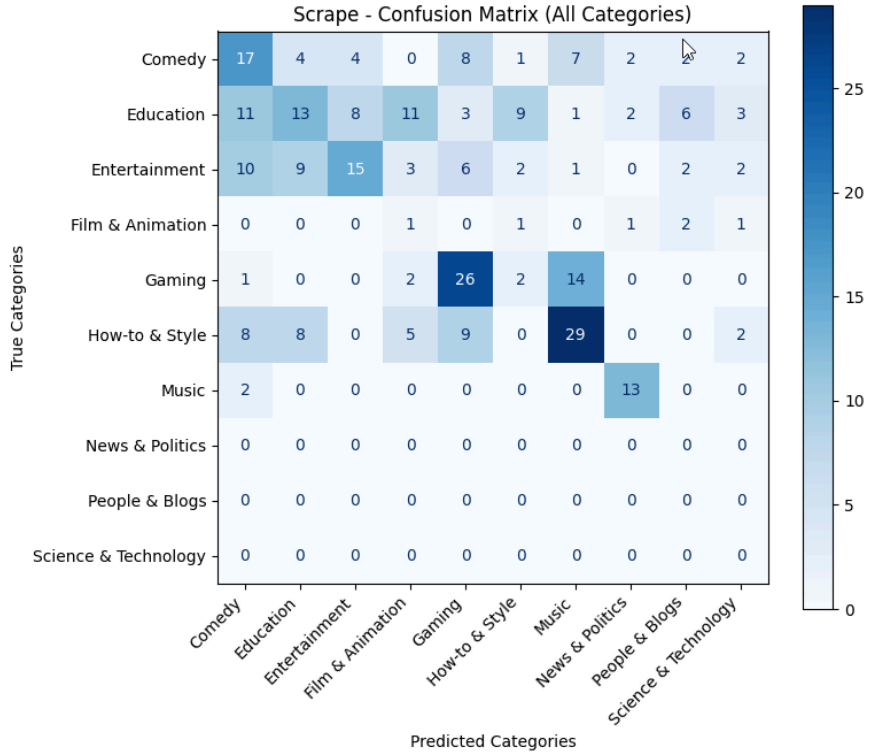


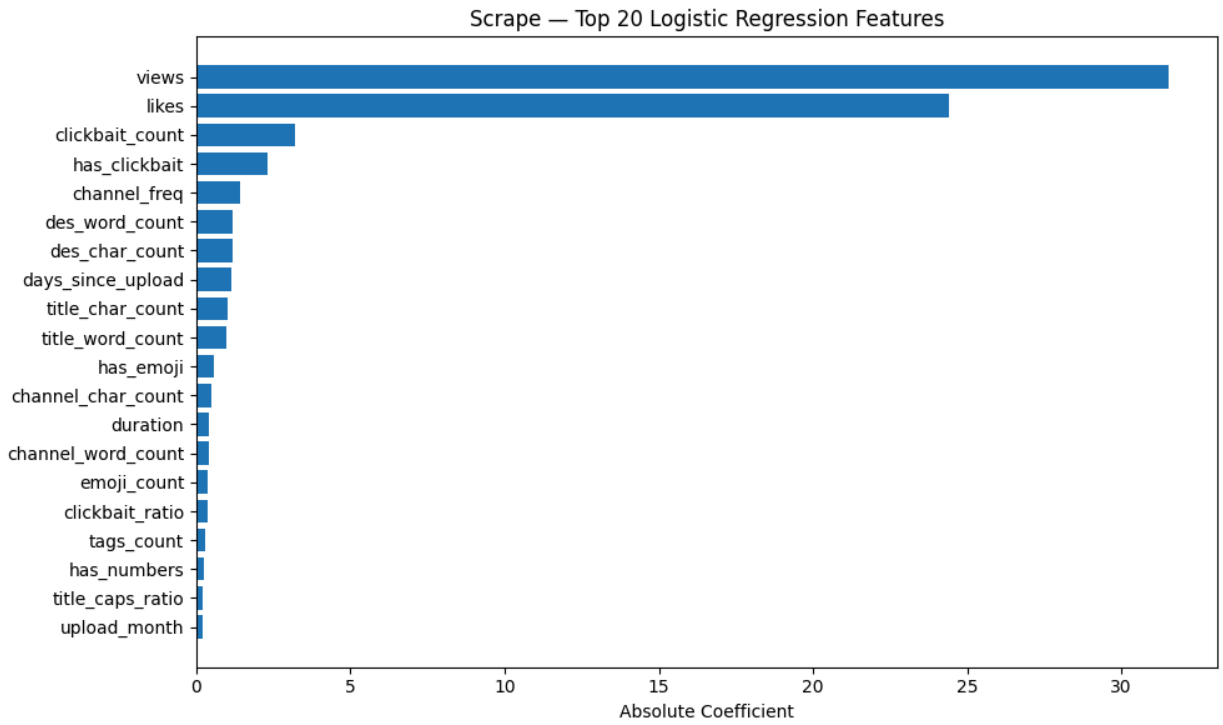
From this we can see that the performances of the data sources differ depending on the type of model used. API data performs better for classification, while web scraping data performs better for regression. These performance differences however are very minimal.

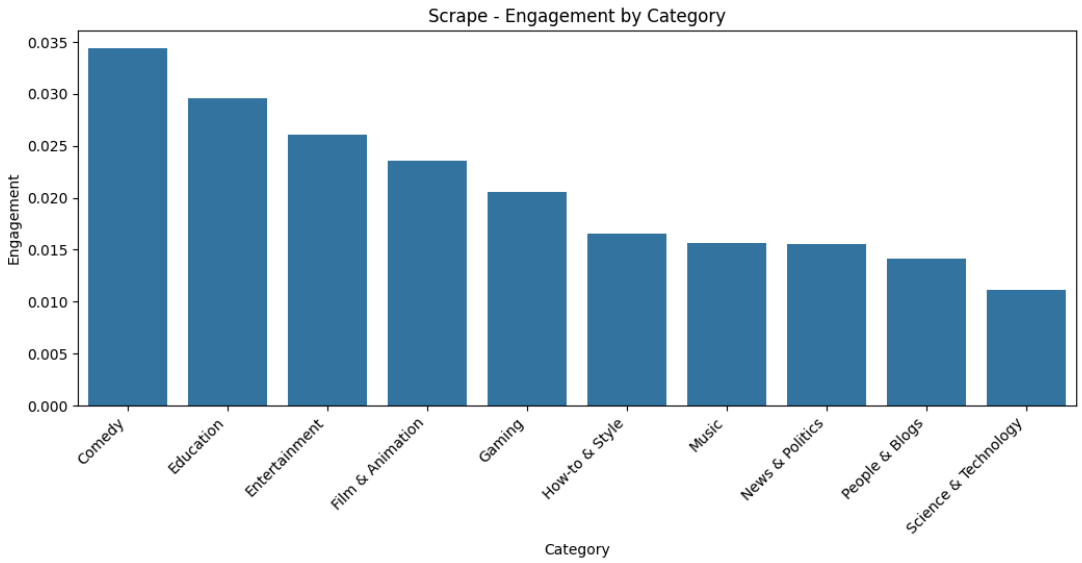
2. Scrape



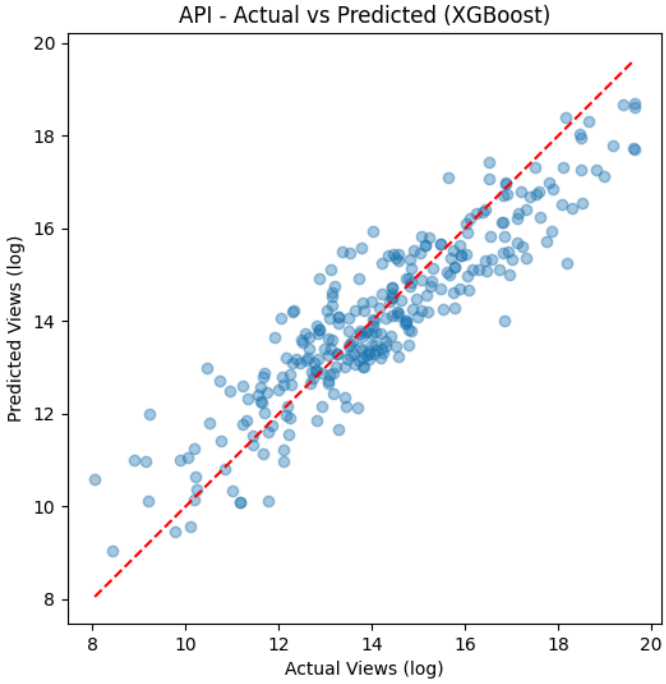

Scrape — Top 20 XGBoost Feature
likes
tags_count
category
days_since_upload
des_char_count
channel_word_count
clickbait_count
des_word_count
des_caps_ratio
channel_char_count
duration
has_emoji
title_caps_ratio
title_char_count
upload_month
0
25
50
75
100
125
150
175
Gain

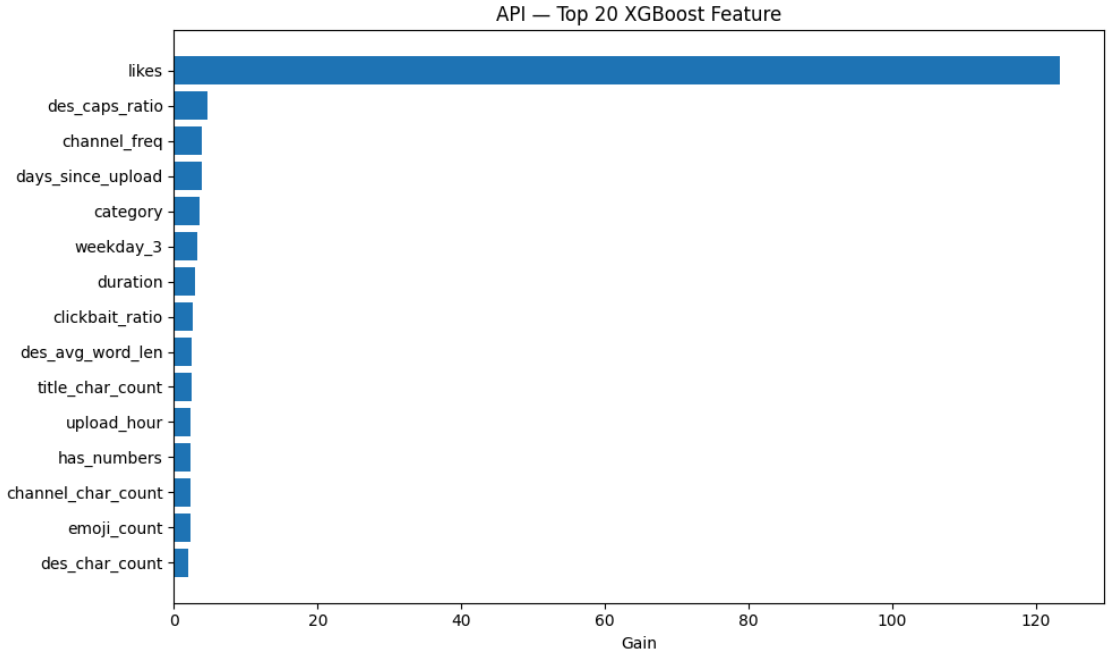

Scrape - Confusion Matrix (All Categories)
Comedy 17 4 4 0 8 1 7 2 2 2
Education 11 13 8 11 3 9 1 2 6 3
Entertainment 10 9 15 3 6 2 1 0 2 2
Film & Animation 0 0 0 1 0 1 0 1 2 1
Gaming 1 0 0 2 26 2 14 0 0 0
How-to & Style 8 8 0 5 9 0 29 0 0 2
Music 2 0 0 0 0 0 0 13 0 0
News & Politics 0 0 0 0 0 0 0 0 0 0
People & Blogs 0 0 0 0 0 0 0 0 0 0
Science & Technology 0 0 0 0 0 0 0 0 0 0
True Categories
Comedy
Education
Entertainment
Film & Animation
Gaming
How-to & Style
Music
News & Politics
People & Blogs
Science & Technology
Predicted Categories
25
20
15
10
5
0

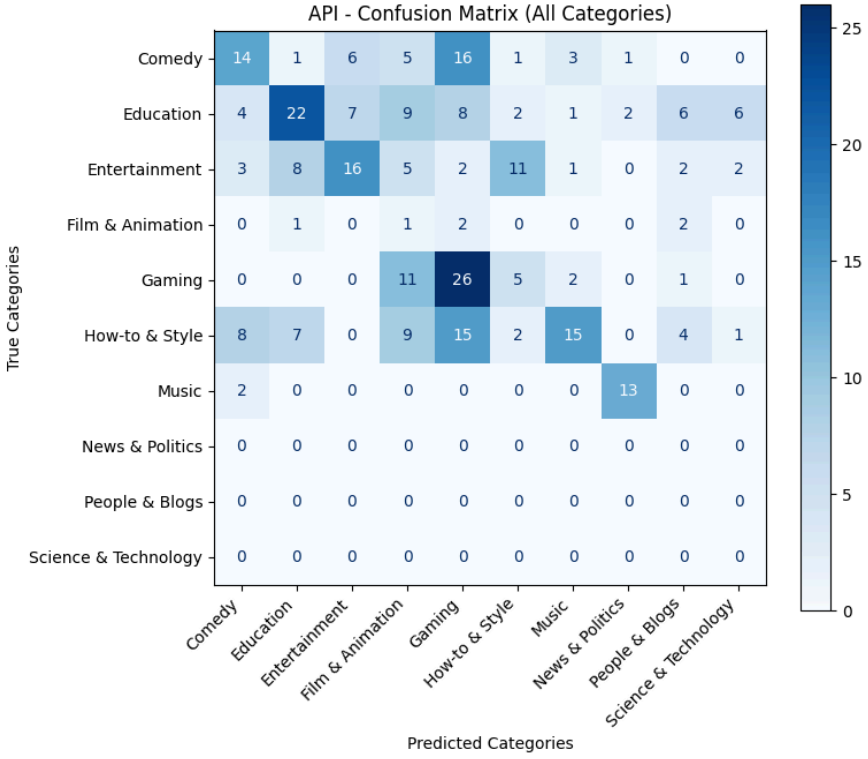

Scrape — Top 20 Logistic Regression Features
views
likes
clickbait_count
has_clickbait
channel_freq
des_word_count
des_char_count
days_since_upload
title_char_count
title_word_count
has_emoji
channel_char_count
duration
channel_word_count
emoji_count
clickbait_ratio
tags_count
has_numbers
title_caps_ratio
upload_month
0
5
10
15
20
25
30
Absolute Coefficient

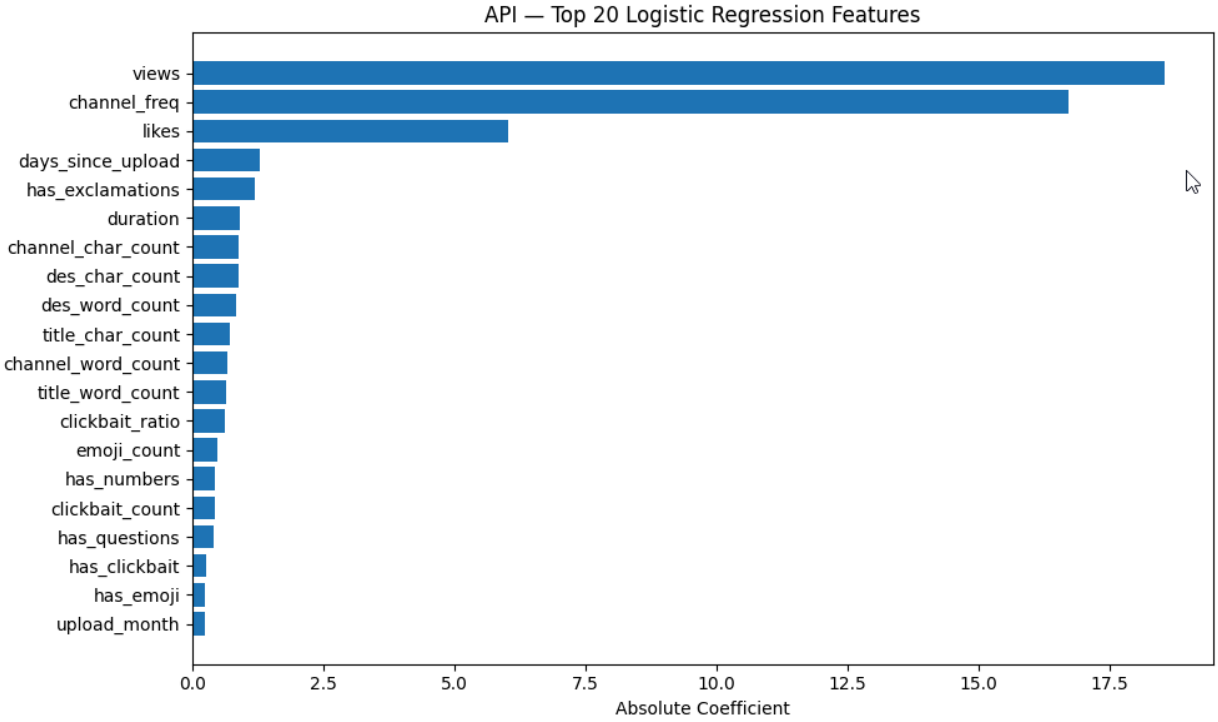

Scrape - Engagement by Category
Engagement
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Comedy
Education
Entertainment
Film & Animation
Gaming
How-to & Style
Music
News & Politics
People & Blogs
Science & Technology
Category

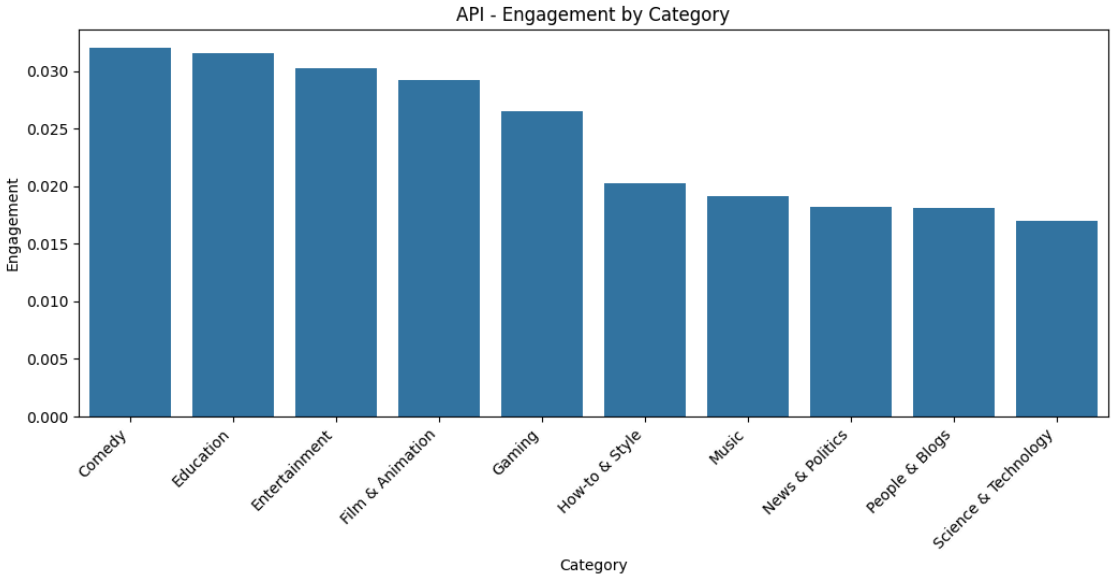
- API


API - Actual vs Predicted (XGBoost)
Predicted Views (log)
Actual Views (log)
8
10
12
14
16
18
20


API — Top 20 XGBoost Feature
likes
des_caps_ratio
channel_freq
days_since_upload
category
weekday_3
duration
clickbait_ratio
des_avg_word_len
title_char_count
upload_hour
has_numbers
channel_char_count
emoji_count
des_char_count
0
20
40
60
80
100
120
Gain


API - Confusion Matrix (All Categories)
True Categories
Predicted Categories
Comedy
Education
Entertainment
Film & Animation
Gaming
How-to & Style
Music
News & Politics
People & Blogs
Science & Technology
14 1 6 5 16 1 3 1 0 0
4 22 7 9 8 2 1 2 6 6
3 8 16 5 2 11 1 0 2 2
0 1 0 1 2 0 0 0 2 0
0 0 0 11 26 5 2 0 1 0
8 7 0 9 15 2 15 0 4 1
2 0 0 0 0 0 0 13 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
25
20
15
10
5
0


API — Top 20 Logistic Regression Features
views
channel_freq
likes
days_since_upload
has_exclamations
duration
channel_char_count
des_char_count
des_word_count
title_char_count
channel_word_count
title_word_count
clickbait_ratio
emoji_count
has_numbers
clickbait_count
has_questions
has_clickbait
has_emoji
upload_month
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Absolute Coefficient



8. Discussion and Conclusions

From this project, it became clear both model types trained on scraped vs API data provided different advantages and disadvantages, depending on the data it is being trained with. For regression models like XGBoost, using scraped data provided more of an accurate model for predicting view count. For classification models, using API data proved to be a better choice for predicting categories, providing a higher accuracy score. The performance gains between both of these models however was very minimal and could be a result of slight variations in the search results for the input terms.

The logistical model was very interesting to see the results of because it did not perform as well as I thought it would. This I believe is due to a need for more data or to improve on the feature engineering inside the API dataset. Looking at the confusion matrix, we can see that it was only decent at identifying Gaming and Entertainment videos. Gaming videos however were often confused as Comedy and How-To & Style videos which is interesting to see. Comedy and Entertainment were identified correctly most of the time but had many errors. From the matrix we can see that News & Politics, People & Blogs, and Science & Technology videos had been completely eliminated from the testing data. This may be due to many factors but I believe that it is the result of a lot of the collected data being dropped.

There were many challenges faced in this project, particularly in the web scraping of youtube. Youtube's front-end heavily relies on java-script, which is used for things like dynamic features and interfaces. This means that certain things like comments needed to be scrolled to for them to load and even then was often unreliable when collecting. This led me to later drop the comments as more videos failed to collect the amount of comments and did. This was not an issue for the API as that value can be directly grabbed on-demand. This loading of things like comments and search results also made the webscrapping of youtube incredibly slow. Often taking up to 5 seconds per video and for 3000 videos that time adds up. This is where I spent the most time of the project besides model fine tuning. I also found that web scraping often collected shorts instead of videos so while my scraper would tell me I collected 1000 videos for a given search, only about 200 of them would be normal videos instead of shorts. This is not a problem for the API as the type of media you would want to collect could be specified in its search.

Another challenge I faced was dealing with Google's API limitations. While this wasn't as frustrating as the webscraping, YouTube's API limited a given API key to 10,000 requests a day. This meant a small mistake or having to recollect data, could mean having to either wait an entire day or generating a new api key from google to continue to collect results. In a business setting where the API isn't running on the free version, this would not be a problem but in my case, I did have to generate another API key to continue data collection more than once.

Some ethical and legal considerations of the data collection is that not everyone wants their videos collected and used without their permission, especially if they are used for monetary gain. This is especially true in generative models, like a video generator. It's essentially taking their content and teaching a machine to use that to make its "own" content without their permission which to many can be seen as unethical and depending on how machine learning and AI are treated in the future, could even spell legal trouble. Web scraping can also bring legal trouble as companies don't always like when information is taken from their websites without permission. Google having the YouTube API, especially with less limitations, behind a paywall could mean that they would rather you use their service than scrape their site for free. These are things that must be kept in mind when developing models and collecting data.

In retrospect there are many things I would change with this project. I do feel that I would try and collect more than 3000 videos, especially before preprocessing as for stuff like the API model, I ended up having to drop about half of the collected videos. I would also try and collect more videos for the existing searches/categories to provide more for the model to train off of in hopes to get better performance. I would also try to properly implement comment scraping as well as getting more features like channel descriptions or related videos. I would even try to incorporate thumbnail data and analysis for things like saturation and complexity. In a world that is overly saturated like

today's, bright colors and crazy visuals do tend to grab peoples eye. I did still learn a lot from this project and plan to take these lessons with me into the workspace.