YouTube Popularity Scraper and Engagement Prediction

By: Gabriel Silva

Github: https://github.com/MrYo10/YouTube_Popularity_Scraper/tree/main

Description of the Project:

The project aimed to predict Youtube video stats which included popularity and engagement using two data sources.

- Scraped Data Which collected straight from the YouTube webpages such as the titles, descriptions, tags, category, etc.

- API data which collected data using the Youtube Data API v3 which are the official statistics such as the view count, likes, comment count, and the channels metrics.

The workflow consisted of the data collection, cleaning & preprocessing, feature engineering, and the machine learning and training and the evaluation for both data sets. The Random Forest Regressor and the optional XGBoost models were trained to predict either the engagement rates which are the (Likes + comments) / views from the api data or the View count from the scrapped data.

How to use:

- Install Dependecies: pip install -r requirements.txt

- Collect data:
  - python src/collect_api.py --resume
  - python src/collect_scraped.py --resume

- Preprocess: python src/preprocess.py

- Feature extraction: python src/features.py

- Train Models:
  - python src/train_api.py   # API-based
  - python src/train_scraped.py   # Scraped-based

- Evaluate and Visualize: python src/eval_visualize.py

The data will appear under:

Data/ interim/ , data/ processed – for the clean and feature data

Models/ - trained models + metrics

Reports/ - plots and comparison firgures

Training and inferencing:

Each Training script splits the data in the standard 80% train and the 20% test and it fits to a Random Forest Regressor:

Model 1: Is trained on scrapped features to predict the log1p(viewCount)

Model 2: is trained on API features to predict engagement_rate

The prediction can be done by loading the saved .joblib mode and calling. predict(new_features)

Data Collection:

Scraped collector: collect_scraped using BeautifulSoup + requests to extract title, description, tags, uploader, catagory, publish date, and view counts from video URLs

API collector: collect_api.py calls the Youtube Data API v3 using videos IDs to fetch viewCount, like count, comment count, channel subscribers, and tags

Tools used: Python 3.12, pands, Numpy, BeautifulSoup4, requests, scikit-learn, xgboost, matplotlib.

Collected attributes:

Source	Attributes
Scraped	video_id, url, title, description, uploader, publish_date, category, tags, views (when available)

API	video_id, url, title, description, uploader, publish_date, category, tags, views (when available)

Number of samples:

Scraped: 10,199 rows

API = 2,719 rows

Data Preprocessing:

Data cleaning:

Handled bad CSV lines and missing colums

Converted numeric fields like view counts and like count to floats

Parsed ISO 8601 public dates

Removed duplicates by video_id

Replaced invalid / zero view counts with NaN

Clipped the videos with extreme views to 99.9th percentile to limit the outliers.

Feature Engineering:

Created new colums for:

Duration_min

time_since_upload_days

tags_count, desc_length, had_links_in_desc

Upload_year, upload_month, upload_dow, upload_hour

Target Variables:

Engagement_rate = (likes + comments)/views

Target_views_log = log1p(viewcount)

Model Development and Evaluation

Train / test Partition

Both the data sets are split 80/20 randomly

Model 1- Scraped Data

Algorithm: Random Forest Regressor

Input features: title n-grams, duration_min, upload time features, text statistics.

Target: target_views_log

Train size = 3500 samples with valid views

Performance

train R^2 = 0.00

Test R^2 = 0.00

MAE = 2.5 log scale -> = x12 error factor in raw views

Conclusion: The scraped text alone is weakly predictive; channel-level features are missing.

Model 2: API data

Algorithm: Random Forest Regressor (+ optional XGBoost)

Input features: duration_min, time_since_upload_days, channel stats, title, desctripton features.

Target: target_engagement

Train size: about 2700 rows

Performance:

Train R^2 = 0.91 MAE – 0.004 RMSE =0.007

test R^2 = 0.41 MAE = 0.0108 RMSE = 0.0159

This explains the 41% of variance in engagement, which is a strong result for YouTube Behavior data.

Feature Importance

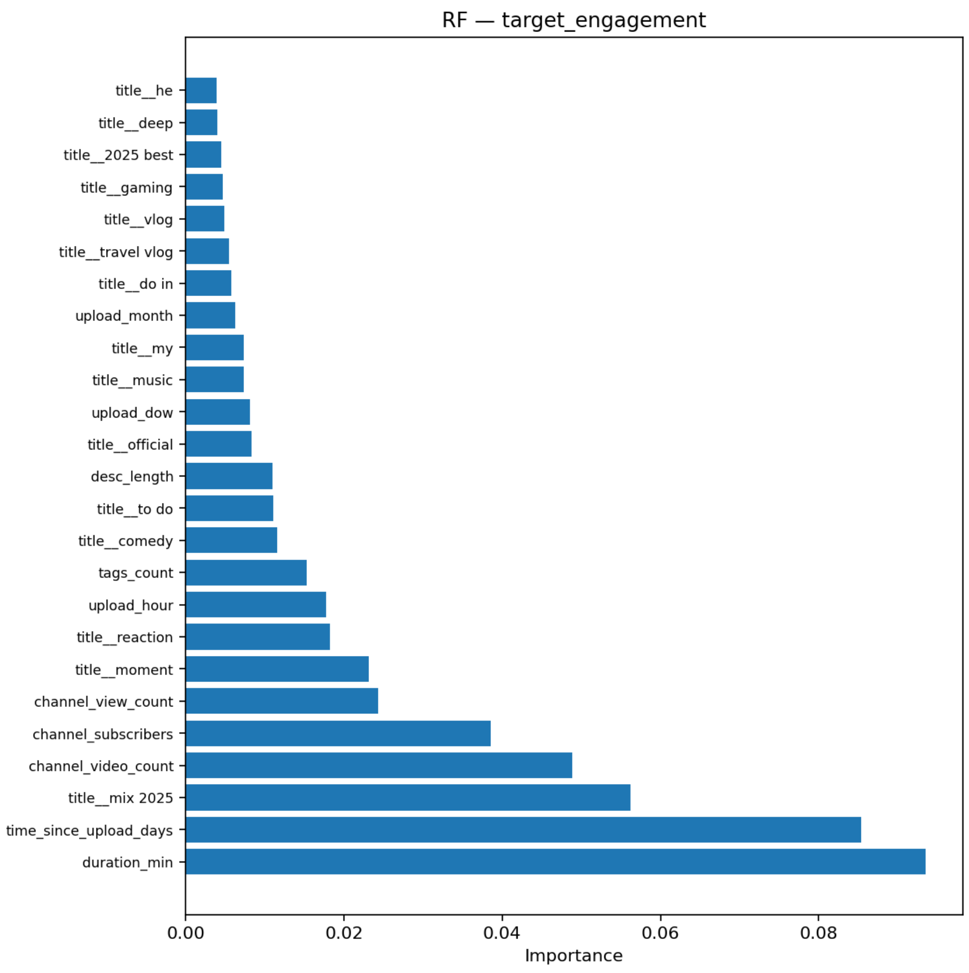
Techniqie: Mean Decrease in impurity (Random Forest feature importances)

Top predictors for API model:

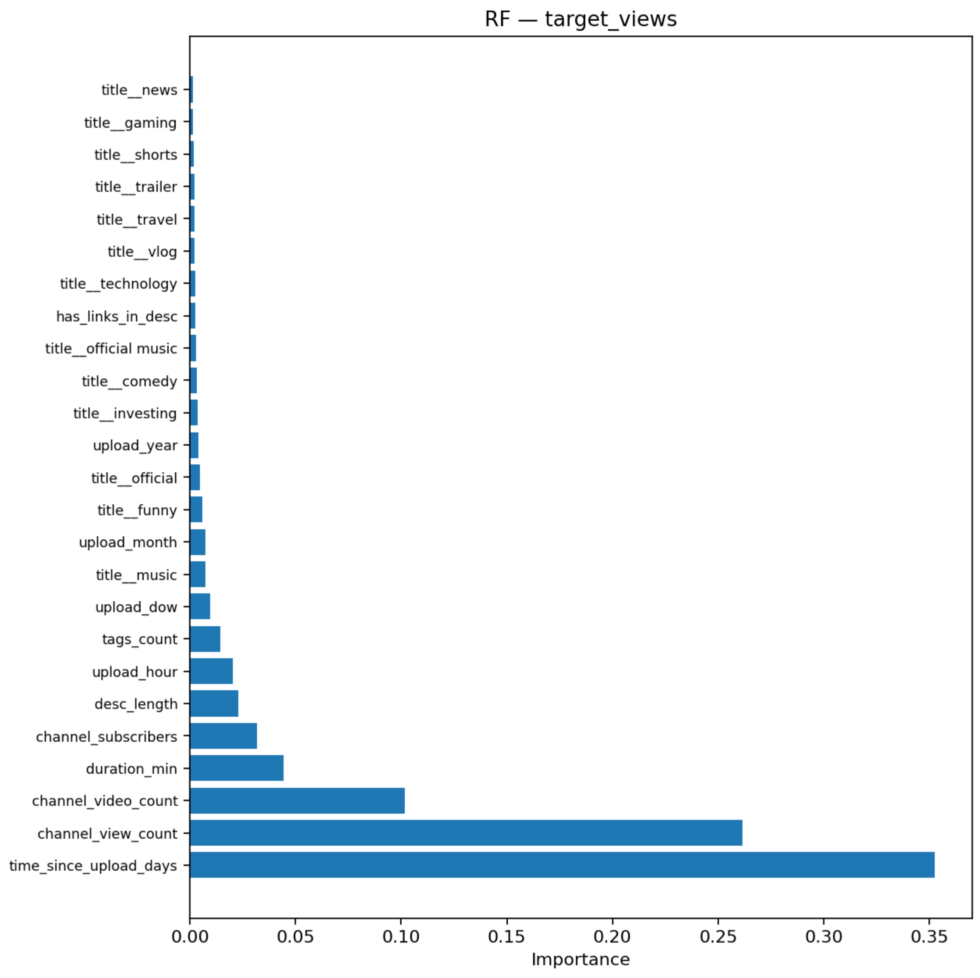
Channel_subscribers, viewCount_clipped, time_since_last_Upload_days, desc_length, tags_count, and the specific title tokens like official, remix and or live.

Feature importance plots saved as reports/fi_api.png

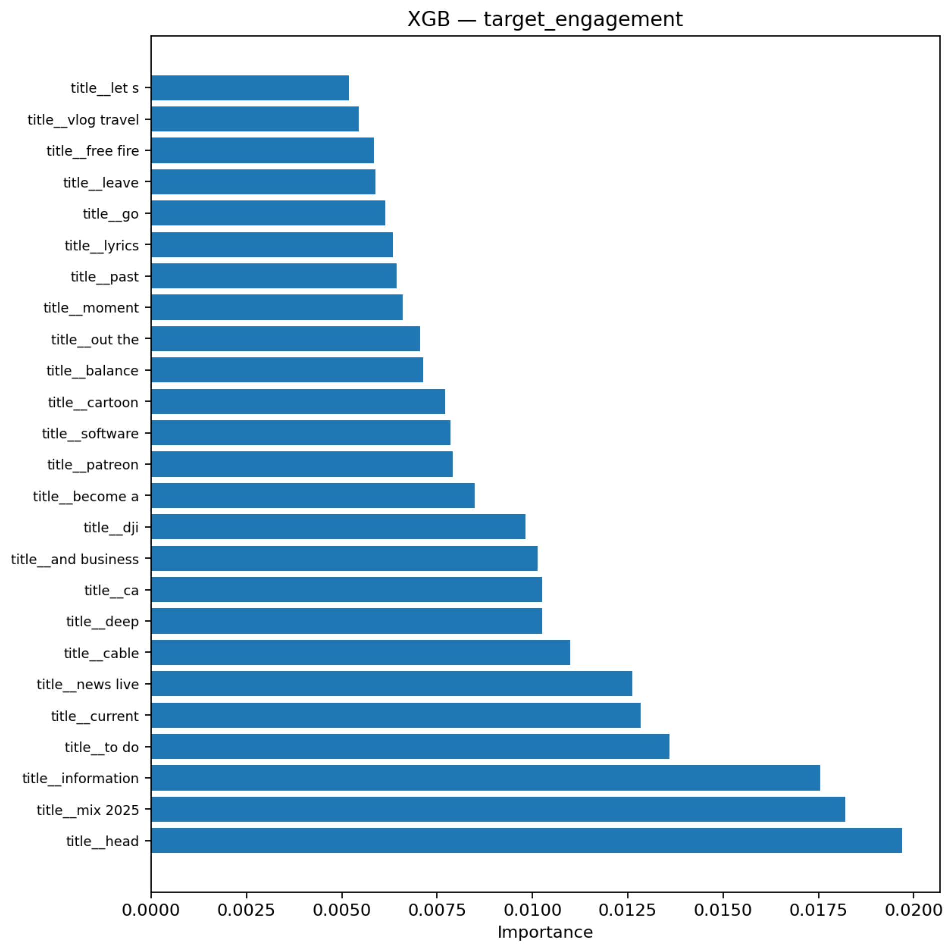
Random Forest Feature Importance:



Random Forest feature importance for View Prediction:



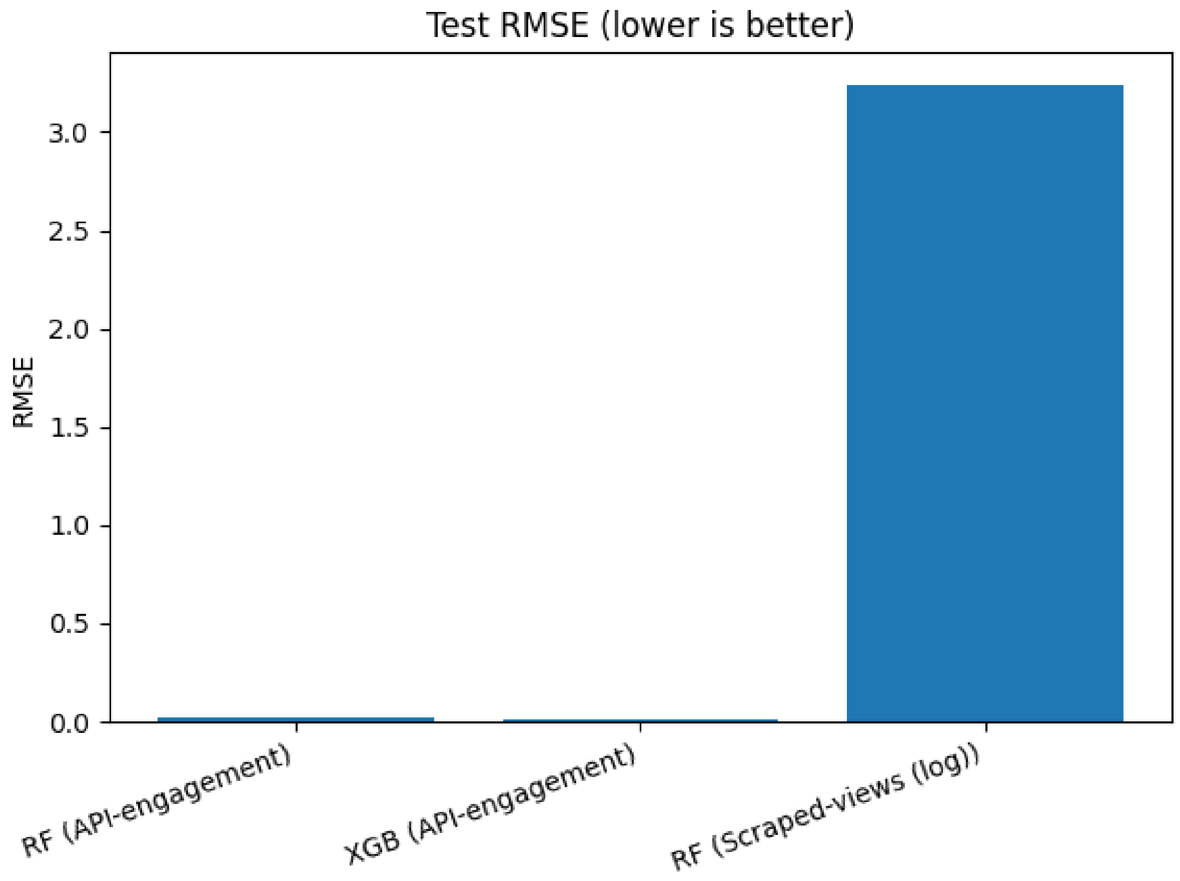
XGBoost Feature importance for Engagement Rate



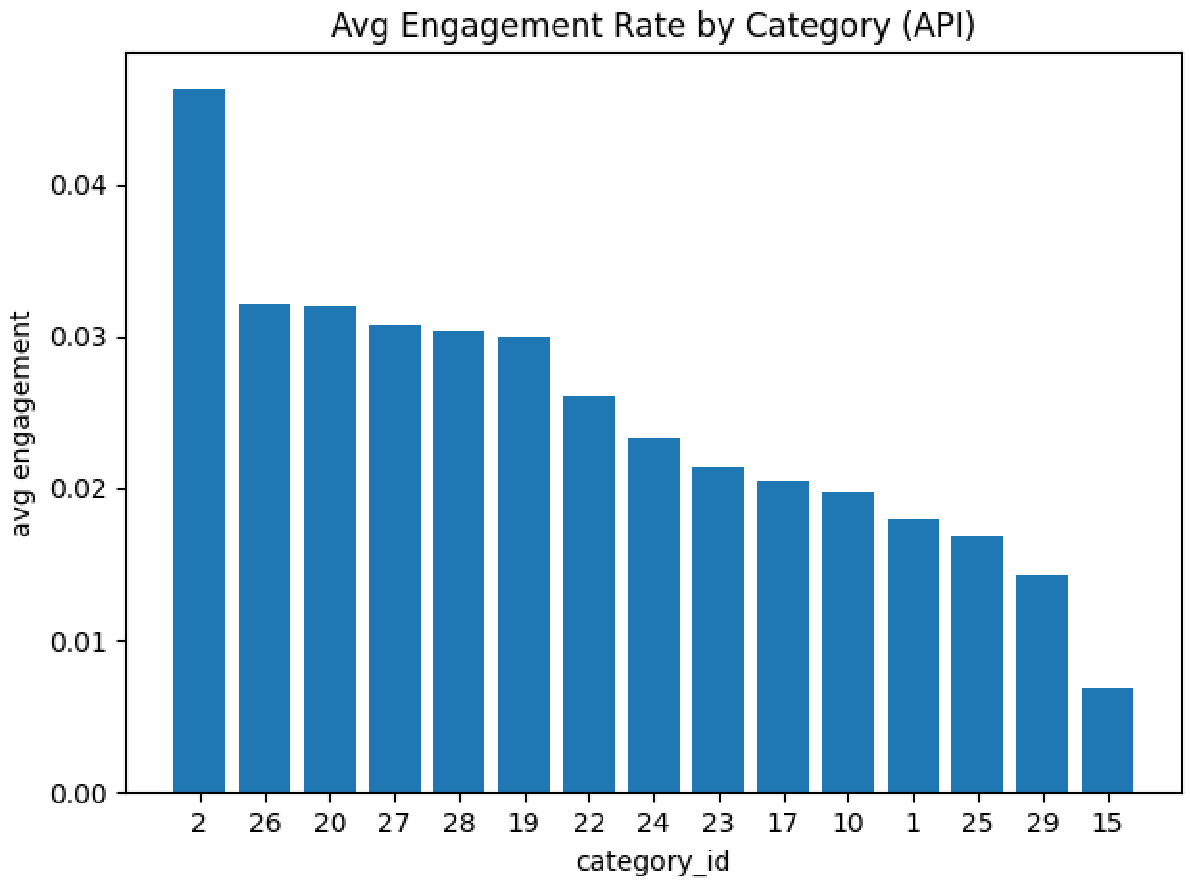
Visualization:

Test RMSE comparison:

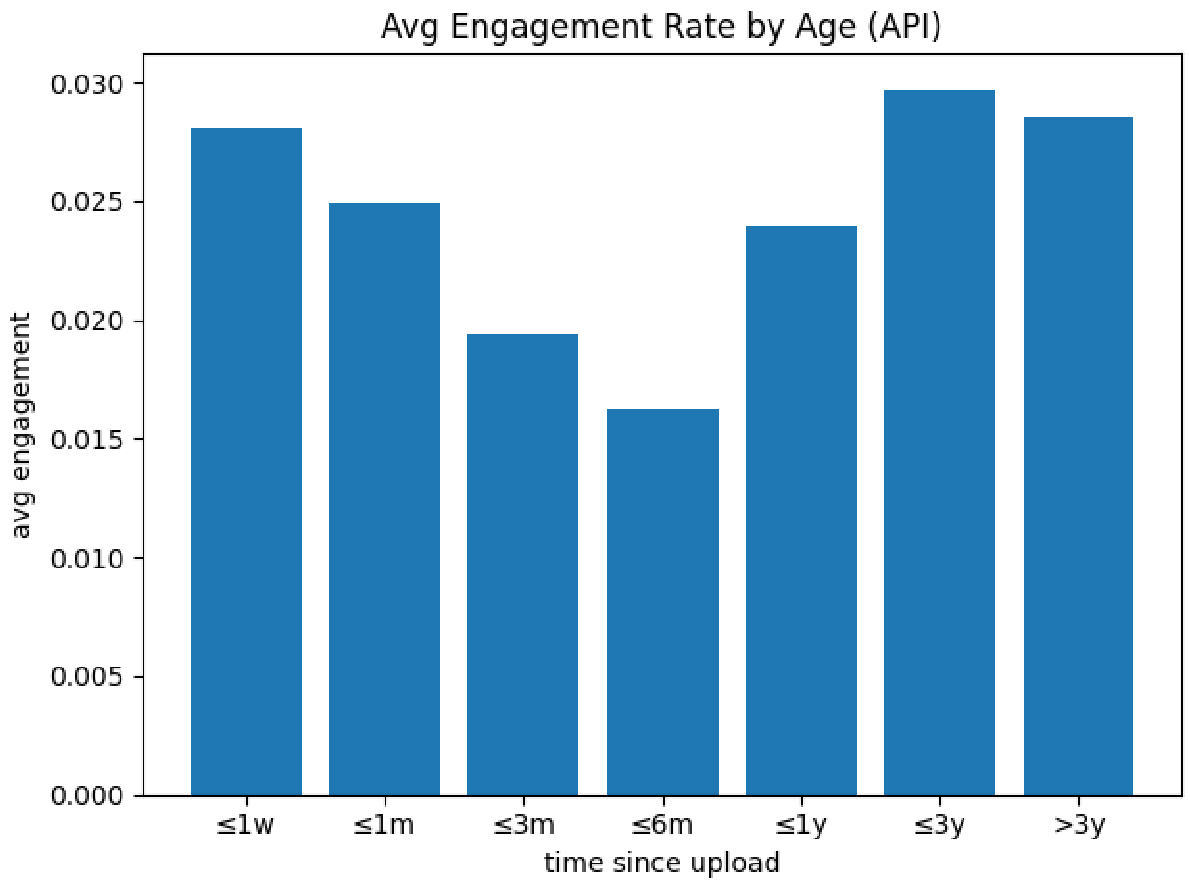
Comparison Test of RMSE between API and Scraped data models:


Test RMSE (lower is better)
RMSE
3.0
2.5
2.0
1.5
1.0
0.5
0.0
RF (API-engagement)
XGB (API-engagement)
RF (Scraped-views (log))

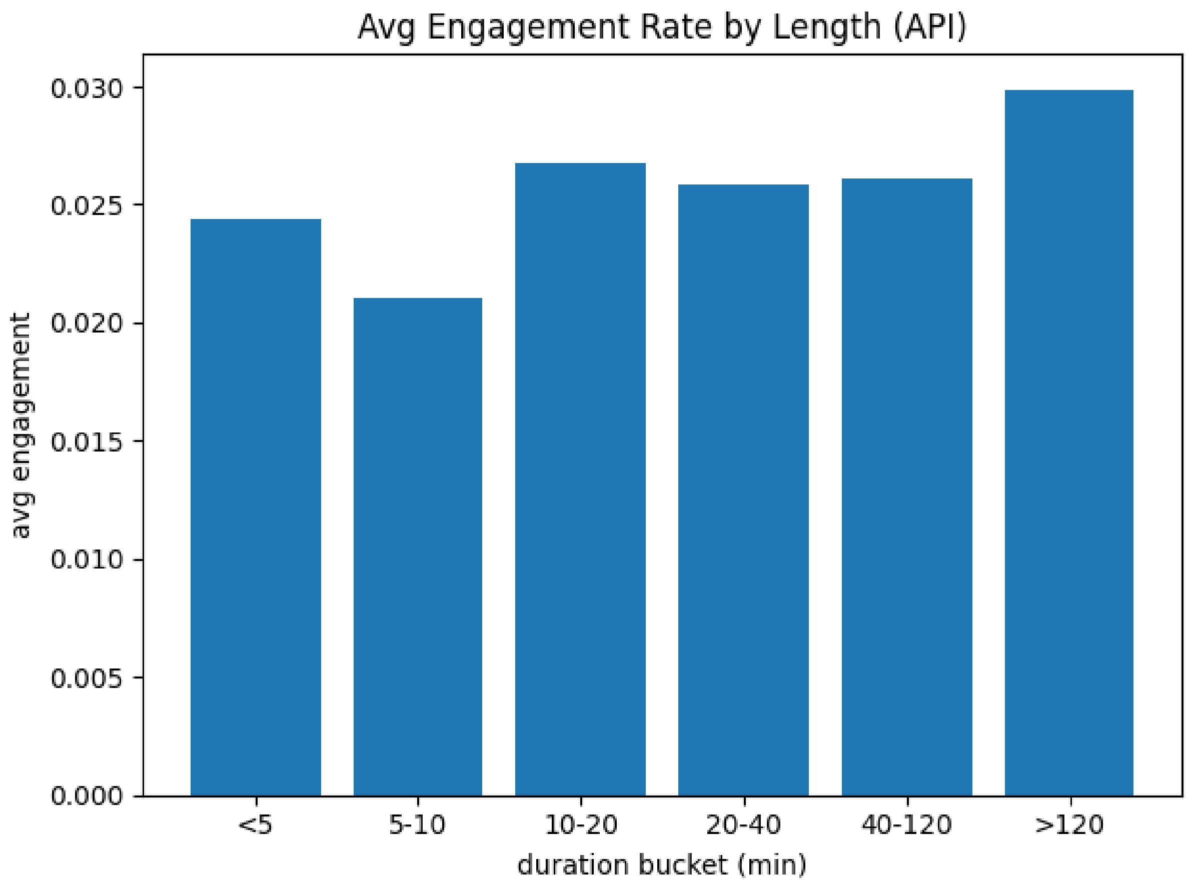
By Category:



By age:


Avg Engagement Rate by Age (API)
avg engagement
0.030
0.025
0.020
0.015
0.010
0.005
0.000
≤1w
≤1m
≤3m
≤6m
≤1y
≤3y
>3y
time since upload

By Length:



Discussions and Conclusions:

The results of the project show the diffrences between the predictive power of the two data sources. The API model outperformed the scraped model in predicting engagement rates by being able to achieve the R^2 of about 0.41 on test data compared to the scraped models near zero correlation when predicting view counts. The outcome tells me that the structured and reliable numerical features of YouTube's API, such as the subscriber count total channel views and how recent the video was uploaded, carries harder when being able to predict values than just textual information. In contrast, the scraped data set, while larger in size, lacked accurate numerical engagement indicators. It mainly relied on text-based metadata such as the titles descriptions and tags which are highly variable across every channel and genre. Because of that, the scraped model struggled to be able to identify consistent patterns in finding viewer behavior.

Several Patterns emerged during the feature of importance and trend analysis. The first videos from music and gaming categorized tended to have the highest engagement, which showed their strong fan communities and frequent sharing behaviors. Secondly, engagement peaks for videos with 5 to 10 minutes in length, which makes it long enough to be valuable to the viewer but short enough to maintain their attention. Third, the recent uploads consistently shower higher engagement, which showed that the freshness and YouTube 's algo's promotion of new content had a high impact.

I encountered a couple of challenges with this project, being technical and data related. The scraped dataset has irregular formatting, broken lines, and missing columns which require more robust preprocessing and error handling. HTML structure changes on YouTube also made the scraping less consistent, often making malformed records. Another issue I faced was the difference in view counts as some videos had millions of views while other videos had only a couple hundred. This made the training model unstable which out log transformation or clipping. The API dataset was a lot cleaner, but it was limited to googles' quota restrictions, which prevented collection of large-scale data in a small amount of time. Finally, since engagement and popularity depend on unobservable factors like thumbnail or algorithm boots, i feel as if even the best model could only partially capture underlying problems.