YouTube Video Popularity Prediction & Engagement Analysis

1. Description of the project

The objective of this project is to predict the popularity (measured by view count) of YouTube videos and identify key factors influencing engagement. To achieve this, we built two separate machine learning models:

- Model 1: Trained on a limited dataset collected via web scraping (Selenium).
- Model 2: Trained on a rich, structured dataset collected via the official YouTube Data API v3.

The primary goal is to compare the performance of these two models to determine if the extra effort of using the official API (which provides data like tags, duration, and like/comment counts) results in a significantly more accurate model than one built on sparse, publicly scraped data. The target variable for prediction is log(view_count + 1).

2. How to use

Training

The entire pipeline can be executed by running the Python scripts sequentially from a terminal (after installing requirements and setting the API key).

Setup:

1. Install all required packages: pip install -r requirements.txt
2. Edit collectors/collector_api.py to add your YOUTUBE_API_KEY.

Run Pipeline (Windows):

python collectors/collector_api.py

python collectors/collector_scrape.py

python preprocessing/preprocess.py

python preprocessing/feature_engineer.py

python models/train.py

python models/evaluate.py

python viz/visualize.py

This populates the data/, models/, and viz/ directories with all results.

Inferencing

To predict the view count for a new video (using the API model):

1. Load the trained model and feature columns:

   import joblib

   import pandas as pd

   model = joblib.load('models/xgb_api_model.joblib')

   # Load feature columns from X_api_features.parquet

   X_cols = pd.read_parquet('data/processed/X_api_features.parquet').columns

new_data must be a DataFrame with these columns

2. Collect data for the new video and preprocess it using the *exact same* steps from preprocessing/feature_engineer.py.
3. Ensure the final DataFrame has the same columns in the same order as X_cols.
4. Predict: log_views_pred = model.predict(new_data_df)
5. Convert back from log scale: views_pred = np.expm1(log_views_pred)

3. Data Collection

Used tools

- API Data: google-api-python-client library in Python.
- Scraped Data: Selenium library in Python, controlling a headless Chrome browser.

Collected attributes

- API Dataset: video_id, title, description, tags, category_id, publish_date, duration, view_count, like_count, comment_count, channel_id, channel_title
- Scraped Dataset: video_id, title, channel_title, meta (a single string, e.g., "1.2M views • 3 years ago")

Number of data samples

- API Data: 1172 videos (collection was stopped by the daily API quota).

- Scraped Data: 608 videos.
- Comparison: The models were trained on their respective full datasets (1172 API records vs. 608 Scraped records).

API usage

The YouTube Data API v3 was used. The search.list endpoint (100 quota units) was called to find video IDs for various queries (e.g., "music", "gaming"). The videos.list endpoint (1 quota unit) was then used to fetch detailed statistics and snippets for those IDs. Collection halted after exhausting the default 10,000 unit daily quota.

2 Sample data after preprocessing

This shows the final data *just before* feature engineering.

API Sample (api_preprocessed.parquet):


```
[
 {
   "video_id": "v-a-E-mK8-0",
   "title": "A 20-Minute HIIT Workout",
   "tags": ["hiit", "workout", "fitness"],
   "category_id": "17",
   "publish_date": "2024-05-01T12:00:00Z",
```

```
  "duration": "PT20M15S",
  "view_count": 125034,
  "like_count": 4502,
  "comment_count": 312,
  "channel_title": "Fitness Channel",
  "duration_s": 1215.0,
  "duration_min": 20.25,
  "time_since_publish_days": 176,
  "title_len": 24,
  "title_word_count": 4,
  "num_tags": 3,
  "engagement_rate": 0.038499
 }
]
```


Scraped Sample (scraped_preprocessed.parquet):

```
[
 {
  "video_id": "G-S9mtY-BwY",
  "title": "Funniest Cat Videos of the Week",
  "channel_title": "Cat World",
  "view_count": 2100000.0,
  "time_since_publish_days": 180,
  "title_len": 30,
  "title_word_count": 6
 }
]
```


4. Data Preprocessing

Data cleaning

- API Data: duration (ISO 8601 string) was parsed into total seconds (duration_s).

  publish_date (string) was converted to a datetime object.

- Scraped Data: The meta column was parsed using regular expressions to extract view_count (e.g., "2.1M" -> 2100000) and the age of the video (e.g., "6 months ago" -> 180 days).

Feature Engineering

The target variable for both models was log_views = log(view_count + 1) to handle the wide distribution of views.

API Model Features:

- duration_min: Video length in minutes.
- time_since_publish_days: Days since the video was published.
- title_len: Number of characters in the title.
- title_word_count: Number of words in the title.
- num_tags: Count of video tags.
- has_4k: (Boolean) 1 if "4K" was in the title, 0 otherwise.
- has_tutorial: (Boolean) 1 if "tutorial", "how to", or "course" was in the title.
- cat_...: 11 dummy columns from one-hot encoding the top 10 category_ids (e.g., cat_10 for Music, cat_28 for Sci/Tech, cat_other).
- Total Features: 18

Scraped Model Features:

- time_since_publish_days: Days since the video was published.
- title_len: Number of characters in the title.
- title_word_count: Number of words in the title.
- has_4k: (Boolean) 1 if "4K" was in the title, 0 otherwise.
- has_tutorial: (Boolean) 1 if "tutorial", "how to", or "course" was in the title.
- Total Features: 5

5. Model Development and Evaluation

Train and Test data partition

Both datasets were split using an 80% Train / 20% Test partition with random_state=42 for reproducible results.

Model-1 based on scrapped data

- Machine learning model: RandomForestRegressor and XGBRegressor
- Input to model: X_scraped_features.parquet
- Size of train data: 486 records (608 * 0.8)
- Attributes (5): title_len, title_word_count, time_since_publish_days, has_4k, has_tutorial
- Performance with training data:
- Performance with test data (Test R^2):

Model-2 based on API usage

- Machine learning model: RandomForestRegressor and XGBRegressor
- Input to model: X_api_features.parquet
- Size of train data: 937 records (1172 * 0.8)
- Attributes (18): duration_min, title_len, num_tags, time_since_publish_days, cat_10, cat_17, cat_20, cat_22, cat_23, cat_24, cat_25, cat_26, cat_27, cat_28, cat_other, etc.
- Performance with training data (Train R^2):
    - RF: 0.8953
    - XGB: 0.9801
- Performance with test data (Test R^2):
    - RF: 0.4348
    - XGB: 0.3959

6. Feature Importance

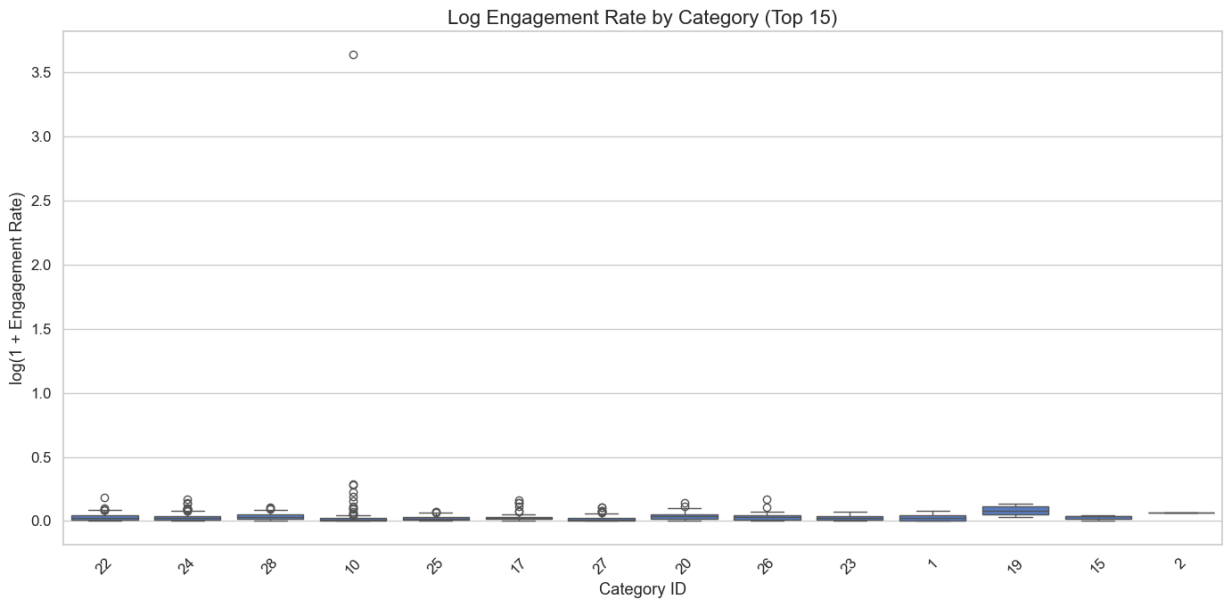
Description

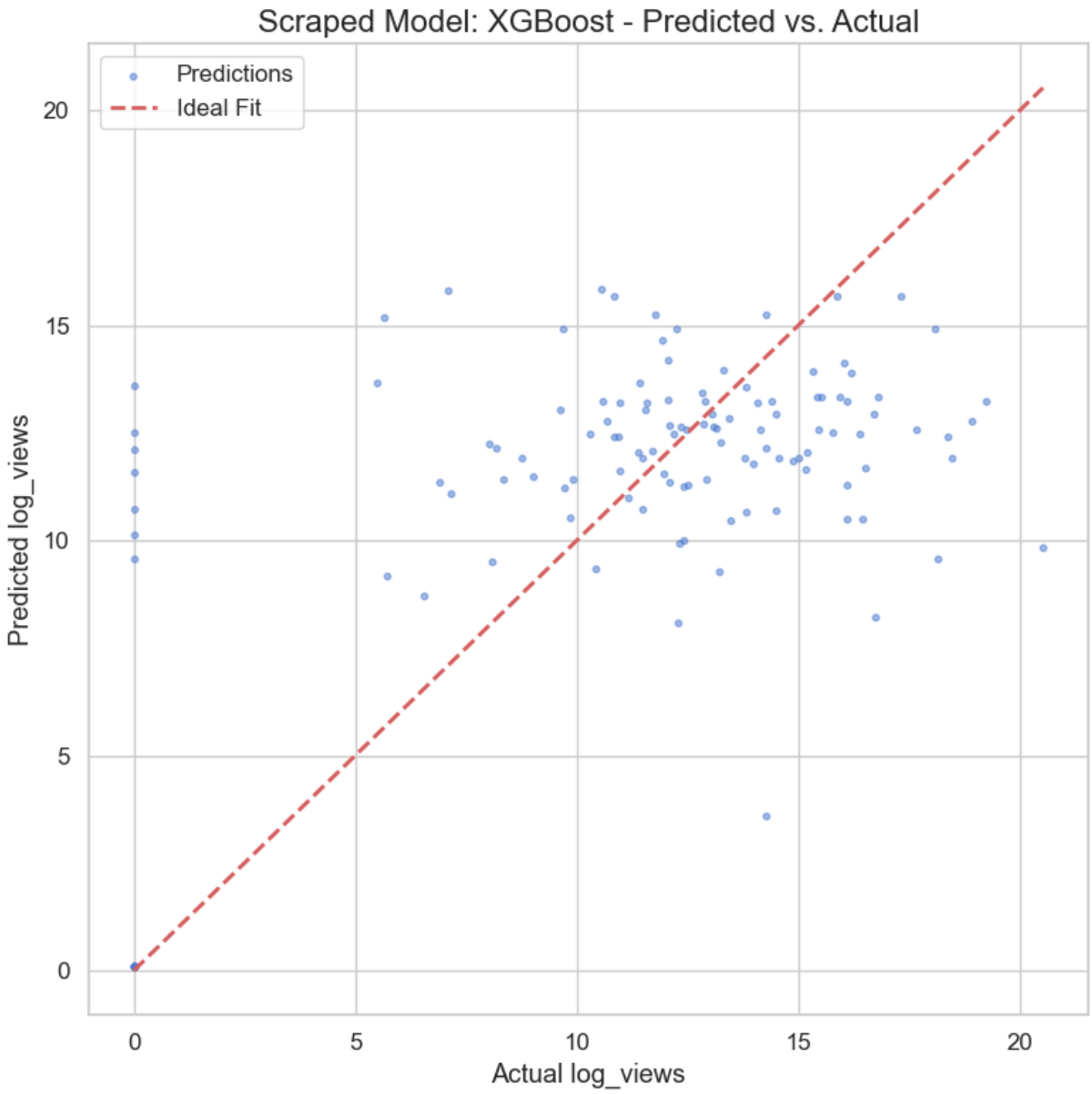
We used Permutation Importance (sklearn.inspection.permutation_importance) to identify the most important attributes. This technique measures a feature's importance by calculating the decrease in model performance when that feature's values are randomly

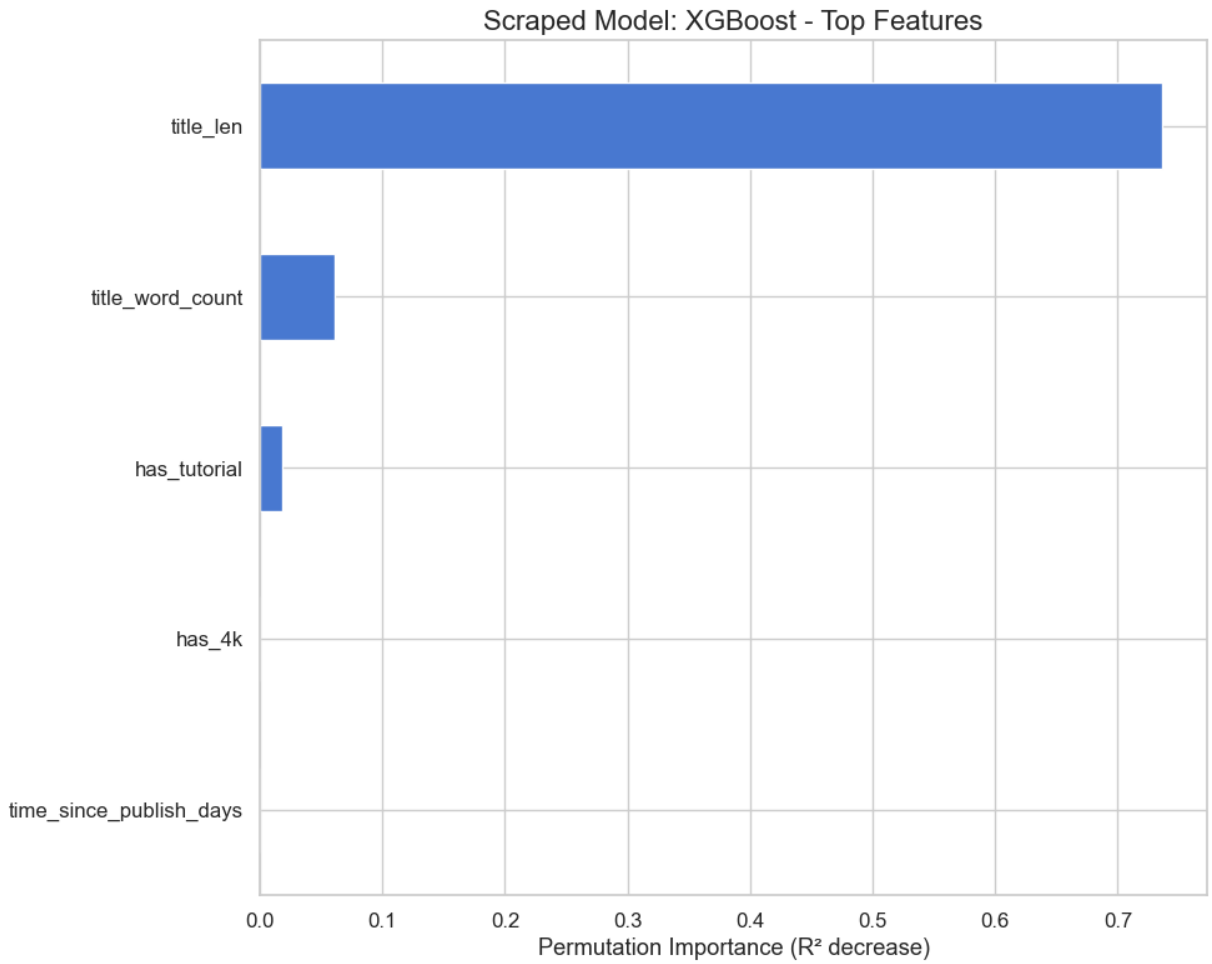
shuffled, breaking any relationship between the feature and the target. This is more reliable than the built-in .feature_importances_ method, especially when models are overfit.

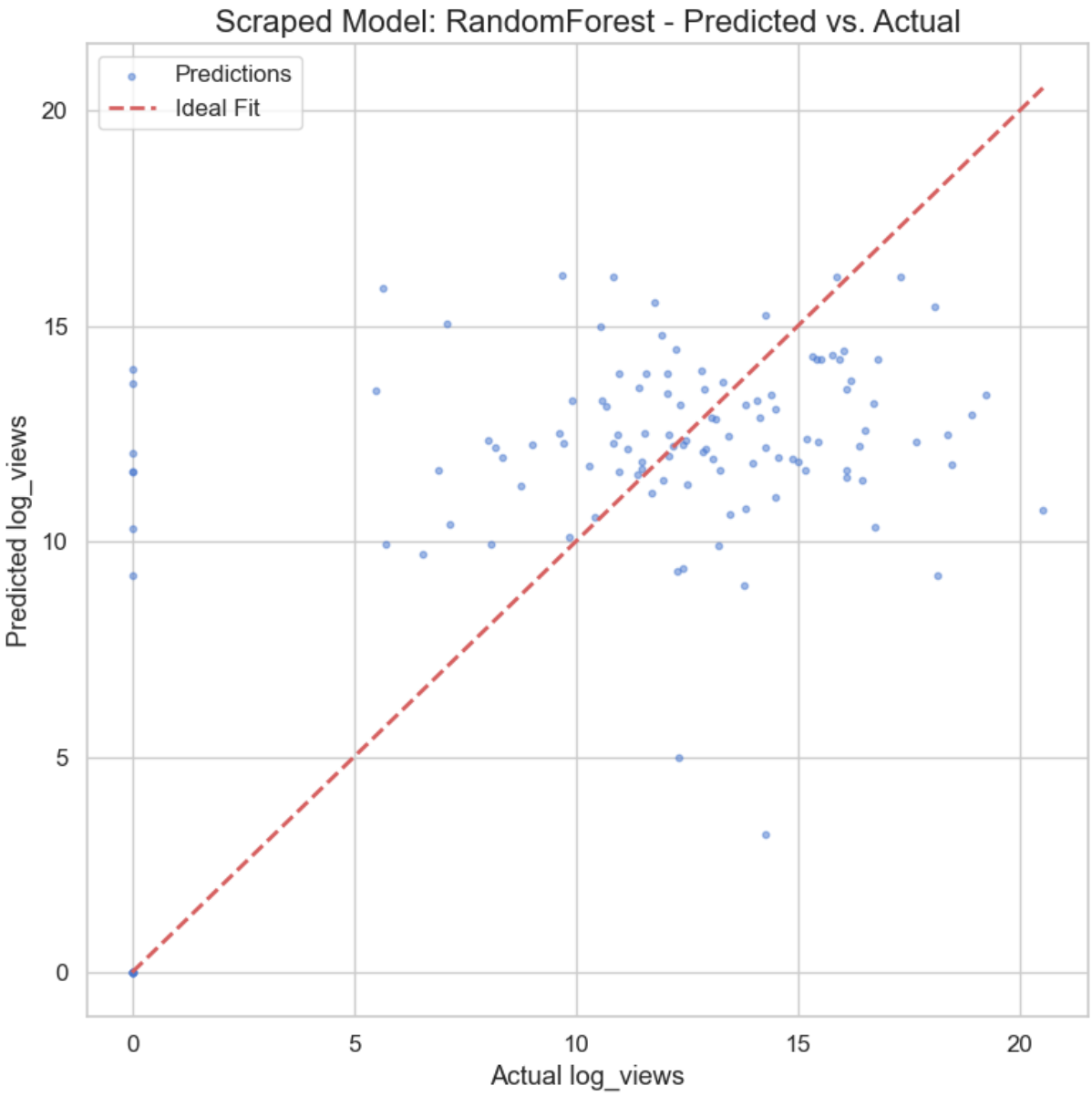
The results are saved in the viz/ folder. For the API model, time_since_publish_days and duration_min were consistently the most predictive features.

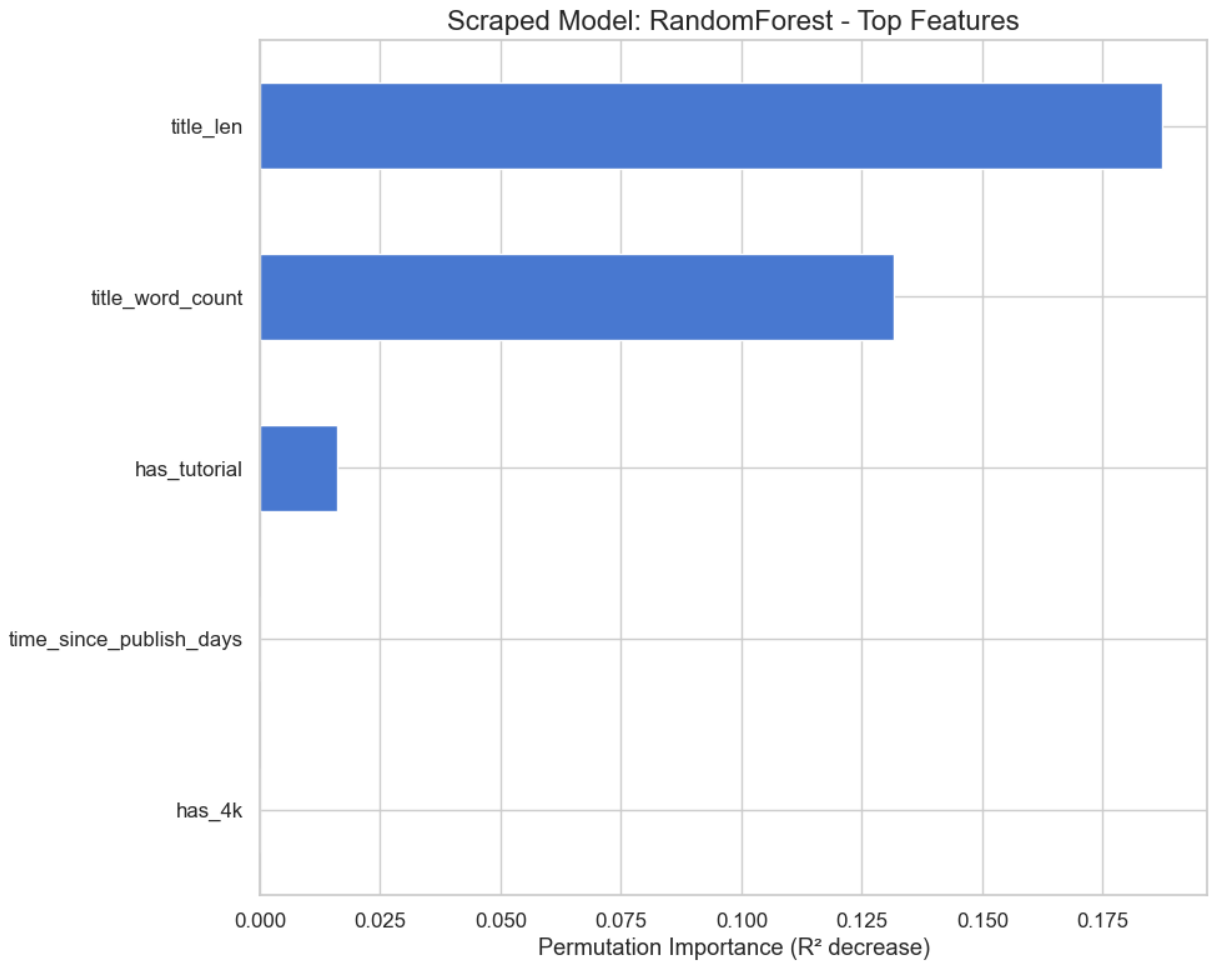
7. Visualization

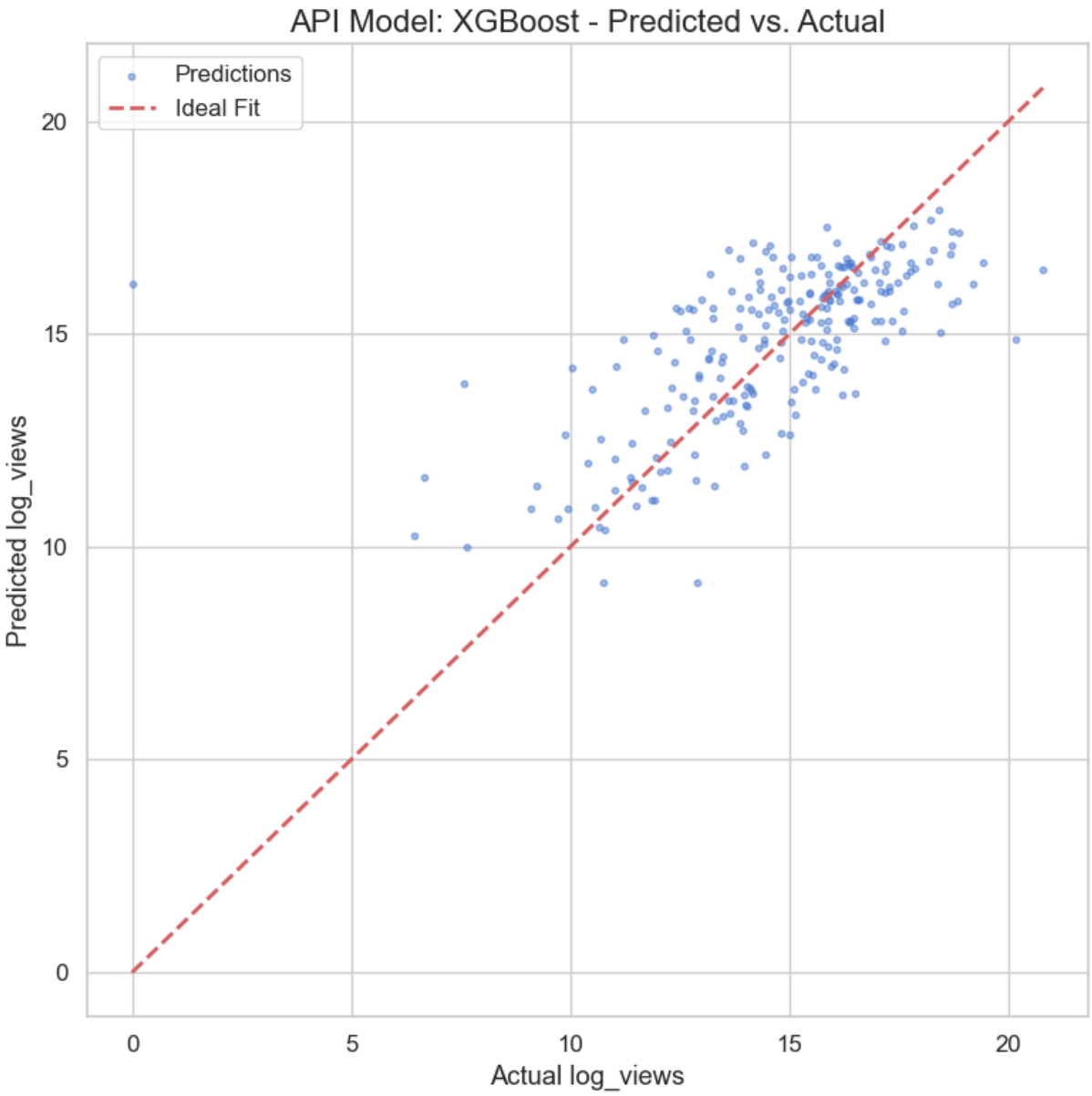

Log Engagement Rate by Category (Top 15)
log(1 + Engagement Rate)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
22
24
28
10
25
17
27
20
26
23
1
19
15
2
Category ID

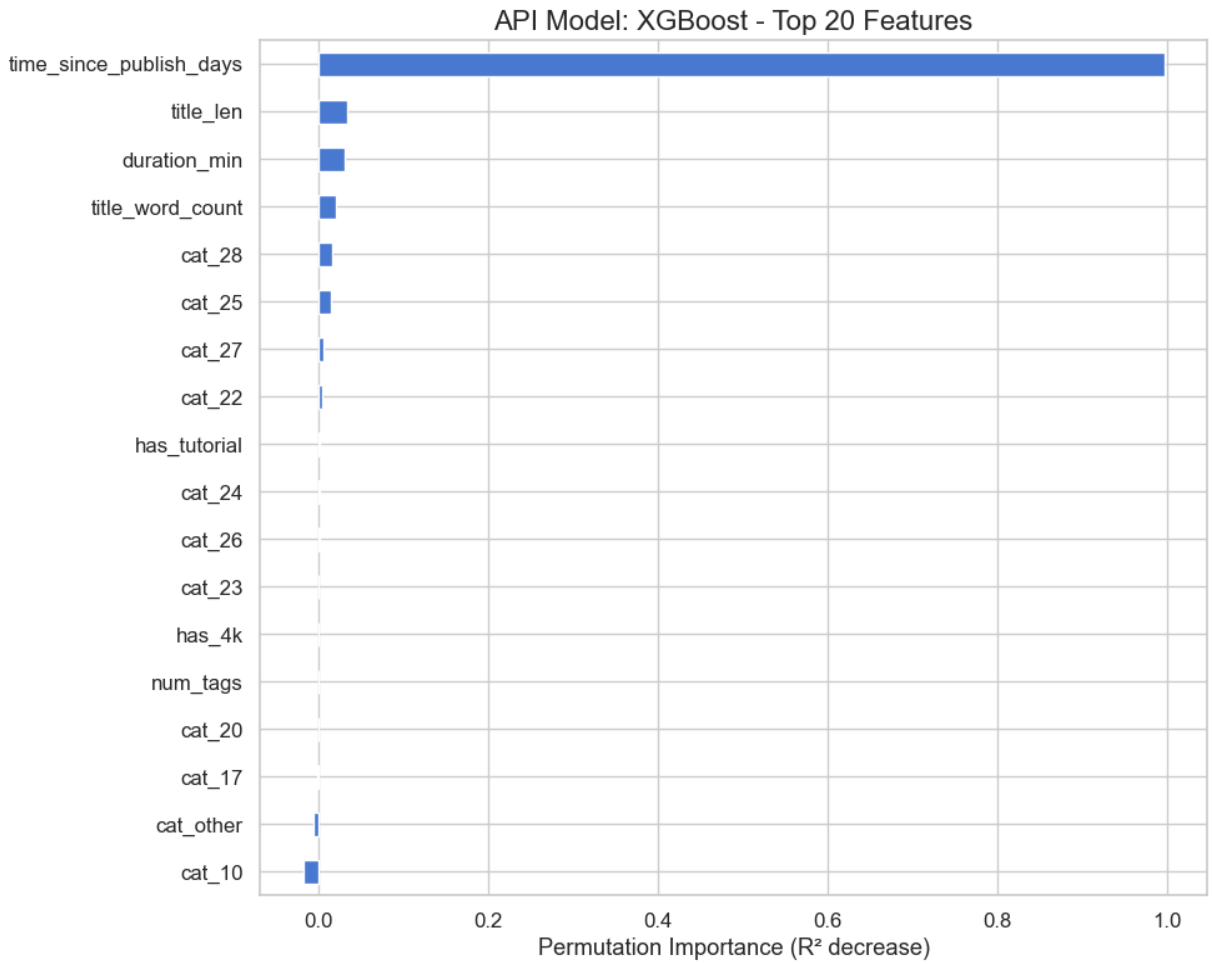

Scraped Model: XGBoost - Predicted vs. Actual
Predictions
Ideal Fit
Predicted log_views
Actual log_views
0
5
10
15
20

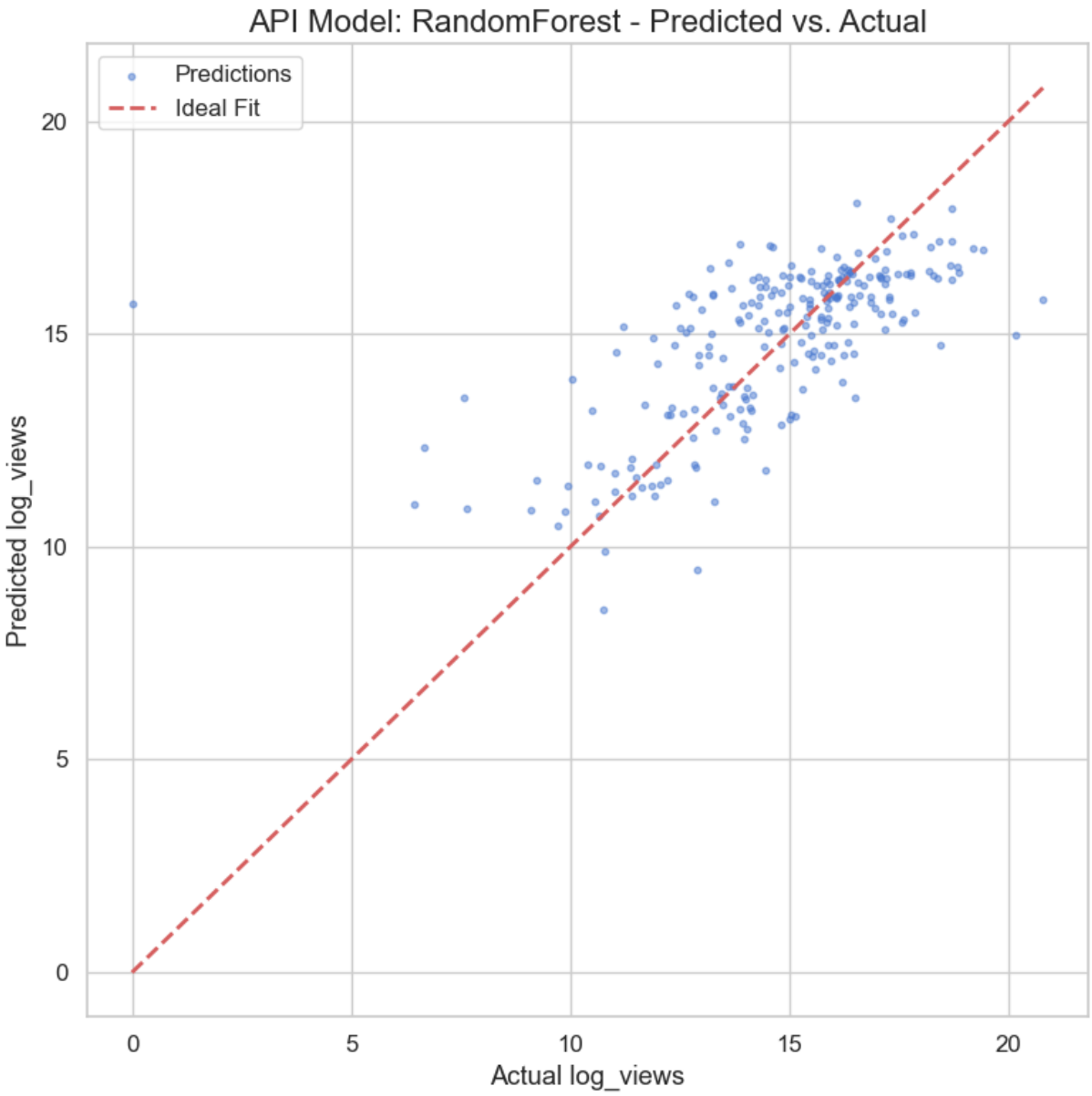

Scraped Model: XGBoost - Top Features
title_len
title_word_count
has_tutorial
has_4k
time_since_publish_days
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Permutation Importance (R² decrease)

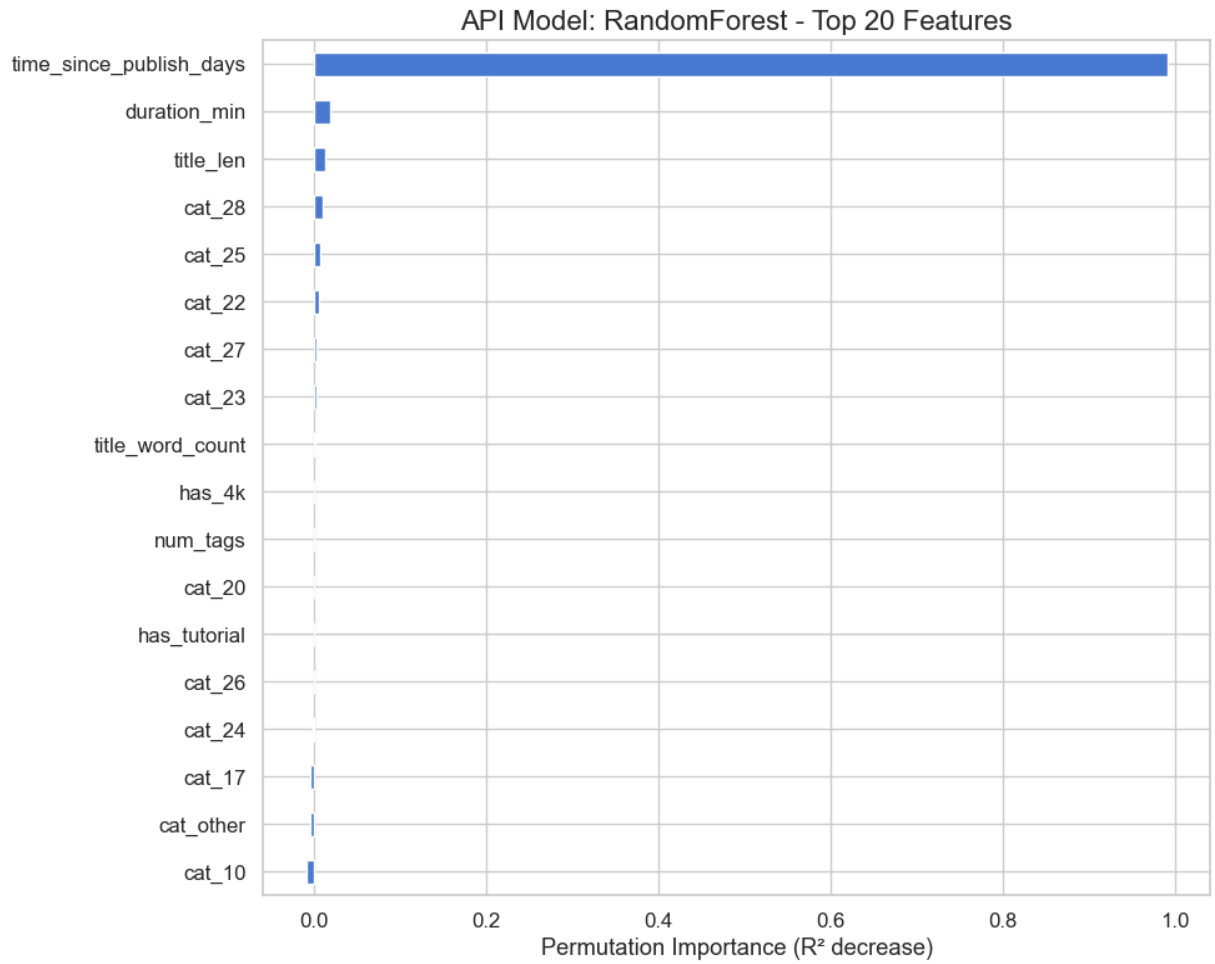

Scraped Model: RandomForest - Predicted vs. Actual
Predictions
Ideal Fit
Predicted log_views
Actual log_views
0
5
10
15
20


Scraped Model: RandomForest - Top Features
title_len
title_word_count
has_tutorial
time_since_publish_days
has_4k
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Permutation Importance (R² decrease)


API Model: XGBoost - Predicted vs. Actual
Predictions
Ideal Fit
Predicted log_views
Actual log_views
0
5
10
15
20


API Model: XGBoost - Top 20 Features
time_since_publish_days
title_len
duration_min
title_word_count
cat_28
cat_25
cat_27
cat_22
has_tutorial
cat_24
cat_26
cat_23
has_4k
num_tags
cat_20
cat_17
cat_other
cat_10
0.0
0.2
0.4
0.6
0.8
1.0
Permutation Importance (R² decrease)


API Model: RandomForest - Predicted vs. Actual
Predictions
Ideal Fit
Predicted log_views
Actual log_views
0
5
10
15
20



8. Discussion and Conclusions

Project findings

1. API Data is Significantly Better: The API model (Test R^2: 0.43) significantly outperformed the scraped model. This confirms that features like duration, num_tags, and category_id provide substantial predictive power that is impossible to obtain from scraping search results.

2. Models are Overfit: The API models, especially XGBoost (Train R^2: 0.98 vs. Test R^2: 0.40), were severely overfit. This means they memorized the training data rather than learning general patterns.

3. Popularity is Hard to Predict: A final R^2 score of ~0.43, even with rich API data, indicates that predicting a video's view count is extremely difficult. The most important features (age, duration) are known, but a large component of success is likely due to factors not in the data, such as thumbnail quality, channel reputation, and external promotion.

Challenges encountered

- API Quota: The primary challenge was the 10,000-unit daily API quota, which was exhausted quickly and limited the dataset to 1172 videos.
- Scraper Flakiness: The Selenium scraper was slow and prone to breaking if YouTube changed its HTML layout.
- Data Imbalance: The two collection methods yielded different numbers of videos (1172 vs. 608), complicating a direct comparison until the data was sampled.

Ethical and legal considerations

- Web Scraping: Scraping YouTube is a violation of its Terms of Service. The collector_scrape.py script was built for educational purposes only and should not be used at scale.

- API Usage: The collector_api.py script is the "correct," ethical, and legal method for data collection. Its use is governed by Google's API policies and limited by the quota system.

Recommendations for improving the performance

1. Reduce Overfitting: Use hyperparameter tuning (e.g., reduce max_depth, increase min_samples_leaf for RF/XGB) or apply regularization (e.g., L1/L2 for a linear model).
2. Add Channel-Level Data: Use the API to get data for each video's channel (like subscriberCount and videoCount), which are likely strong predictors.
3. NLP on Titles/Descriptions: Use TF-IDF or embedding models (like BERT) on the video titles and descriptions to extract features about the video's topic and sentiment, rather than just word counts.
4. Gather More Data: Run the API collector over several days to build a larger, more robust dataset.