CMPSC 455 Project 1

YouTube Video Popularity Prediction and Engagement Analysis

10 / 22 / 2025

Andrew Herman

Dr. Yang

Table of Contents

Project Description

In this project, I developed and compared two linear regression models, each using datasets gathered from two different techniques. One model used data that was scraped from YouTube dynamic webpages. The second model used an API key that directly loaded YouTube video data into a csv file. Each model's goal was to predict a videos view-count, given metadata like upload date, video title, comment count, tags, etc. To build optimal model predictions, techniques like data preprocessing and feature engineering were used. The resulting accuracies were compared and visualized to determine which model was better suited to predict view-count.

How to Use

Before being able to train and use the model, there are necessary files that need to be run first and libraries to have installed.

- Libraries Needed:
  - Scraping Data: Selenium: WebdriverManager, os, googleapiclient
  - Reading and Writing Data: Pandas, csv
  - Preprocessing: datetime, isodate, time, numpy
  - Training Models: Scikit-learn
  - Visualization: Matplotlib, seaborn
- Gathering the Data
  - Scraped Data Steps
    - Run “GetThumbnailData.py” - Gathers as much data from YouTube’s page of search topic.
    - Run “GetSpecificData.py” - Loads an individual YouTube video and gathers more specific video data
    - Run “AssignCategories.py” - Defines which category a video belongs to.
  - Gather API Data Steps
    - Run “Get_Video_Ids.py” to get a videos’ id in YouTube url.
    - Run “Get_Video_Metadata.py” to get specific videos data.

Training

Both models use “train_test_split” with an 80% training and 20% testing ratio

In the “ModelDevelopment” directory, “API_Model” trains the API data. “Scraped_Model” trains the scraped data.

Inferencing

Both regression models use LinearRegression() to make predictions.

The same files from before, “API_Model” and “Scraped_Model” will run the line:
regressor.predict(X_test_selected) to define y_pred with predictions.

Data Collection

Used Tools

1. Web scraping
    a. Webdriver- This was the main library used to scrape YouTube video data. It begins by opening a YouTube page in chrome and gathering as many video ids as it could. Once it searches for the full visible page, the driver will execute a scrolling motion to load more videos. Once over 3,000 video ids were gathered, a second driver would open an individual YouTube video and do a similar scraping process to get metadata like comments and tags.
2. API
    a. Googleapiclient- After getting a YouTube data API key from google, googleapiclient can be used to connect to your API key. The first step is like the scraping method of only gathering video ids. Then a second run will open URL links with the found ids and get the videos metadata

Collected Attributes

1. Web scraping Attributes
    a. Title, URL, Views, Upload Date, Comments, First Tag
2. API Attributes
    a. Video ID, Title, Video Duration (seconds), Days Since Upload, Views, Likes, Comments, Channel Title, Subscribers

Number of Data Samples

Web = 3292 samples | **API**- 3644 samples

API Usage

I used Google YouTube API to request an API key for YouTubes data. Once processed, I defined the API key as “API_KEY” in my terminal and configured the PyCharm IDE with that variable. Now when the file is run, it can access the key and gather data with the connection provided by “googleapiclient”.

2 Sample Data After Preprocessing

Before Preprocessing	After Preprocessing
• Max = 612000000 • Min = 0 • Mean = 1388368.95 • Median = 82000.0 • Std = 11860308.69	• Max = 16.32 • Min = 2.77 • Mean = 11.19 • Median = 11.311 • Std = 2.77
• Max = 140199 • Min = 0 • Mean = 1157 • Median = 108.0 • Std = 5403.90	• Max = 11.85 • Min = 0.0 • Mean = 4.55 • Median = 4.69 • Std = 2.44

Data Preprocessing

Due to the inconsistency of collected data, many preprocessing steps were needed to develop an efficient and accurate machine learning model. Directly scraping HTML tags from a source page will result in many formatting issues. All values that were intended to be numerical were scraped as strings. For example, views and upload dates were stored as text because of the measurements that followed, such as “days ago”, “weeks ago”, “million”, “K”, etc. Once removed, the values were able to be converted into integers. Skewed and missing data causes errors in model development. To handle these cases, I began by visualizing each feature and determined that certain columns had curved relationships. This curved relationship provided me with the necessary information to preprocess values. Missing values were filled with the median value in that column because there is less influence on major outliers. The assumption that there must be homoscedasticity for each column when building a linear regression model meant log-transformation was needed to linearize the data. Before doing so, I noticed there were a few extreme outliers in view-count that heavily influenced the model, so I removed any rows that had views within the first and last 2% of the sorted data.

Feature Engineering

The lack of available data from scraping meant I needed to utilize features I could gather. Columns like title and video tags needed to be expanded than their literal strings. Applying label encoder or one hot encoder was not feasible when titles and tags contain thousands of unique words. The next best option was to use Sklearns “TfidfVectorizer” function to assign words a value between 0 and 1 relative to its frequency. Other numerical values like character-length and word-count could be extracted from title and tag text. To add more meaning to video titles and tags, I created three addition columns calculating their length and word count. These different metrics could potentially find information that was not easily visible. Model performance and interpretability could be enhanced if these features are selected. Lastly, I wanted to build a new column of which category a video belonged to, so I ran my “AssignCategory.py” file to assign a category to a video based off the title keywords. Once each video was assigned a class, I applied One-Hot encoder to convert the class into a numeric value.

3 Examples of Feature Engineering

3 Title Words After Vectorizing – title_android, title_arcade ,title_best

3 Tag Words After Vectorizing – firsttag_duty, firsttag_game, firsttag_games

3 Category After One-Hot Encoder – category_General_Gaming, category_Minecraft, category_Nintendo

Model Development and Evaluation

Model 1 – Scraped Data Model

1. Machine learning model – Linear Regression Model using Sklearn
2. Input to model

Data Type	Picked Features
Engagement Metrics	Comments, Upload Date, Tag length
Title Keywords	Android, Best, Century, Gameplay, Arcade, Mobile, Nintendo, PC, PS5, Switch, Trailer, Upcoming
Tag Keywords	2026, Duty, New, PlayStation, WildGamersK
Category	Minecraft, Mobile, VR, Nintendo, Rocket League

3. Size of train data = 2633 (80%)
4. Attributes to the machine learning model

Engineered Method	Attributes
One-Hot Encoding	Category
Tfidf-Vectorizer	Video Title and First Tag
Numerical Values	Views, Upload Date, Comments

5. Performance with training / test data

Training Data	MSE = 3.03 and R-Squared = 0.61
Test Data	MSE = 2.88 and R-Squared = 0.62

Model 2 – API Usage Model

1. Machine learning model – Linear Regression Model using Sklearn
2. Input to model

Data Type	Picked Features
Engagement Metrics	Likes, Comments, Subscribers, Duration
Channel Attributes	Channel Name Length

3. Size of train data = 2915 (80%)
4. Attributes to the machine learning model

Engineered Method	Attributes
Numerical Values	Views, Upload Date, Comments, Duration, Channel Name Length

5. Performance with training / test data

Training Data	MSE = 1.34 and R-Squared = 0.91
Test Data	MSE = 0.97 and R-Squared = 0.93

Feature Importance

How Features Were Selected

Now that the data was preprocessed and ready to be input into a model for training, some columns were more meaningful for the models' predictions. As previously stated, title and tags were converted into vectorized values based on their frequency, and One-Hot label encoder was applied to categories. This created an additional 61 columns, increasing the risk overfitting.

I applied Recursive Feature Elimination on the trained data to remove the least significant features. For the scraped-data model, I kept the top 25 most meaningful features. This value is higher than a typical number of kept features because of the additional vectors and one-hot label columns. For the API model, I was able to find and use the top 5 features.

Top Features Used

Scraped Data Features

- Comment count
- Titles that included the word “Android”
- Video Tags that included the year “2025”
- Videos under the “Minecraft” category

API Selected Features

- Likes
- Comments
- Subscribers
- Video Duration
- Channel Name

Evaluation and Visualization

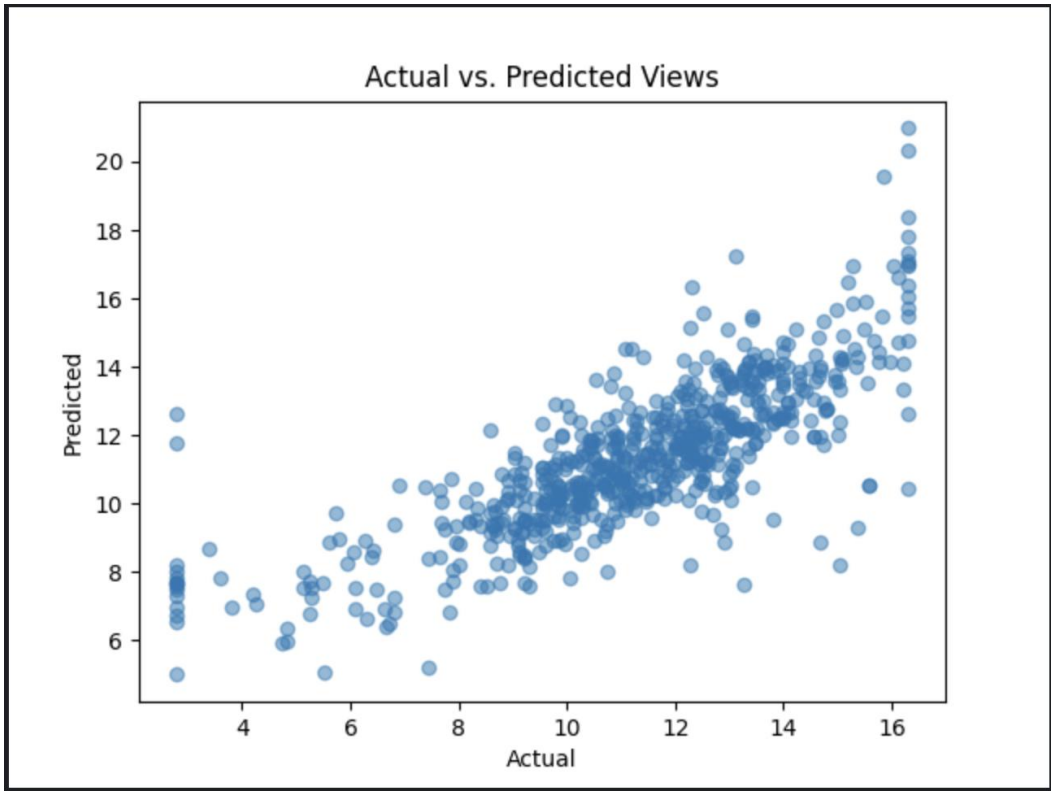
Model Accuracy Comparison

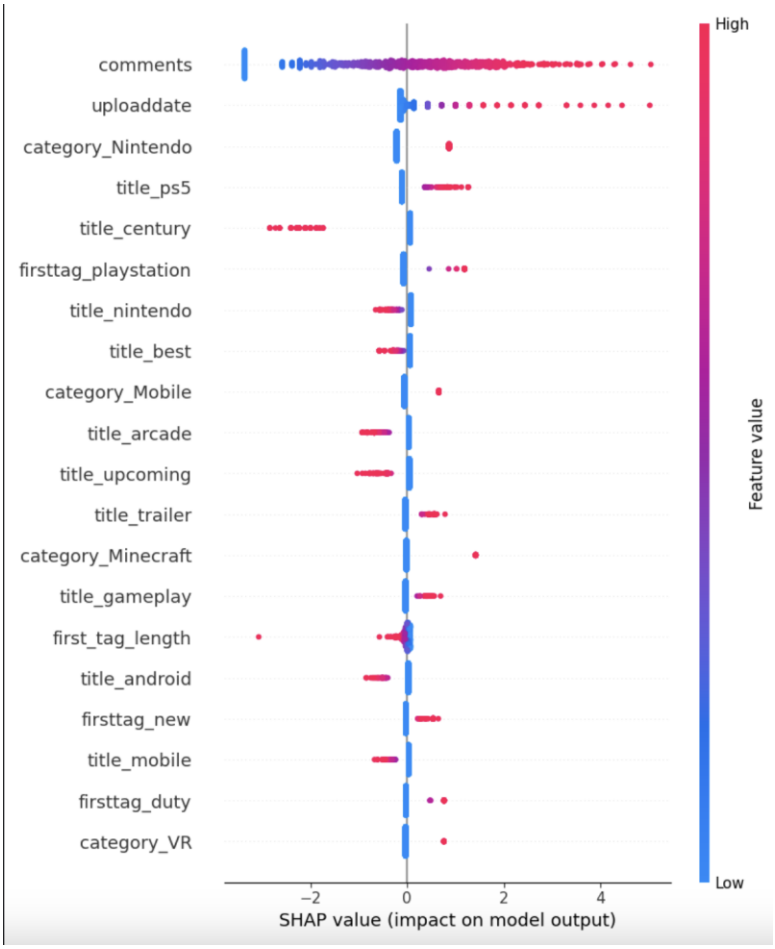
Model	Accuracy Score
Web Scraping Model	0.61
API Model	0.91

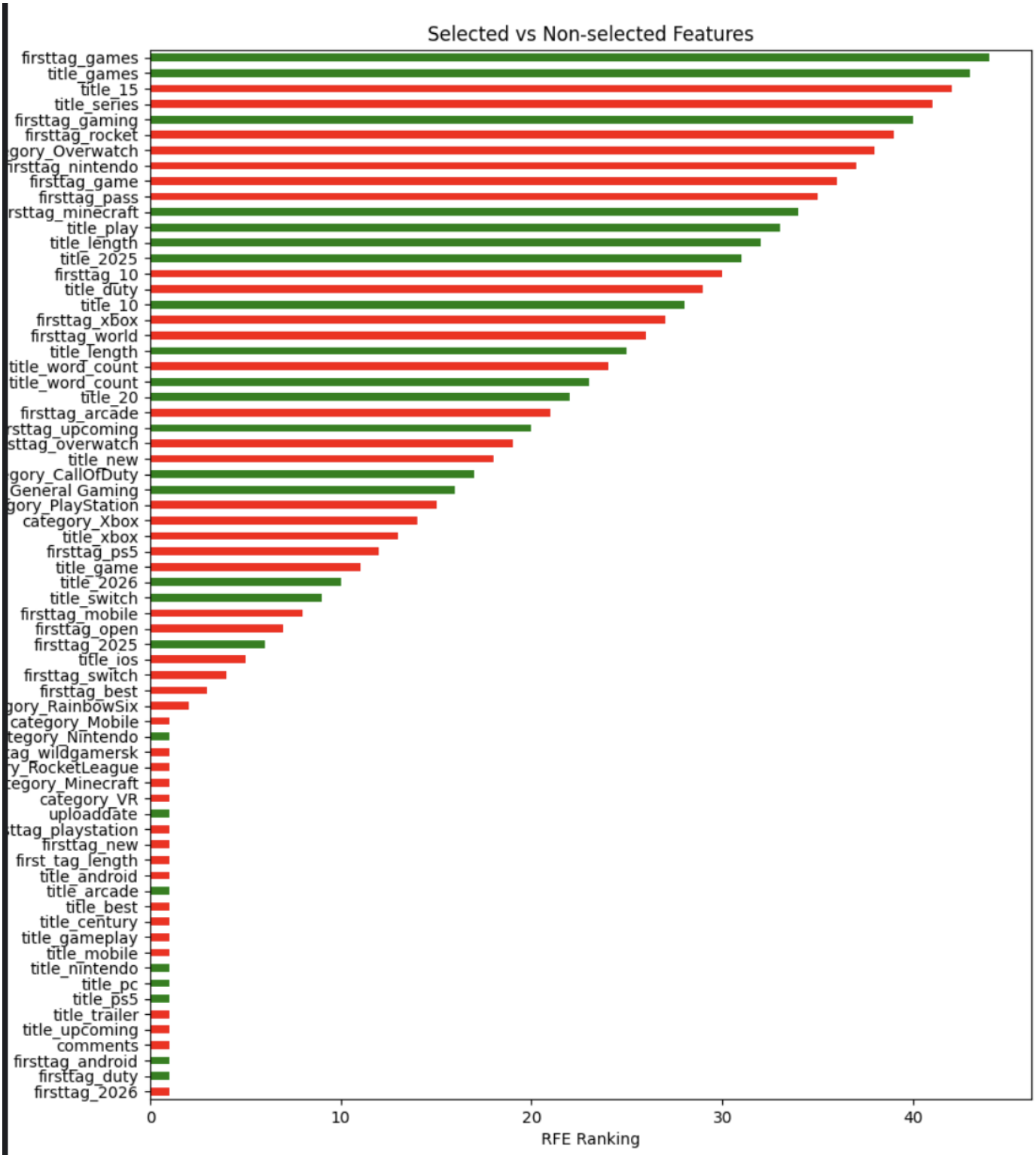
The API model had a larger prediction accuracy with 91%, beating the Web Scraping Model of 61%. Other metrics for comparison are the API keys lower Mean-Squared-Error and higher R-squared value. The API model had a lower error and was able to explain a larger proportion of the variance in views.

Graph Visuals

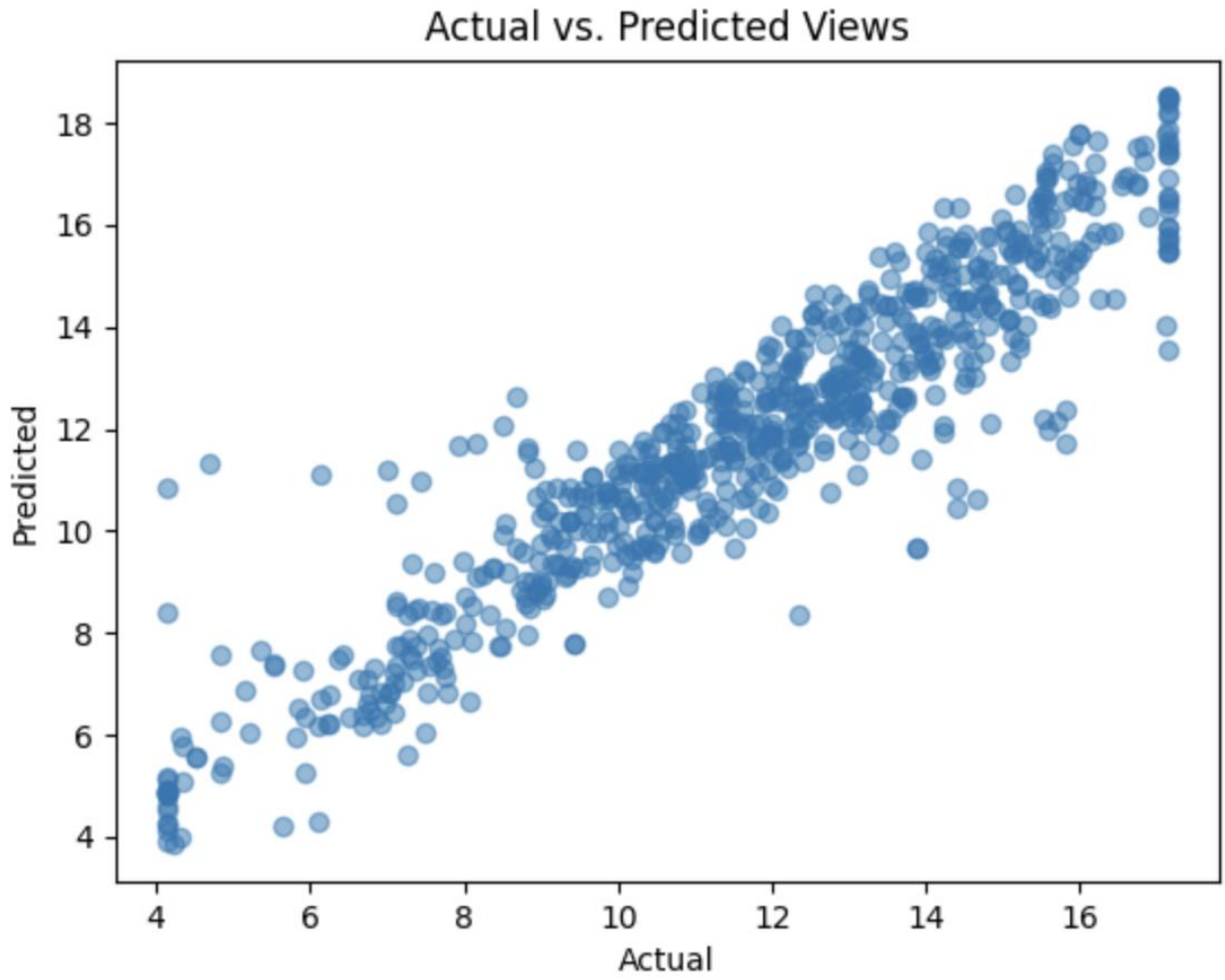
Web Scraping Model

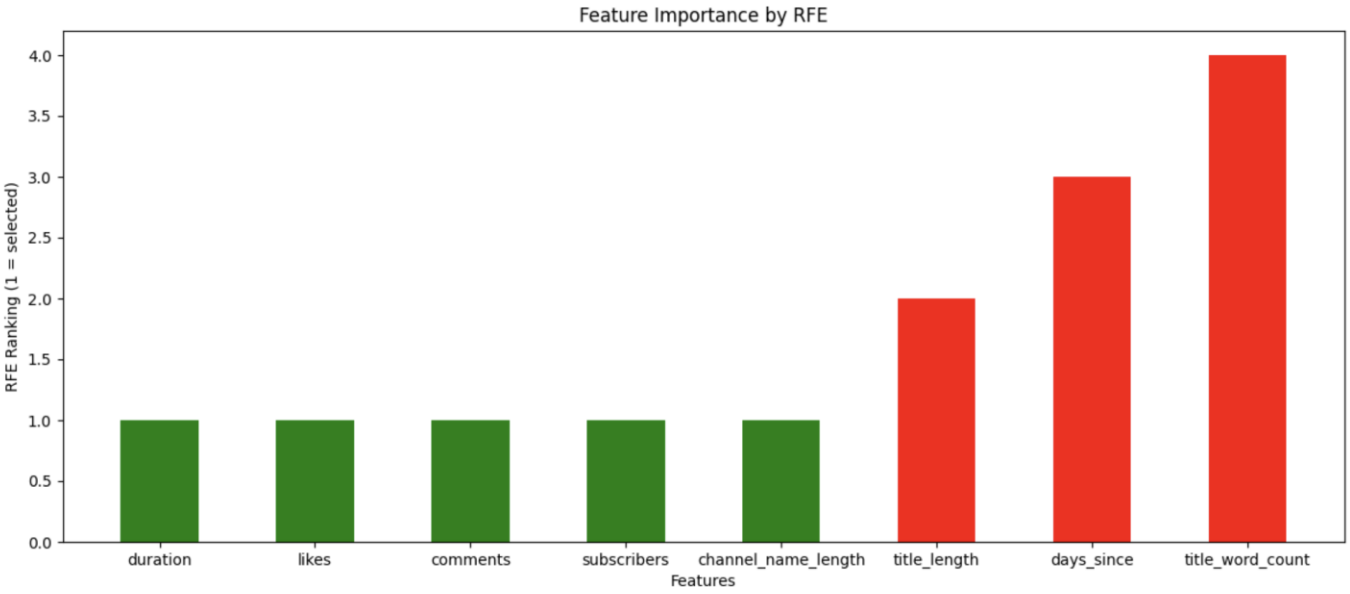





Selected vs Non-selected Features
firsttag_games
title_games
title_15
title_series
firsttag_gaming
firsttag_rocket
gory_Overwatch
irsttag_nintendo
firsttag_game
firsttag_pass
rsttag_minecraft
title_play
title_length
title_2025
firsttag_10
title_duty
title_10
firsttag_xbox
firsttag_world
title_length
title_word_count
title_word_count
title_20
firsttag_arcade
rsttag_upcoming
sttag_overwatch
title_new
gory_CallOfDuty
General Gaming
gory_PlayStation
category_Xbox
title_xbox
firsttag_ps5
title_game
title_2026
title_switch
firsttag_mobile
firsttag_open
firsttag_2025
title_ios
firsttag_switch
firsttag_best
gory_RainbowSix
category_Mobile
tegory_Nintendo
tag_wildgamersk
y_RocketLeague
tegory_Minecraft
category_VR
uploaddate
ttag_playstation
firsttag_new
first_tag_length
title_android
title_arcade
title_best
title_century
title_gameplay
title_mobile
title_nintendo
title_pc
title_ps5
title_trailer
title_upcoming
comments
firsttag_android
firsttag_duty
firsttag_2026
0
10
20
30
40
RFE Ranking

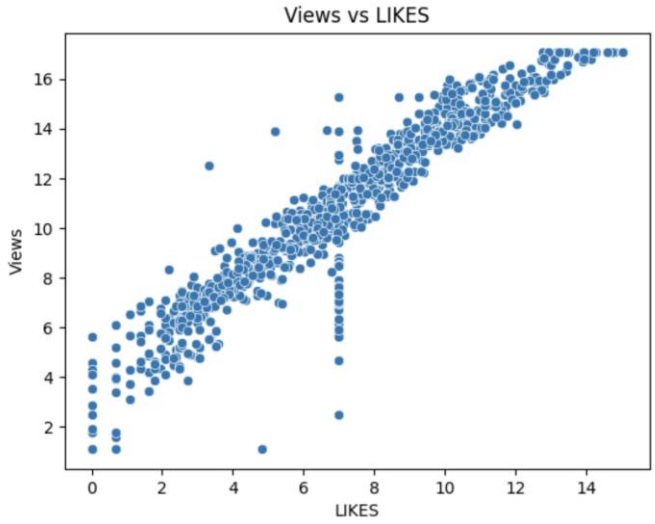
API Model

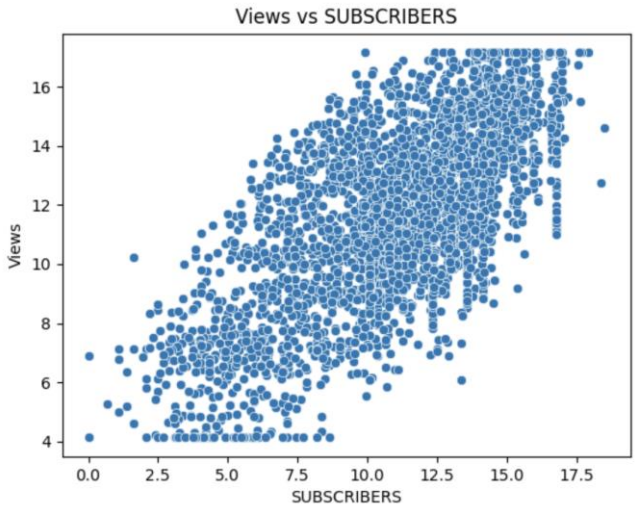

Actual vs. Predicted Views
Predicted
Actual
4
6
8
10
12
14
16
18

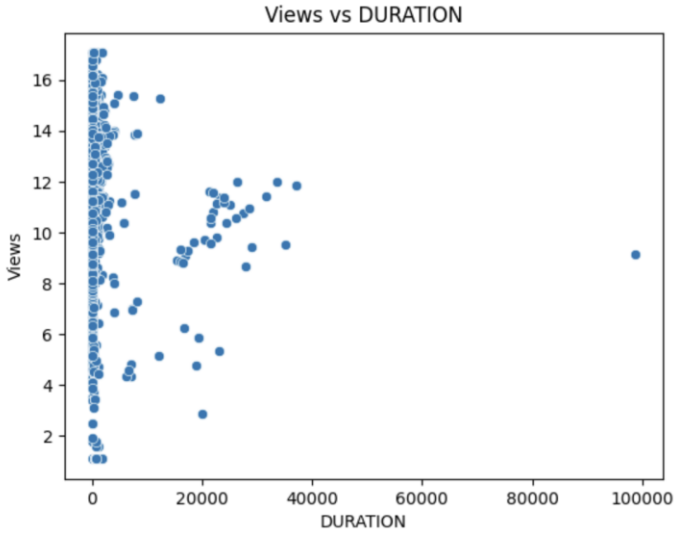

Feature Importance by RFE
RFE Ranking (1 = selected)
Features
duration
likes
comments
subscribers
channel_name_length
title_length
days_since
title_word_count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

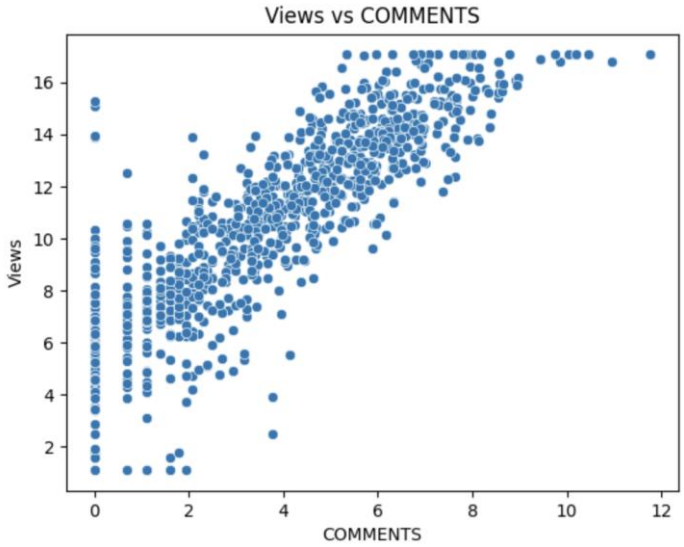
Engagement Trends

Likes – Subscribers – Duration – Comments









Discussion and Conclusion

Project Findings

After fully implementing both linear regression models using scraped versus API key data, it is no surprise that the API key model performed better in all aspects. The mean squared error was smaller by 1.91 mean squared error, and the R-squared value was greater by 31%. This enhancement was due to the ability to gather more meaningful data from the API key, such as “Video Likes” and “Channel Name”. After applying feature selection, I determined that the most meaningful features were engagement factors and channel names. This makes sense because videos with more likes and comments enhance a video's exposure, leading to more views. Channel name is a YouTubers version of name-brand product, and those who are more popular will get more views.

Challenges with Project

The main challenges I faced with this project were scraping dynamic data, limited daily data imports using an API key, and deciding which preprocessing methods to use. Unlike previous assignments, YouTube is a dynamic website, making the WebDriver library necessary. One full screen of YouTube data only contains a limited number of videos, so I had to apply scrolling and moving to new pages in the automation. Secondly, Google’s YouTube-API limits users to 10,000 exports a day. Although a large number of programming mistakes and tests caused me to use this limit and had to wait until the next day to try again. Lastly, to determine which preprocessing methods to use, I had to evaluate and visualize the raw data. I attempted multiple methods until I found those most suitable for the model.

Ethical and Legal Considerations

The main ethical and legal consideration with the assignment is taking a channel's information without permission. But since this is for research and educational purposes with no intent to make money, privacy and ethical concerns are minimal.

Model Improvement Recommendations

There are many ways I can improve my models and collect data next time. Starting with feature selection, I can change which model I use in the RFE function. Instead of a linear regression model, I should have used the RandomForestRegressor model. This would have provided a stronger method of getting important features. Secondly, I should have built functions that were recallable instead of letting the program go line by line to create the model. This would help with readability and efficiency when running files.